

# Maya® Character Creation

MODELING AND ANIMATION CONTROLS

Chris Maraffi

88  
TZAC  
3958



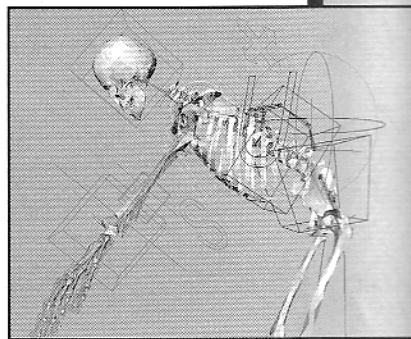
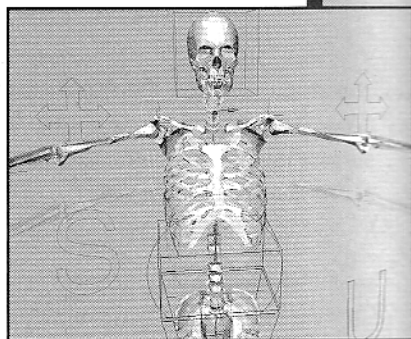
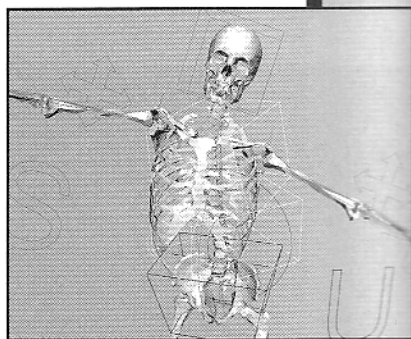
VOICES THAT MATTER



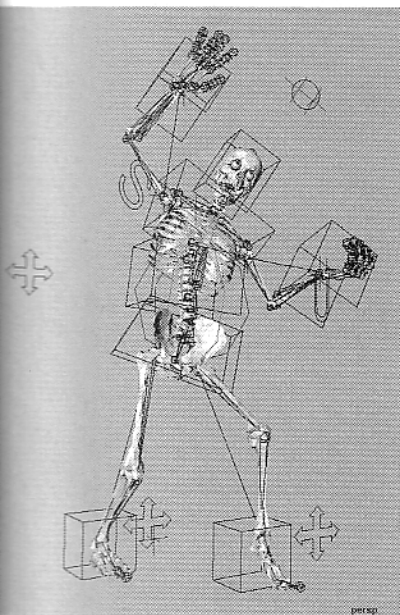
## CHARACTER SKELETON SETUP

**T**HIS CHAPTER SHOWS YOU how to create complex controls for animating each of the main parts of your character model. Creating such controls involves many of the tasks that a character setup artist does on a daily basis, including such things as drawing skeletons, creating *Inverse Kinematics* (IK) handles, constraining objects, using control icons, and parenting objects into a complex hierarchy. After completing this chapter, you should have a good understanding of all the basic techniques that a character setup artist frequently uses to create good character controls.

The skeleton controls shown in this chapter create a good general-purpose character rig (see Figure 3.1). The basic hierarchy was based on a rig shown by a Blue Sky animator who taught at the School of Visual Arts back in 1996. (I have unfortunately forgotten that animator's name.) I've been developing and adding to the rig functionality since that time. Although nothing on this rig was taken directly from any other rig, ideas for controls have come from a variety of sources. I have gotten inspiration from colleagues, students, industry presentations, and Alias|Wavefront master classes by people such as Jason Schleifer at SIGGRAPH (who used controls similar to the ones shown for the advanced backbone on the *Lord of the Rings* characters). The complete rig presented in this chapter incorporates a sampling of all the basic kinds of controls that you would be required to create for a production-ready character.







1 In this chapter, you create a general-purpose rig with controls for each part of your character's body.

Keep in mind that there is no perfect all-purpose skeleton rig that will work well in all situations. In a real production, character setup artists create rigs for particular purposes. It is not uncommon for the main character in an animation to have a variety of rigs, with controls designed for particular actions in a scene. For instance, there might be separate rigs for walking, tumbling, lifting, and lip-syncing. However, an understanding of how to build all the controls in a general-purpose rig better prepares you to create more production-specific rigs.

## CREATING BASIC ANIMATION CONTROLS

This section shows you how to create basic skeleton controls for your character. You learn to draw skeletons with precisely placed pivot points, so that the joints rotate accurately. Polygon reference bones are used as guides for drawing Maya skeletons, and are later parented to the skeletons to use as animation reference. Drawing, manipulating, and editing skeletons is covered in detail. Then, a rigging method that involves adding curve-based control icons is shown to make it easier to manipulate the skeletons. Finally, all skeletons and control icons are parented into a hierarchy used as a functional character control rig.

## UNDERSTANDING SKELETAL ANATOMY

Creating believable character motion requires that you create controls based on how real bodies work. Therefore, you need to place carefully the joints in your models so that the skin bends like it has a real skeleton inside of it. Your skeleton creates a 3D structure for your body, and enables you to move around. Without a skeleton, your body would be just a flat lump of muscle and skin. You wouldn't even be able to stand up! Skeletal joints are therefore very important for defining how your character moves.

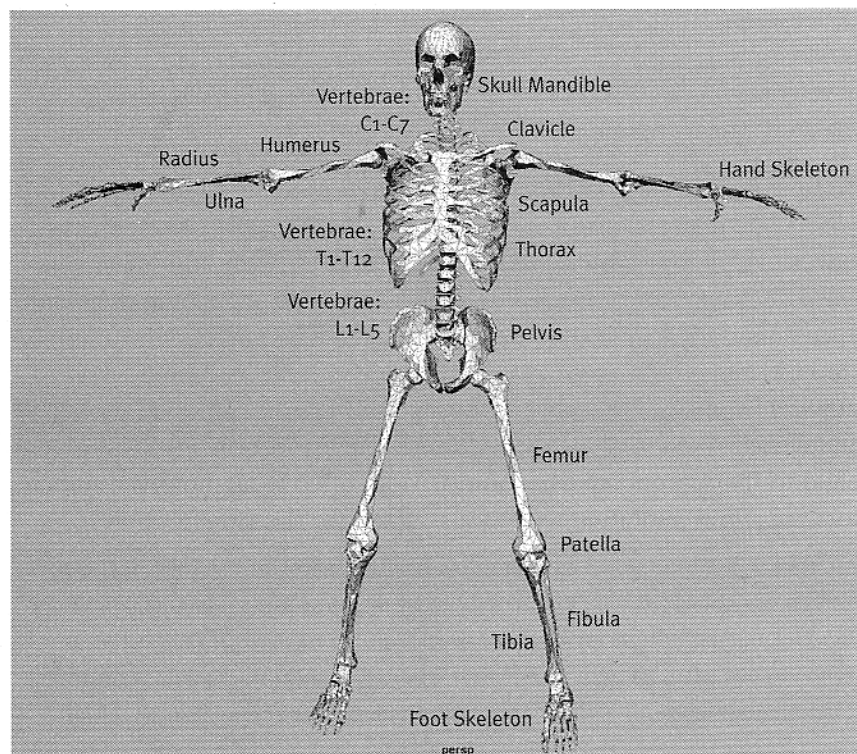
Be aware, however, that some important differences exist between how a real organic body works and how a computer-generated 3D character is set up. In a real body, muscles contract and stretch to move the bones in a skeleton. Normally this process works in the reverse way for a 3D character, where the bones are the main movers, and the muscles deform in response. Because both work together very closely, the difference is hardly noticeable, but doing it this way makes setting up the character a lot easier.



Another important difference is that you can simplify some areas of the body in a 3D character. The backbone, for instance, doesn't usually require the number of bones in a real backbone. Because Maya smoothly blends the effects of the bones on the deformed skin, using fewer joints is usually desirable, because the use of fewer joints simplifies your setup tasks.

It is a good idea for anyone who wants to design, model, or set up 3D characters to have a good understanding of the skeletal anatomy of the human body (see Figure 3.2). Most characters are based on the same basic structure. Even other mammals, such as horses and dogs, have a similar structure. One of the best references a character artist can have is a variety of anatomy books to use when creating 3D characters.

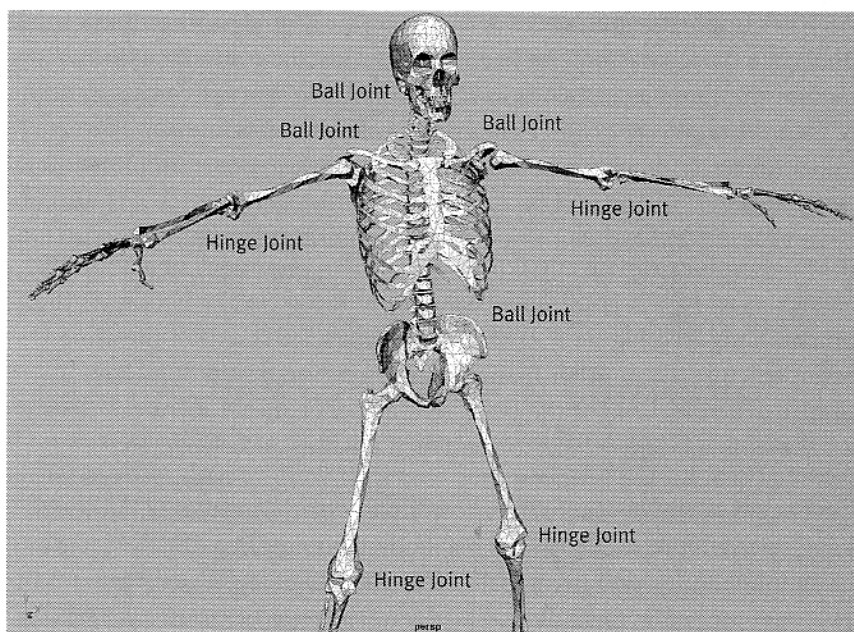
One thing to notice carefully is how a particular joint moves. Although joints have a lot of subtleties in a real body, the joints in a 3D character can be simplified into two kinds: ball joints and hinge joints (see Figure 3.3). *Ball joints* can rotate in all three directions, whereas *hinge joints* can rotate in only one direction. Examples of ball joints can be found in the backbone, shoulder, and legs where they attach to the hips. Examples of hinge joints can be found in the elbows and knees.



3.2 Character setup artists must have a good understanding of skeletal anatomy.



- 3.3 The two main kinds of joints in the human body are ball joints and hinge joints.



## PLACING POLYGON REFERENCE BONES

One thing that many animators use today is a low-resolution reference skeleton—a 3D skeleton made from polygons or NURBS bones—as a guide for drawing Maya skeletons. The advantage to using a 3D skeleton as reference is that you can see exactly where the joints are, which makes it easier to place your pivot points correctly.

Toward the end of this chapter, you make the reference bones child to your Maya joints so that it is easier to see whether your skeleton controls work properly. You can then hide your deforming skin when you are ready to animate, to speed up your system, and show only the reference bones while animating. Traditionally, animators often used a nondeforming proxy of their skin to animate in real time. To do this, you duplicate your skin models, convert them to low-resolution polygons, and detach the skin at the skeletal joints. Then you make the skin pieces child to the appropriate Maya joints. More often today, however, animators use a low-resolution reference skeleton for the same purpose.

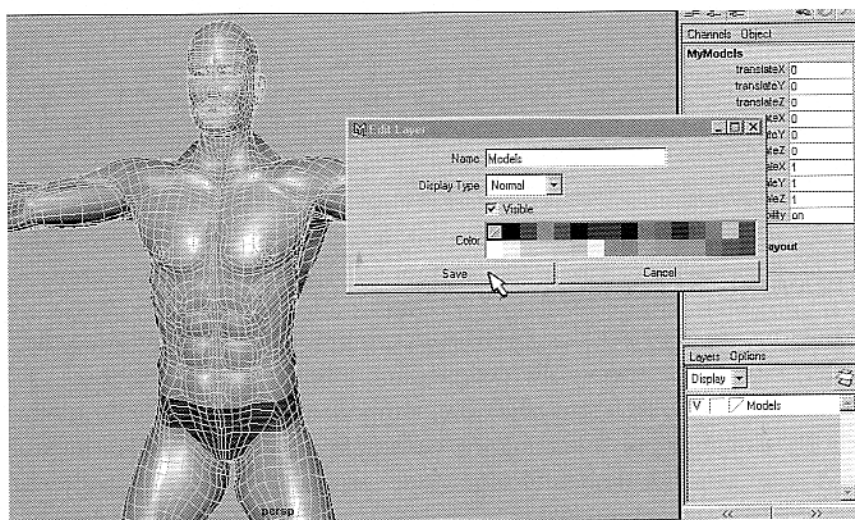


## EXERCISE 3.1

## PLACING POLYGON REFERENCE BONES

Now that you have your model finished, it's time to start creating controls for animating your character. In this exercise, you create a polygon reference skeleton that fits correctly inside your character's skin.

1. Open the scene that contains your character model from the previous chapters. Make sure the scene contains only your final character models, and doesn't contain any creation curves, instances, or history connections. Your models don't need to be in any particular hierarchy, but you do want to organize them so that you can display or hide them easily. To do this, group your models under a node that you name **MyModels**. Then create a new layer by clicking the Create a New Layer icon in the Layer Editor and name the layer **Models** by double-clicking it to enter the new name in the layer options box. Select the MyModels group node, and place it on the layer by right-clicking the layer, and choose Add Selected Objects (see Figure 3.4). Finally, import into your scene the ReferenceBones.mb file from the download site.



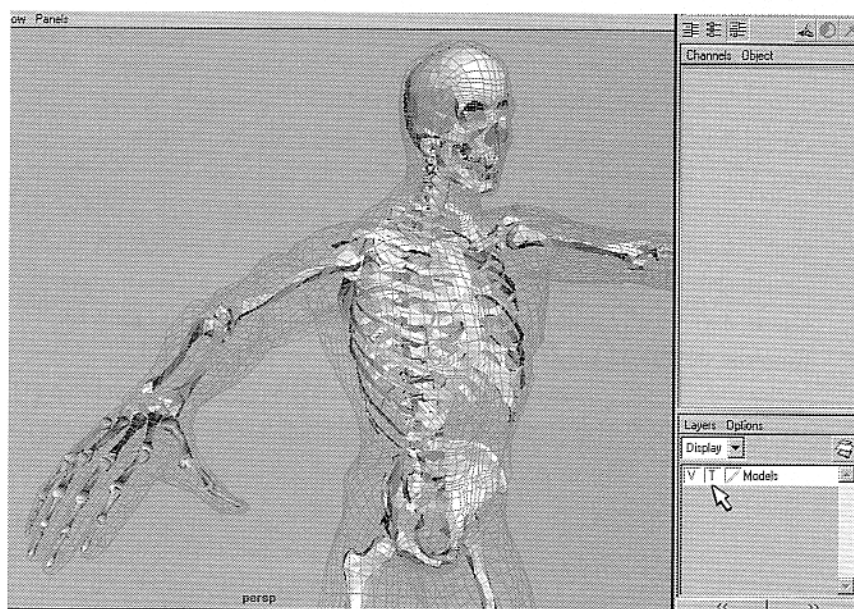
3.4 Create a new layer for your models.

2. After you import the file that contains the polygon reference bones, set the Models layer to Template by clicking the layer box until a *T* appears. Template enables you to use your models as a guide for placing the reference bones. You want to manipulate the transforms on the reference bones to make them fit inside your model (see Figure 3.5). To make the polygon bones fit well, you will probably have to manipulate their components to



some degree. You can use all the tools you used in Chapter 2, “Modeling the Skin of a Biped Character,” to refine your models, including lattice deformers. You may also want to create special features if your character model is more surreal. If your character is a devil, for instance, you can pull out horns on the skull bone. Or if it has wings and a tail, duplicate some of the bones from the backbone or arms and make wing and tail bones. ■

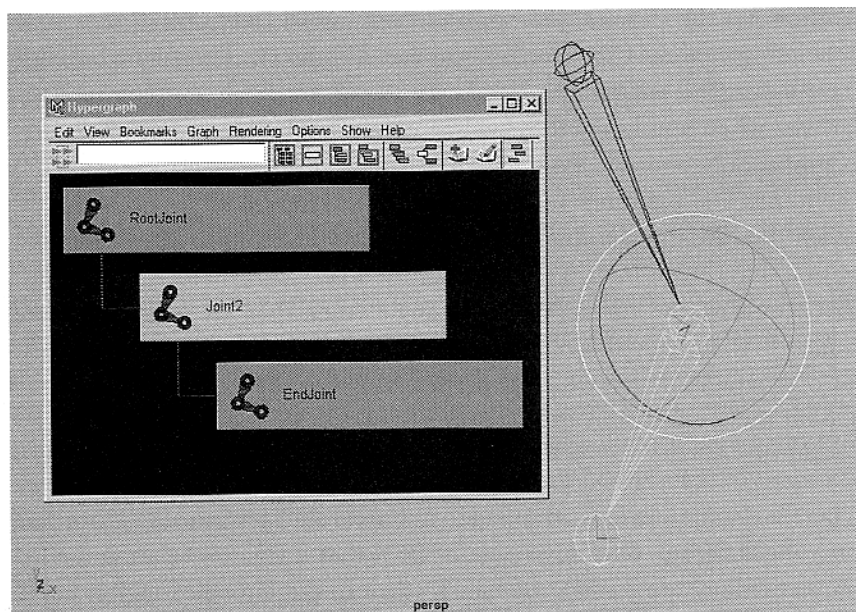
- 3.5 Template your model, and use it as a guide for placing your polygon reference bones.



## DRAWING SKELETONS

Skeletons are a special kind of deformer found in the Animation module (press F2) and are specifically designed for animating characters. Like other deformers, skeletons affect the component structure of your models. By assigning and animating the skeletons, vertices on the skin move, and your character models change shape over time. Skeletons usually have length, which you create by drawing a skeleton from point A to point B. Most skeletons have at least two joints: a root joint and an end joint. A bone connects each joint. Although you can create single-joint skeletons, multiple-joint skeletons are most common in characters.

The most basic way to manipulate skeletons is to rotate their joints, which is called *Forward Kinematics* (FK) (see Figure 3.6). It is not desirable to translate any joints other than the root joint. Translating a joint in the skeleton chain causes the previous joint's center to no longer be oriented down the length of the bone, which can cause rotation problems on your

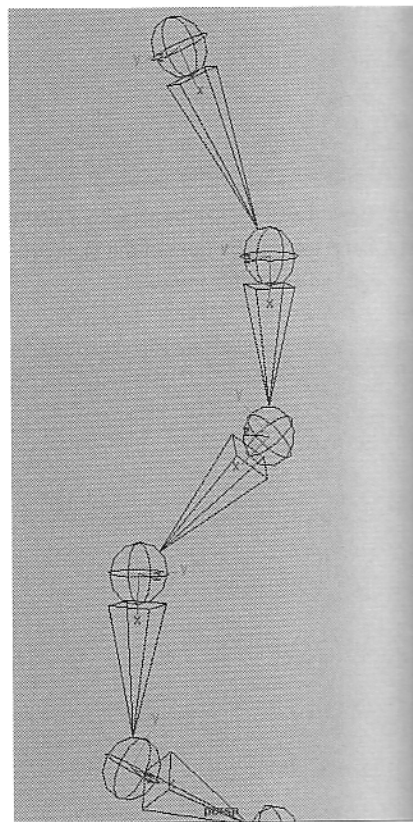


3.6 The most basic way to animate a skeleton, called Forward Kinematics, is to rotate the joints.

controls. By rotating the joints, you can avoid this problem. Rotating the joints also enables you to animate the skeletons to bend in any direction. As the name implies, you animate with FK by starting at the root joint, and progressively rotate each joint down the skeleton chain.

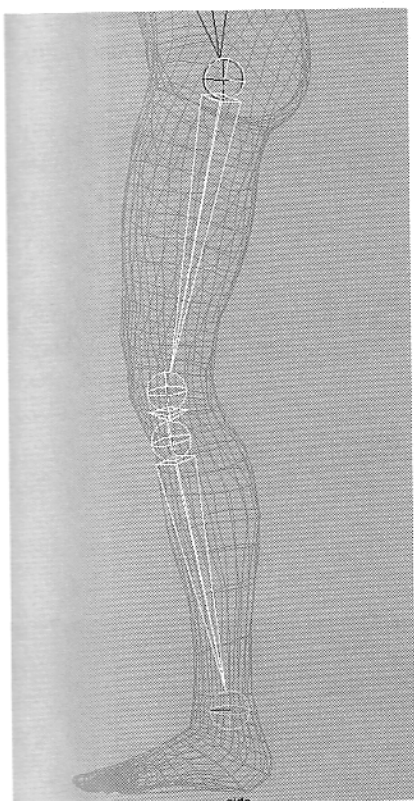
The other way to manipulate skeletons is to use *Inverse Kinematics* (IK), which constrains the skeleton to bend in a single direction by assigning it an IK solver. You manipulate the skeleton by translating an IK handle, which is created when you assign the solver. Translating the handle causes all the joints to rotate that are constrained by the solver. Usually the IK handle is on the last joint in the skeleton chain, so that translating it affects the joints higher up in the chain—hence the name Inverse Kinematics.

Understanding skeletons is important if you want to create effective character controls. If you display the center on a skeleton joint by choosing Display, Component Display, Local Rotation, notice that the local center of a joint is not set to the global orientation. When using the default joint creation settings, the X-axis always points down the bone to the next joint (see Figure 3.7), enabling you to rotate easily around a joint's local center to twist a bone. You will want to do this in several parts of your character (to make a forearm twist, for example). Also notice that the Z-axis points toward you in the view in which you created the skeleton, because the Z-axis is the preferred rotation axis if IK is attached to the skeleton.

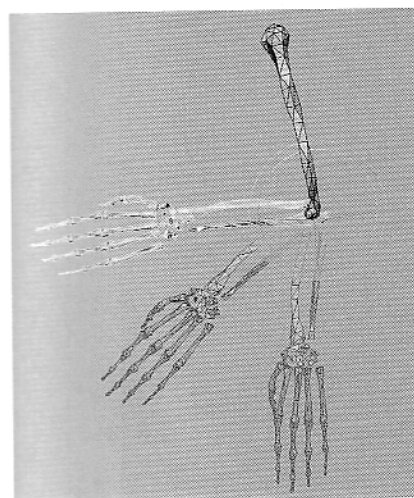


3.7 The default orientation of joints has the local X-axis pointing toward the next joint in the skeleton chain.





3.8 Draw the IK leg skeletons with a slight bend toward the front of the knee to set the preferred angle.




3.9 A swinging motion on the limb joints is not possible when IK is constraining a skeleton.

You can set the skeleton joint creation options to create IK automatically when you draw a skeleton, or you can add the IK manually after you have drawn the skeleton. In either case, draw your skeletons in a particular way when you know they will be constrained with an IK solver. IK bends in only one direction, which is based on the preferred angle of the joints. The preferred angle is the direction the joints are pointing when they are initially drawn. Draw your leg skeletons in the side view with a slight bend toward the front of the knees, for instance, to ensure that they have the correct preferred angle when their IK is activated (see Figure 3.8). Usually this requires you to draw the skeletons in a particular orthographic view, which is perpendicular to the axis that the joint should rotate in. The main axis of rotation on a normal IK skeleton is always the Z-axis.

There are some obvious advantages and disadvantages to using IK or FK on your skeletons. One advantage of IK is that it is faster to set and edit translation keys on a single IK handle, than to set and edit rotation keys on multiple joints. It also is easier to target the end of a limb in 3D space when you are animating (to make the feet target the floor, for instance). On the other hand, IK is constrained to bend in only one direction, whereas FK can bend in any direction. This makes IK more suitable for hinge joints, such as the elbows and knees. FK, on the other hand, is more suitable for joints that move more like ball joints, such as the backbone vertebrae.

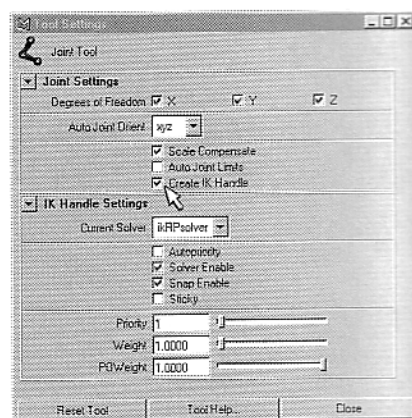
Another limitation of IK is that all the joints in a solver move when the IK handle is animated, making it impossible to isolate the rotation of a single joint in the chain. You must be able to rotate a child joint without rotating the parent if you want to create a swinging-type motion on the arms or legs (see Figure 3.9). This motion type usually occurs only as an unconscious movement while walking, throwing, or kicking. Because many limb motions are conscious, however, it is still better to use IK on the arms and legs most of the time. For the times when you need to create a swinging motion, however, you must have controls for switching between IK and FK in the middle of your animation.

All the tools for drawing skeletons and creating IK are under the Skeleton menu in the Animation module. Before you create a skeleton, check the settings in the Joint Tool options box by choosing Skeleton, Joint Tool . Here you can constrain a skeleton to rotate in a specific way, by turning off the Degrees of Freedom for a particular axis. You also can change the way Maya orients the local centers on joints by

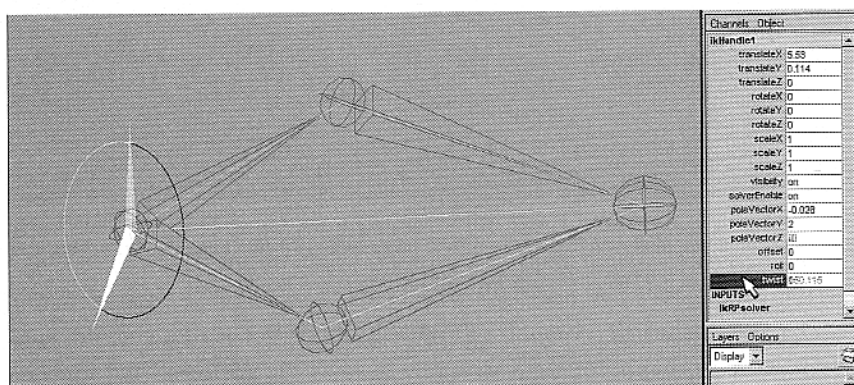
setting the Auto Joint Orient to something other than XYZ. For most skeletons, however, it is best to use the default settings. The only setting you will frequently change is the Create IK Handle option (see Figure 3.10).

You can add IK to your skeleton automatically when you draw it, or you can add it later after you draw the skeleton by choosing IK Handle Tool in the Skeletons menu. The available options are the same in either case. The main difference is that IK, if added automatically, always constrains the entire skeleton with the solver; if added manually, however, IK enables you to specify what joints will be constrained. You also can add more than one IK handle to different parts of the same skeleton if you add the IK manually. Like the joint options, you usually use the default IK handle option settings. Keep in mind that you also can adjust most of the joint and IK handle options in the Attribute Editor after you create a joint or IK handle.

One IK handle option you will occasionally change is whether the current solver is a Single Chain (SC) or a Rotate Plane (RP) solver. The difference between these two solvers is how they control the overall twist orientation of the skeleton. The SC solver forces the skeleton to twist when the IK handle is rotated. The RP solver, on the other hand, has a separate twist channel for twisting the skeleton, and the IK handle affects the skeleton only through translation (see Figure 3.11). You get more flexibility by separating the Twist attribute from the Rotation attributes of the IK handle, and the separation enables you to control the twist channel with a separate object by using a pole vector constraint. Because of this, you will be using an RP solver most of the time. The arms and leg skeletons of your character, for instance, will use RP solvers so that you can control where the elbows and knees point by using pole vector constraints.



**3.10** Open the Joint Tool options box to turn on or off the automatic creation of an IK handle on a skeleton.

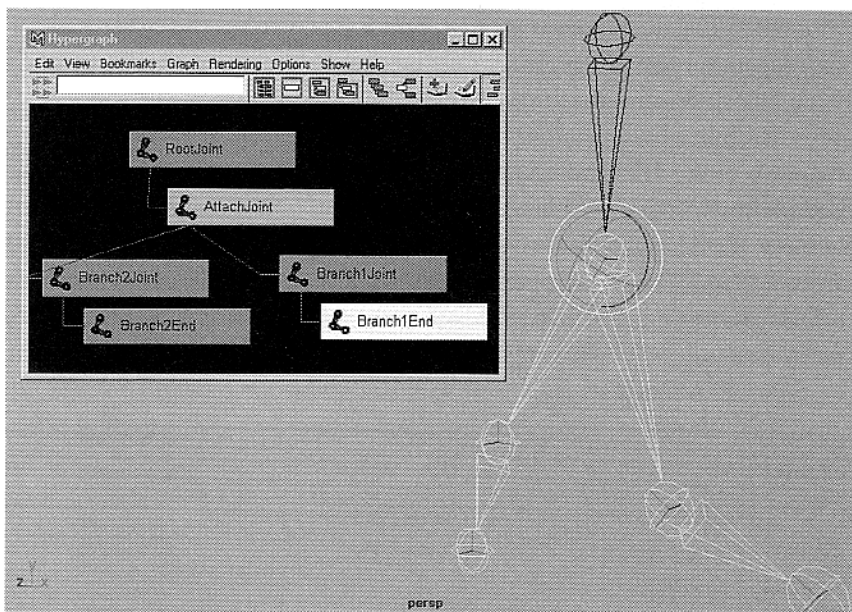


**3.11** When you create an IK handle with an RP solver, a separate twist channel controls the overall orientation of the skeleton.



When drawing skeletons, it is best to click and drag with the left mouse button held down. This action enables you to place joints precisely while drawing them. Correct placement is important, because modifying skeletons after they have been drawn creates values in the joint's rotation channels, which can be undesirable. If you place a joint in the wrong place while drawing the skeleton, you can press the Z key to undo, and proceed to redraw the joint. When all the joints are drawn, press the Enter key to set the skeleton.

One thing to consider when drawing skeletons is whether you want to attach multiple branches to a single joint. Do this by first clicking a joint within an already existing skeleton when drawing a new skeleton. When you finish drawing the new branch, notice that rotating the joint you clicked rotates both branches together (see Figure 3.12). This joint rotation occurs because the two joints have merged into one joint. Although you can create an entire character skeleton as one piece this way, this method provides limited flexibility for animation because it prevents you from being able to animate branches separately from each other.



**3.12** Attaching two skeletons creates a single parent joint for two separate branches. You cannot rotate the two branches separately from one another.

Instead of attaching skeleton branches, draw the joints separately, and parent the branches to a single joint or control object. Doing this enables you to animate the branches together by animating the parent object, or separately by animating the child joints, giving you more flexibility when animating. To draw a skeleton branch so that it starts on a joint but is not attached to the joint, avoid directly clicking the already existing joint. Instead, after clicking, drag the new joint on top of the previously created joint, and continue drawing the branch. You can then parent the joints under a control object or group node.

When parenting joints, notice that a bone is always drawn between the parent joint and the root joint of the branch. Keep in mind this can sometimes clutter your interface with crisscrossing joints on a complex skeleton. To keep this from happening, you have to put two group nodes between the joints. Do this by parenting the two joints, and then select the child root joint and press **Ctrl+G** twice. After doing this, notice that the connecting bone disappears. Also be aware that this hasn't changed the functionality of the skeletons.

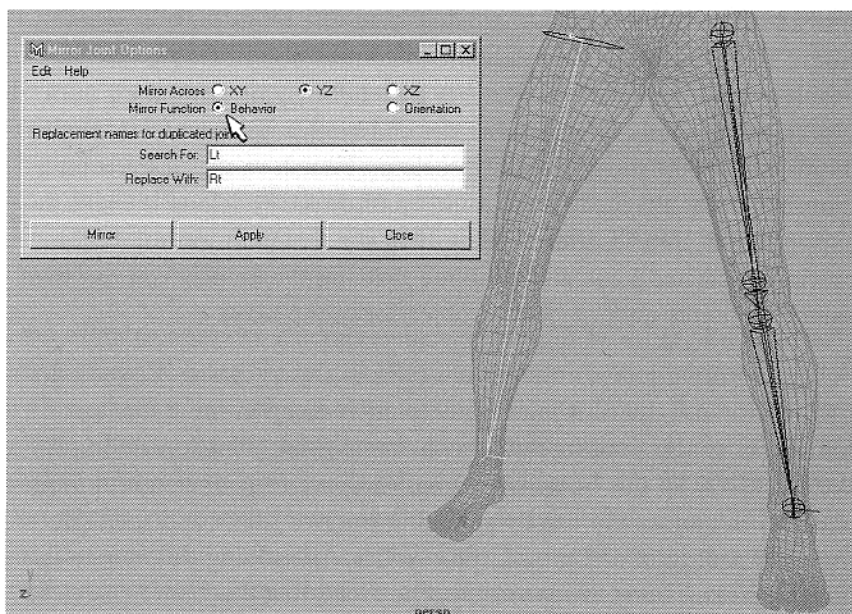
## EDITING SKELETONS

The Skeleton menu contains all the tools and commands for editing skeletons. Use the Insert Joint tool to add joints to a skeleton chain. Just click any parent joint, and drag to place the new joint. Delete joints in a skeleton chain by selecting any joint but the root joint, and choose Remove Joint. Some other useful commands are Disconnect Joint, Connect Joint, and Mirror Joint. Using the Mirror Joint tool speeds up your workflow by enabling you to more easily duplicate skeletons to the opposing side of your character's body (see Figure 3.13). This command is based on your character's position in global space, so make sure your character is centered on the global axis in a symmetrical manner. A new option in Maya 5 is the ability to replace naming conventions on the mirrored joints. For instance, you can specify that all joints that begin with Lt to begin with Rt.

Ideally, you want to draw your skeletons with the correct preferred angle. Sometimes, however, that won't be possible, and you will have to reset the preferred angle on a skeleton. Do this by removing any IK that may be on the skeleton, rotate the joints to their new preferred angle, and with the joints still selected, choose Skeleton, Set Preferred Angle. You can also set this by right-clicking the skeleton. You then have to reassign the IK to the skeleton. If you ever have any question what the preferred angle of a joint is, select the joint and choose Assume Preferred Angle. Finally, in the Skeleton menu you can disable or enable all IK solvers in your scene, or specific IK handles as needed.




- 3.13 Mirroring a skeleton in a particular plane is one of the most useful commands for editing skeletons.

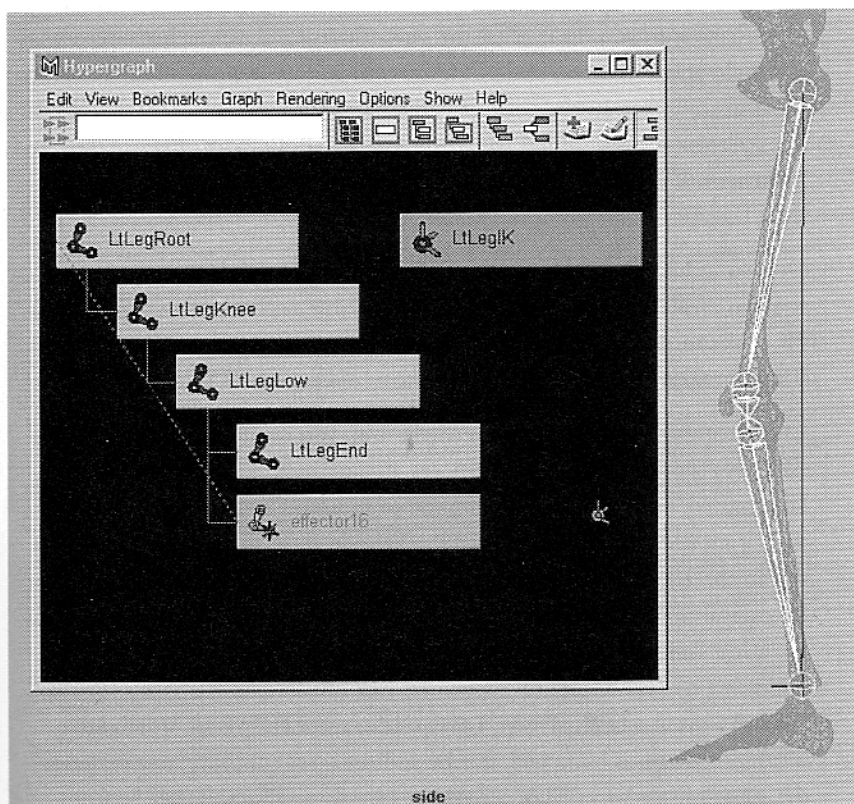


## EXERCISE 3.2

### DRAWING AND EDITING YOUR SKELETONS


1. Using the polygon reference bones as a guide, you now draw the basic Maya skeletons for your character. Before you start to draw the joints, however, place all the polygon reference bones on a layer, and set the layer to Template. Because the skin models are not needed at this time, hide them by turning off their layer's visibility. In addition, if you need to adjust the display size of the joints, because your model is too large or too small, do so in the Display menu under Joint Size.
2. Begin by drawing an IK leg skeleton in the side view. Do this by choosing Skeleton, Joint Tool , and turn on the Create IK Handle option. Make sure the Current Solver is set to IKRP Solver, and close the options box. Starting at the hips and ending at the ankle, click four times to create the joints. In the knee area, create a small joint that will mimic the flat of the knee. This will make the knee bend nicely when the character is bound to the skin. Make sure the skeleton is bent slightly toward the front of the knee so that you get the correct preferred angle, and then click Enter.

- Open a floating hypergraph view by choosing Window, Hypergraph on the top menu bar. You will use the Hypergraph view to name and organize your skeletons. Size the Hypergraph window so that you can still see your character in the perspective view. Then place your cursor in the Hypergraph window, and click the A key to Frame All. Notice that you have a four-joint skeleton displayed as a hierarchy of graphical nodes, with a hidden effector node, and a separate IK handle node. The IK handle is forced to stick to the effector, which is usually placed on the last joint of an IK skeleton. This changes as soon as you parent the IK handle under an object. Try selecting the IK handle in the hypergraph view, and translate it in the perspective view. You should see your leg skeleton bend. Press Z to undo the movement, and then name each of the leg joints **LtLegRoot**, **LtLegKnee**, **LtLegLow**, and **LtLegEnd**. Name the IK handle **LtLegIK** (see Figure 3.14). It is not necessary to name the effector.



3.14 Name all your skeleton leg joints and leg IK handle.

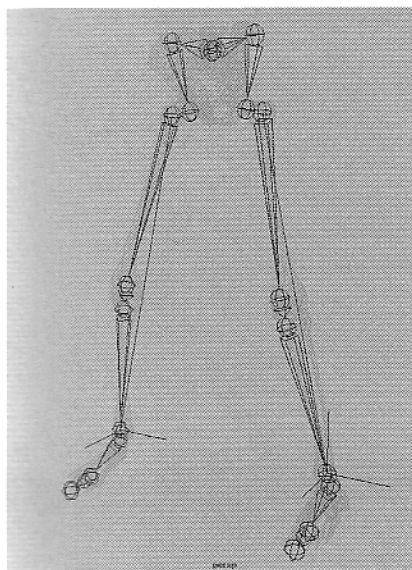


Rotate and translate the **LtLegRoot** joint in the front and side views so that it sits right on top of your polygon leg bones. Make sure you do not translate any joints except the root joint. Also make sure the **LtLegKnee** and **LtLegLow** joints rotate correctly around the left knee's pivot point. When you get the left leg placed correctly, select the **LtLegRoot** joint, and choose **Skeleton, Mirror Joint** . Set the **Mirror Across** option to the **YZ** plane, and the **Mirror Function** to **Behavior**. This creates a right-leg skeleton and associated IK handle that is a mirror copy of the left-leg skeleton. Use the new joint renaming option to name the right-leg joints the right-leg joints and IK handle **RtLegRoot**, **RtLegKnee**, **RtLegLow**, **RtLegEnd**, and **RtLegIK**.

4. Draw a three-joint skeleton for the left foot in the side view. This time, however, turn off the **Create IK Handle** option in the **Joint Tool** options box. Start the skeleton at the ankle pivot, going down to the ball of the foot, and then out to the end of the toe. When you are drawing the foot root, make sure you don't click the end of the leg skeleton, because doing so would attach the two skeletons. You will be parenting the foot skeletons to control objects, so they don't need to be attached. After drawing, adjust the skeleton as needed, and name the joints **LtFootRoot**, **LtFootBall**, and **LtFootEnd**. When you are done with the left foot, mirror it to create the right foot with the appropriate name changes.

One consideration to note is that if your character has bare feet, you may want to create individual toe skeletons. Do this easily by drawing them as **FK** skeletons in the side view. Later, when parenting your skeletons into a rig, just make the individual toe skeletons child to the **LtFootBall** joint.

5. Even though the hips move as a single unit, you create two separate hip skeletons for skin-weighting purposes. Again, these skeletons should not be attached, but will later be parented to give you more flexibility when animating them. Begin by drawing a skeleton from the center of the hips out to the edge of the pelvis, and down to the top of the left-leg skeleton. Be careful not to place the pivot for your character's hip skeletons too low. Each hip skeleton should have the pivot point for the root joint in the middle of the pelvis slightly below the belt line. Name the joints for the left-hip skeleton **LtHipRoot**, **LtHipSide**, and **LtHipEnd**. When your left-hip skeleton is complete, mirror and rename it appropriately to create the right-hip skeleton. This completes your lower-body skeletons (see Figure 3.15).

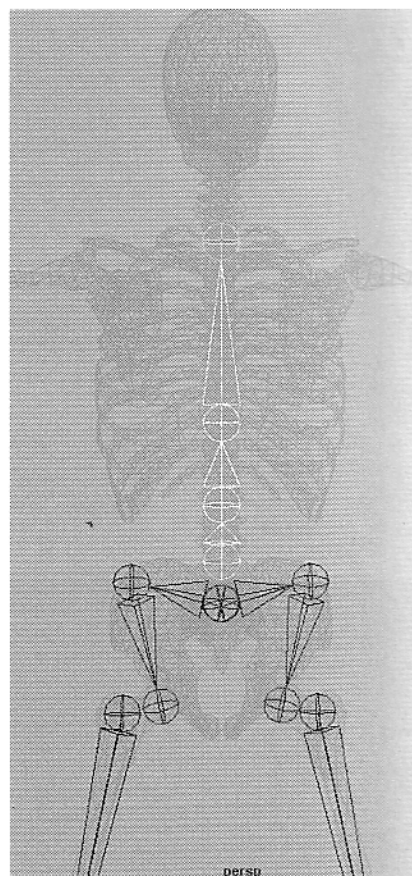


3.15 Using your polygon bones as reference, finish drawing the lower-body skeletons for the hips, legs, and feet.

6. The part of the body that connects the lower- and upper-body skeletons is the backbone, which begins at the center of the hip and ends at the base of the neck. The backbone should be a separate skeleton from the hips, because you want to be able to animate the spine without moving the hips, and vice versa. To ensure this is the case, make sure you do not directly click the hip root joints when you begin drawing the backbone. Instead, in the front view, click away from the character and drag the backbone root over the hip roots. Then hold down the Shift key as you draw three joints straight upward, without any preferred angle (see Figure 3.16). The joints should go from the hip's pivot to around the belly button, then to the solar plexus, and finally to the top of the sternum. Be sure to also check the placement of the skeleton in the side view to make sure it is inside your character's torso. If necessary, translate the root joint in the Z-axis to position the skeleton correctly.

This is obviously a simpler version of the backbone than what exists in a real body, as it has fewer joints and doesn't follow the contour of the spine. The reason you are simplifying this skeleton is because it is going to be used as a control for a more complex backbone skeleton you create later in this chapter. You draw the skeleton straight, because it is going to be using only FK, and this gives you a smooth arc when the backbone bends in any direction. Keep in mind, however, that a simplified backbone such as this is acceptable as-is for a simple character. Finish the backbone skeleton by naming the joints **BackRoot**, **Back2**, **Back3**, and **BackEnd**.

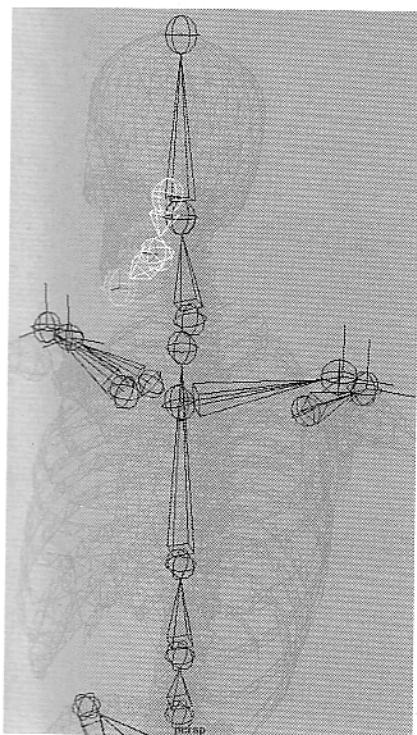
7. To begin the upper-body skeletons, create two IK skeletons for the clavicles. Make sure you turn on Create IK Handle in the Joint Tool options box, and then draw the left-clavicle skeleton in the front view, using the polygon bones as a guide. It should originate at the middle of the chest and end at the top of the shoulder. After you have created the left skeleton, select the root joint, and transform it as needed to position it correctly over the polygon clavicle. Name the joints **LtClavicleRoot**, **LtClavicleEnd**, and **LtClavicleIK**. When completed, mirror the skeleton to create the right clavicle and rename the joints as needed.
8. Create a left IK scapula joint in the same way as you created the clavicle joint. Draw it in the front view so that it goes from the inner edge of the scapula to the top of the shoulder, where the clavicle and scapula meet. The finished scapula skeleton should follow the raised area at the top of the



3.16 Draw a simplified FK backbone skeleton straight upward, without any preferred angle.

polygon scapula bone. Transform it in all the views to make it fit properly, and name the joints **LtScapulaRoot**, **LtScapulaEnd**, and **LtScapulaIK**. Mirror the result to the other side, and rename the joints appropriately.

9. The neck skeleton will be simplified to a two-joint FK skeleton. The reason for this is that the neck really doesn't bend much in a real body, but instead serves mostly as a pivot for shifting the head around. On a simple character, this kind of neck works fine. Later in this chapter, you refine the neck to be more complex. In the side view, draw the neck joint from the end of the backbone skeleton to the base of the skull. Because this should be an FK skeleton, make sure to turn off the Joint tool's Create IK Handle option before drawing. Also be careful to not attach the neck skeleton to the back skeleton. Name the resulting neck joints **NeckRoot** and **NeckEnd**.
10. To create the head and jaw skeletons, you also use FK, so leave the Joint tool's IK option turned off. In the side view, begin the head skeleton at the base of the skull, and end it at the top of the skull. Be sure not to attach it to the end of the neck joint. The pivot point for the head root joint should be slightly below the ear, from where your character tilts its head. Name the two head joints **HeadRoot** and **HeadEnd**. You also should draw the jaw in the side view, from close to the same pivot point as the head root. Make the jaw a three-joint chain that goes down and out to the end of the chin (see Figure 3.17). Contouring the shape of the jaw creates better weighting when the skin is bound. Name the jaw joints **JawRoot**, **JawLow**, and **JawEnd**.



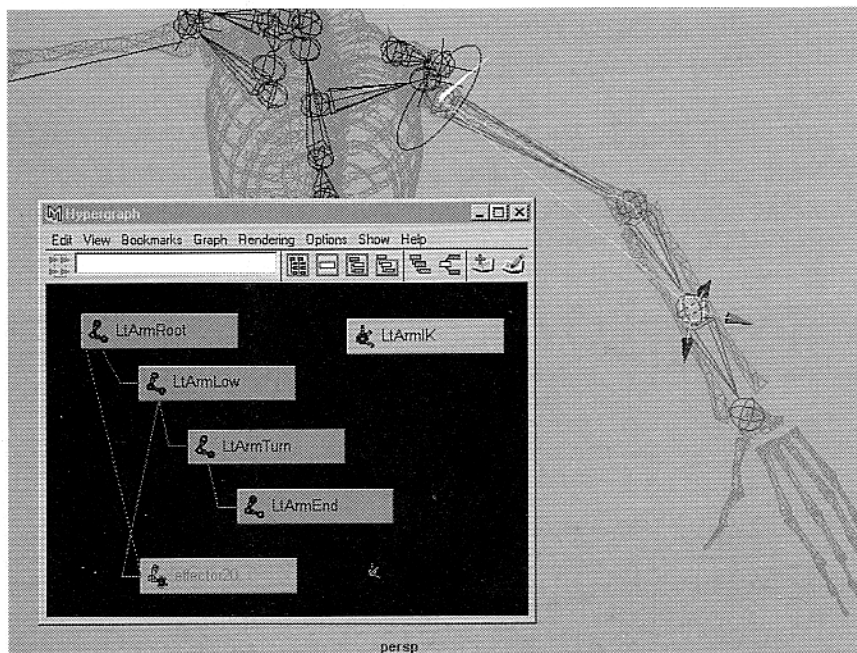
3.17 Add an FK jaw skeleton after drawing skeletons for the clavicles, scapulas, and skull.

11. The last joints you are going to draw for your basic skeleton controls are for the arm and hand. Draw the arm with a slight bend toward the elbow. This means that you will draw the arm skeleton in the top view if your character's elbow faces backward and the hand faces downward. Or draw it in the front view if your character's elbow faces downward and the hand faces forward. Make sure IK is turned off in the Joint Tool options box, and begin by clicking close to the left shoulder socket. Draw the arm skeleton from the shoulder to the elbow, from the elbow to midway down the forearm, and finally to the wrist. Name the joints **LtArmRoot**, **LtArmLow**, **LtArmTurn**, and **LtArmEnd**.

The reason you turn off IK when drawing the arm skeleton is that the IK for the arm should not be placed on the entire skeleton. As mentioned previously, IK always constrains all the joints within the solver to rotate in one direction. For the elbow joint, which works as a hinge joint, this is fine. But



in addition to the elbow joint, you have drawn a **LtArmTurn** joint that will be used to twist the lower arm of your character. To make sure the arm IK doesn't interfere with the rotating of this joint in X, leave this joint out of the solver. Do this by selecting the IK Handle tool, and click the arm root joint, and then click the **LtArmTurn** joint (see Figure 3.18). Name the resulting IK handle **LtArmIK**.



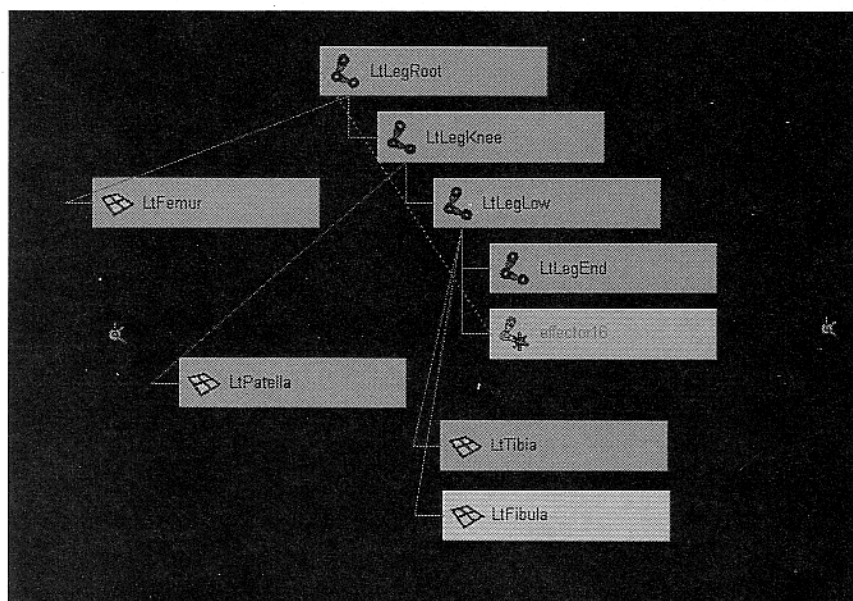
- 3.18** Place the arm IK from the shoulder to the middle of the forearm so that the arm turn joint is outside the RP solver. Then move the pivot of the arm effector to the wrist.

When the IK is added, notice that you can still bend the arm using the IK handle, while still being able to select the **LtArmTurn** joint to rotate it freely in X. To make the IK act more like it controls the whole arm, you can move the effector to sit over the wrist joint. Do this in the hypergraph view by selecting the arm joint's hidden effector node, and then click the Insert key to move the effector's pivot. Notice that the IK handle sticks to the effector when it is moved. Hold down the V key to snap the effector on top of the **LtArmEnd** joint. When finished, click the Insert key again to go out of pivot point mode. Now when you translate the IK handle, it behaves as if it controls the arm from the wrist, rather than from the forearm.

Finally, draw a two-joint FK arm skeleton for the left hand. You can draw this in the front view, from where the wrist rotates to where the finger bones begin. Do not attach the hand skeleton to the end of the arm. Name the hand joints **LtHandRoot** and **LtHandEnd**. When finished, mirror the left-arm and left-hand skeletons to the right side, and rename them appropriately.

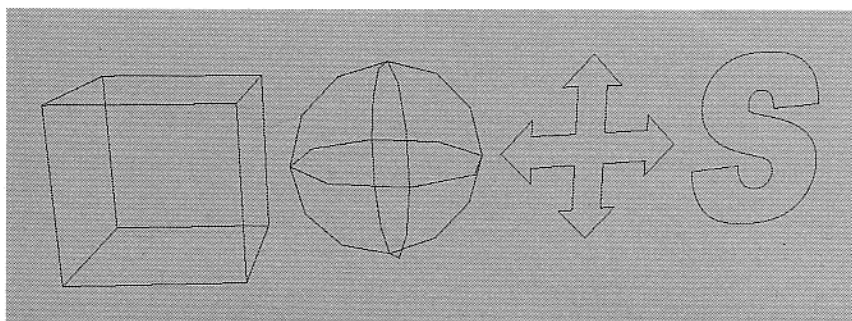
12. After you have drawn the last skeleton, go back and parent all the polygon reference bones under the appropriate skeleton joints. For instance, make the left femur child of the LtLegRoot joint. The left patella should be child of LtLegKnee, and both the left tibia and fibula should be child of LtLegLow (see Figure 3.19). Make the many back vertebrae children of the closest backbone joint, and make the rib cage child to the Back3 joint. Avoid parenting any of the polygon bones under IK handles or control icons, except the pelvis bone, which should be made temporarily child of the Hips box. ■

- 3.19 After you have finished drawing all your skeletons, parent the polygon bones to the appropriate Maya joints, as seen here with the leg skeleton.



## CREATING CONTROL ICONS

You may have noticed while you were naming your joints and IK handles that they are not that easy to select in the interface. Some joints are sitting on top of each other, and IK handles look just like locators. Traditionally, character setup artists have dealt with this problem by creating control icons. A control icon can be any easily selectable 3D shape, such as spheres, boxes, arrows, dials, and so forth, that will be parented to your skeletons and IK handles (see Figure 3.20). You create these shapes




**3.20** Control icons are made from curves that can be in a variety of 3D shapes

from curves so that they will not be visible when the surfaces are rendered. When the control icons are created, you place their centers directly over the pivot points of the joints they are meant to control. Once the icons are created and placed, they must then be parented over the appropriate joints and IK handles. Some control icons may control only a single skeleton or IK handle, whereas others may control many skeletons. By using control icons, you make your controls more efficient and useful.

### EXERCISE 3.3

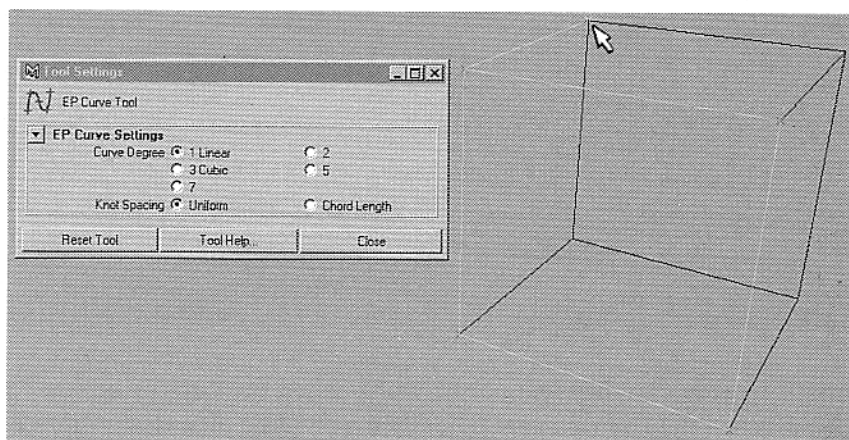
#### CREATING AND PLACING CONTROL ICONS

1. Character setup artists often use a variety of shapes when creating control icons out of curves. It doesn't really matter which shapes you use, as long as the shapes are easy to select in all the views. Before you begin creating some control icons, select all your skeletons and IK handles, and place them on a layer named Skeletons. Make sure your Models, Polygon Reference Bones, and Skeletons layers are all set to Template.

One shape that is easy to create with a curve is a box. Begin by creating a polygon cube by choosing Create, Polygon Primitives, Cube on the top menu bar. Scale the cube so that it is not too small in relation to your character. Make the perspective view full screen, and use the Alt key to orbit around so you can clearly see all the corners of the cube. Then choose Create, EP Curve tool  on the top menu bar. Inside the EP Curve Tool options box, set the Curve Degree to Linear, and close the options box. Hold down the V key as you click the corners of the cube (see Figure 3.21). It is important to make your control box from a single curve, so you need to make the curve overlap itself to cover all the edges of the cube. Keep clicking until you think the box is complete. When finished, select the polygon cube in the hypergraph view, and delete it. You should now have a 3D box made from a curve!



- 3.21 Snap the points of a linear EP curve to the edges of a polygon cube to create a box icon.

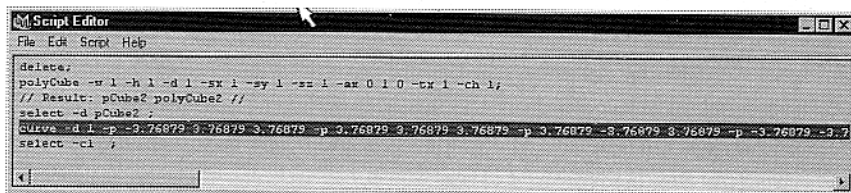


One thing you might want to do for creating more boxes in the future is create a shelf button for the creation of a box. Open the Script Editor, and in the gray History field, find the line of *Maya Embedded Language* (MEL) code that was used when you created your box icon. It will look something like this:

```
curve -d 1 -p -3.511352 3.511352 3.511352 -p 3.511352
3.511352 3.511352 -p 3.511352 -3.511352 3.511352 -p -
3.511352 -3.511352 3.511352 -p -3.511352 3.511352
3.511352 -p -3.511352 3.511352 -3.511352 -p -3.511352 -
3.511352 -3.511352 -p -3.511352 -3.511352 3.511352 -p -
3.511352 3.511352 3.511352 -p 3.511352 3.511352 3.511352
-p 3.511352 3.511352 -3.511352 -p -3.511352 3.511352 -
3.511352 -p -3.511352 -3.511352 -3.511352 -p 3.511352 -
3.511352 -3.511352 -p 3.511352 3.511352 -3.511352 -p
3.511352 3.511352 3.511352 -p 3.511352 -3.511352 3.511352
-p 3.511352 -3.511352 -3.511352 -k 0 -k 1 -k 2 -k 3 -k 4
-k 5 -k 6 -k 7 -k 8 -k 9 -k 10 -k 11 -k 12 -k 13 -k 14 -
k 15 -k 16
-k 17 ;
```

Highlight this line of code in the Script Editor, and using the middle mouse button drag it to your shelf (see Figure 3.22). This action produces a shelf button that creates a new curve box every time you click it. Open the Shelf Editor, and name the new button **Box**.

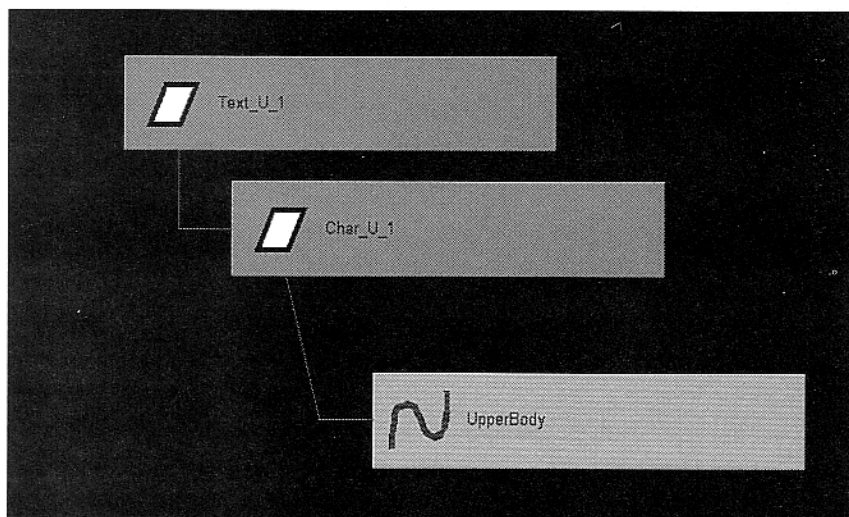
- 3.22 One way to make a shelf button is to highlight MEL code in the History field of the Script Editor, and then use the middle mouse button to drag it to the shelf.



2. Now you should create and position all the control boxes for controlling your character's limbs. Place your new control box directly on top of the LtHandRoot joint, and scale it so it is easily selectable. Then either duplicate the left-wrist box or click your Box shelf button to create more control boxes for the right wrist and both ankles. Place the leg boxes so their centers are directly on top of each foot root joint. When the boxes are placed appropriately, reset their transform channels by choosing Modify, Freeze Transformations, and name them **LtArm**, **RtArm**, **LtLeg**, and **RtLeg**.
3. Create two new control boxes, and name them **Hips** and **Head**. If you want to change the shape of any control boxes to make them look different from the others, just go into component mode, and transform the vertices. Scale the Hips box so that it is slightly larger than, and contours, your character's hips. In insert mode, place the pivot point for the Hips box where your character's hip root joints are located. Place the Head box so it fits around your character's head, and move the pivot point so it sits on top of your head root joint.
4. Another kind of control icon you can use is a text curve. These icons are easy to create, and are also easy for the animator to recognize. Create a text curve by choosing Create, Text ☐ on the top menu bar, and make sure the type is set to Curves. Avoid using multiple letters, or letters that require multiple curves, such as *e*, *a*, *p*, or *d*. If you do use these letters, delete the inner curve that creates the hole. You can use letters that have no holes, such as *u*, *z*, *v*, *t*, *y*, *l*, and *s*, without modifications.
5. Create a letter *U*, and move it so that it sits right on top of the Hips box. In the hypergraph view, notice that the *U* curve has a couple of parent group nodes. Select the curve node, disconnect it from its parent, and delete the group nodes (see Figure 3.23). Name the *U* curve **UpperBody**. Finally, in component mode, select all the vertices of the curve, and transform them to the left and in front of your character. In addition, to make the curve easier to see in the side view, rotate the vertices in Y about 35 degrees. The reason you make these adjustments by moving points in component mode, instead of adjusting the transforms of the UpperBody curve, is that you want the center of the curve to remain in the middle of your character's torso. This will ensure that your character's torso can be rotated around the correct pivot point. When the UpperBody icon is in the correct place, freeze its transforms.

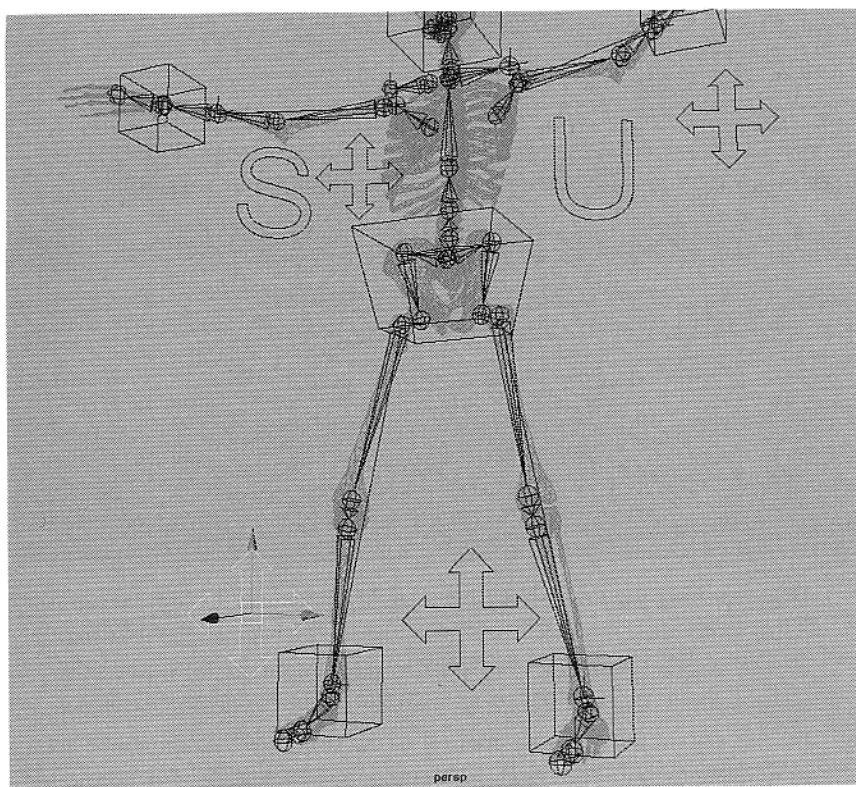
Create an *S* text curve, and name it **Shoulders**. Move the Shoulders curve so that it sits in the middle of your character's shoulders. Then, in component mode, select all its vertices, and transform them to the right and in front of your character. Also rotate them in Y counterclockwise about 35 degrees. Freeze the Shoulders icon when you are finished.

- 3.23 Separate and delete the two group node parents of the U text curve that will be used as the upper-body icon.



6. Another kind of icon frequently used by character setup artists is an arrow. You can make multidirectional arrows for your elbow and knee controls. Do this in the top view by turning on Snap to Grid, and draw an EP curve on the grid in the shape of an arrow. When drawn, name the arrow **LtElbow**, and transform it behind the left elbow. Make sure that the left-elbow icon is sitting several grid units away from the arm, and not right on top of the elbow. When positioned, duplicate the icon, and create three more arrows named **RtElbow**, **LtKnee**, and **RtKnee**. Move the icons into place, making sure the knee icons are sitting several grid units in front of each knee (see Figure 3.24). When positioned correctly, freeze all the icons.

For the elbow and knee controls to work, you must assign them to the arm and leg skeletons with a pole vector constraint. This constraint controls the overall orientation of each skeleton by forcing the elbows and knees to point at the appropriate arrow icon. This is possible because the arm and leg IK skeletons use an RP solver, which has a twist channel that controls the rotate plane of the skeleton. This twist channel will be constrained using the pole vector constraint. Do this by selecting the left-elbow arrow, and then Shift-select the left-arm IK handle, and choose **Constrain, Pole Vector** on the top menu bar. A line should display connecting the IK to the left-elbow arrow in the 3D views. Move the arrow up and down to see it rotate the arm skeleton. Then create pole vector constraints on the IK for the right arm and knees. ■



**3.24** In front of your character's legs, place arrow icons that control where the knees point through the use of pole vector constraints.

## CREATING A BASIC CHARACTER RIG

Up until now, you have been creating all the elements you will use to control your character while animating. They are currently not very useful, however, because they are not connected in any way to each other. To use all your controls effectively, you must make them into a character rig. A rig is created when you parent all your skeletons, IK handles, group nodes, and control icons into one big hierarchy. This hierarchy organizes all your controls into a logical setup that is easy to use, and easy to import into multiple scene files.

Keep in mind while you are going through the rigging process that if objects need to move independently of one another, they must be on separate branches in a hierarchy. They cannot be child of each other. The feet, for instance, should be able to stay on the ground when the upper body moves. The upper body should also be able to stay still when a foot is raised. To accomplish this, you must place these two parts of the body on completely separate branches in the hierarchy. Although variations in rig structures will always exist, keep in mind that most basic rigs are built under these same principles.



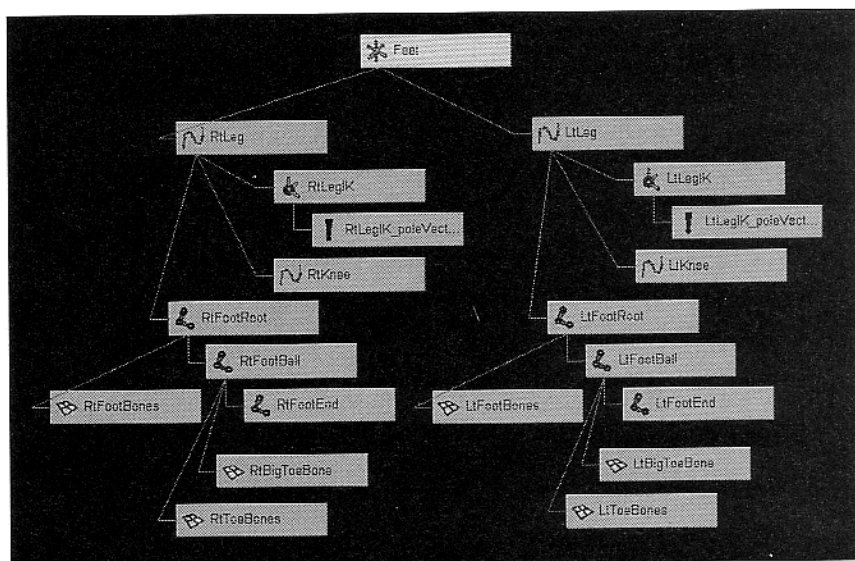
**EXERCISE 3.4****PARENTING SKELETONS AND CONTROL ICONS INTO A CHARACTER RIG**

1. It may be easiest to do all your parenting in the hypergraph view. If you haven't done so already, change your hypergraph to freeform layout by choosing Options, Layout, Freeform Layout on the Hypergraph window's top menu bar. Keep in mind that an easy way to parent nodes in the hypergraph is to drag the child on top of the parent with the middle mouse button. Use the middle mouse button to drag an already parented node over an empty space in the view to unparent it. Other things to be aware of when working in the hypergraph is that you want to keep everything organized neatly. If shape nodes are showing up in your window, turn them off by choosing Options, Display, Shape Nodes. You can also collapse hierarchies, expand hierarchies, and create bookmarks by right-clicking in the hypergraph.

Begin creating the lower-body hierarchy by parenting the leg and hip root joints under the Hips box. Then make the leg IK handles, the root joints for the feet, and the knee icons child to the appropriate leg boxes. For instance, the LtLeg box should be the parent of LtLegIK, LtLegRoot, and LtKnee. Parent the right-leg controls in the same way. You should be able to translate the Hips box, and the legs will bend, while the feet remain stationary. You should also be able to translate a leg box, and the foot will move with the leg, while the hips remain stationary.

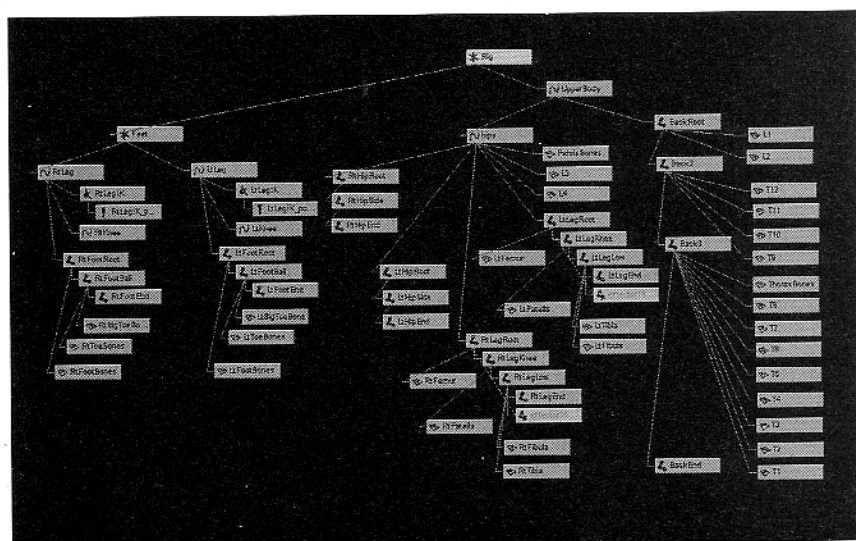
Then, create a locator by choosing Create, Locator. Name the locator **Feet**, and translate it so it sits directly between the two leg boxes. When in place, freeze the locator, and make it the parent of both leg boxes (see Figure 3.25). This locator is occasionally used to move the feet together, such as to make your character jump.

2. Begin creating the upper-body hierarchy by making the UpperBody icon the parent of both the Hips box and the backbone root. Moving the UpperBody icon should move the hips and backbone together. It is important to make the hips and backbone on separate branches, so they can be moved independently of each other. In this basic rig setup, the Hips box should not be translated, because it separates the hips from the backbone. Instead, you should only rotate the Hips box to move the hips, and use the UpperBody icon to move the torso.
3. Create another locator and name it **Rig**. Translate this locator so that it sits directly on the pivot point of the UpperBody icon. You can either try to hold the V key to snap it into place, or you can use a point constraint to move it, and then delete the constraint. Do this by selecting the UpperBody



3.25 The Feet locator is the parent of both leg boxes, which have the individual foot skeletons, IK handles, and knee icons child to them.

icon, and then Shift-select the Rig locator, and choose Constraint, Point on the top menu bar. The locator should move into place. Then, in the hypergraph view, just delete the constraint node connected to the locator. When the Rig locator is in place, freeze it, and make it the parent of the Feet locator and the UpperBody icon (see Figure 3.26). The Rig locator will be the top of your character hierarchy, and can be used to move your entire character around in your scene.



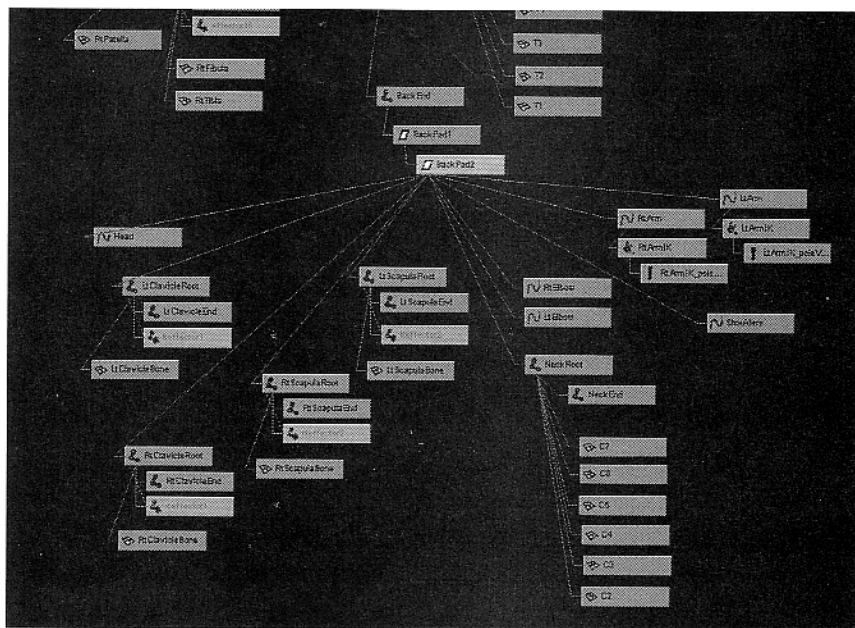
3.26 Create a locator named Rig as the top node of your hierarchy. Make the Rig locator the parent of the feet and upper body branches.

4. All the parts of your character's upper torso should move when the backbone bends. This includes the shoulders, neck, head, and arms. For this to work correctly, all these body parts will be made the indirect children of the

BackEnd joint. The only part of the body that is sometimes made to not follow the backbone is the hands. If you don't want your hands to follow the backbone's movement, make the arm boxes child to the UpperBody icon. Otherwise, they should be child to the end of the backbone, which is how you should do it for now. Later in this chapter, you learn how to switch this.

Begin creating the upper-body hierarchy by making the NeckRoot joint child of the BackEnd joint. Before continuing, create two group nodes that will be between the backbone and all the child nodes. Do this by selecting the NeckRoot joint, and press Ctrl+G twice. Name the two group nodes from top to bottom **BackPad1** and **BackPad2**. Because there will be several joints child to the end of the backbone, creating a couple of group nodes will keep unnecessary bones from being displayed. Then make the following nodes child to BackPad2: clavicle roots, scapula roots, neck root, Head box, Shoulders icon, arm boxes, and elbow icons (see Figure 3.27).

- 3.27 Make all the nodes that should move with the backbone child to the BackPad2 group node.

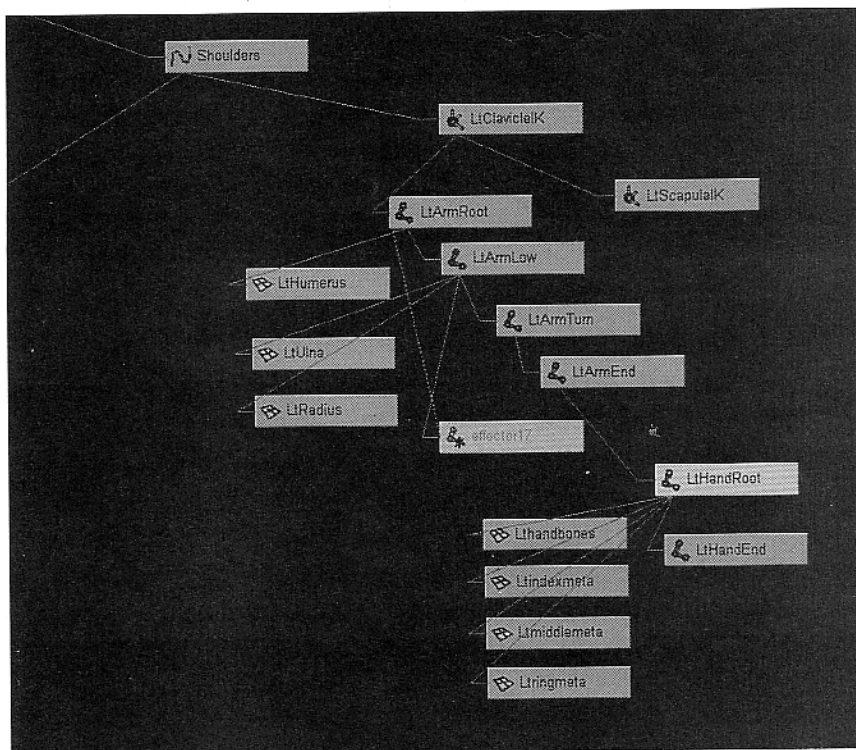


5. You need to parent several more nodes before you will be finished constructing your basic character rig. First, instead of parenting, you need to use a constraint to connect the Head box to the neck skeleton. This constraint causes the head to follow the neck movement, while keeping a vertical orientation. Select the NeckEnd joint, hold the Shift key down to also select the Head box, and constrain it by choosing Constrain, Point. Finally, make the Head box the parent of HeadRoot and JawRoot. When this is

done, rotating the NeckRoot joint should move the Head box, and rotating the Head box in turn should also rotate the head and jaw skeletons.

Make both clavicle IK handles children of the Shoulders icon. And make each scapula IK handle child of the clavicle IK that is on the same side of the body. For instance, LtClavicleIK should be the parent of LtScapulaIK. Once parented, translating the Shoulders icon up in Y should make both shoulders shrug, and make both scapula bones rotate outward slightly.

The clavicle IK handles are the main controls for each shoulder. The two things that should follow the clavicle IKs when they move are the scapula and the arm roots. To achieve this, make each arm root child of the appropriate clavicle IK. Then continue down the arm, making each arm IK child of the appropriate arm box, and each hand root child of the appropriate arm end joint (see Figure 3.28). The reason you make the arm end joint the parent of the hand rather than the arm box is to keep the hand oriented with the arm as the wrist box moves. This is usually what a real arm does most of the time, so it makes sense to make this the default hand orientation. Otherwise, if you were to make the hand child of the arm box, you would have to be constantly rotating it into place, which is not very efficient. In the next section, you create controls for moving the hand around as needed.

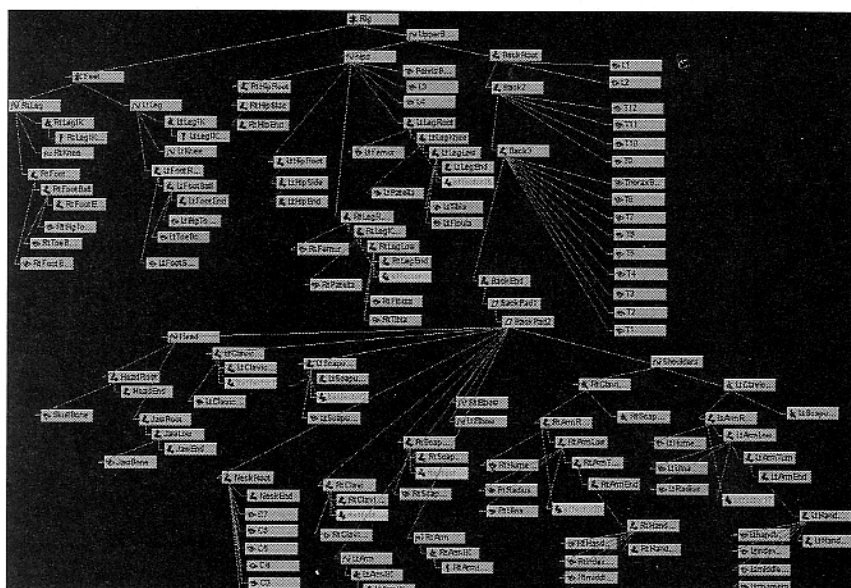


**3.28** Parenting the hand root joint under the end of the arm skeleton completes the basic arm hierarchy.



6. This completes the basic skeleton setup. Check your basic hierarchy in the hypergraph view to make sure everything is parented (see Figure 3.29). Try moving your controls around to make sure everything is parented correctly. For instance, moving the UpperBody icon up and down should make your character crouch. All your polygon reference bones should move with your skeleton joints. Be aware that there are currently no limits set on your controls, and translating a control too far may pull your rig apart. Make sure you undo after testing your controls to get your character back into its default position before going on to the next section. ■

- 3.29 Check to make sure all the nodes in your basic rig hierarchy are parented correctly.



## INCREASING RIG FUNCTIONALITY


You can animate the basic rig you created in the preceding section as it is, but it would lack efficiency. You would have to manually animate every control, and some skeletons and IK handles would still not have easily selectable control icons associated with them. The backbone joints, for instance, must be selected and animated individually. In this section, you refine the basic rig that you created in the preceding section, and add new controls that make it easier for you to animate your character. This process primarily involves connecting channels using constraints, mathematical expressions, and setting driven keys.


## USING CONSTRAINTS TO CREATE EYE CONTROLS

One of the few body parts of your character that won't be bound to skeletons is the eye geometry. Eyeballs usually don't deform much and need to be able to move around freely in their sockets. Even on a cartoon character that squashes and stretches, the eyeballs should be deformed with a lattice rather than skeletons. This is because skeletons lock the transforms of the geometry bound to them. So if you bind your eyeballs, you won't be able to move them around to make your character look in different directions. Instead, you should parent the eyeballs into your rig hierarchy, and use constraints to make controls for moving your character's eyes around.

### EXERCISE 3.5

#### CREATING EYE CONTROLS

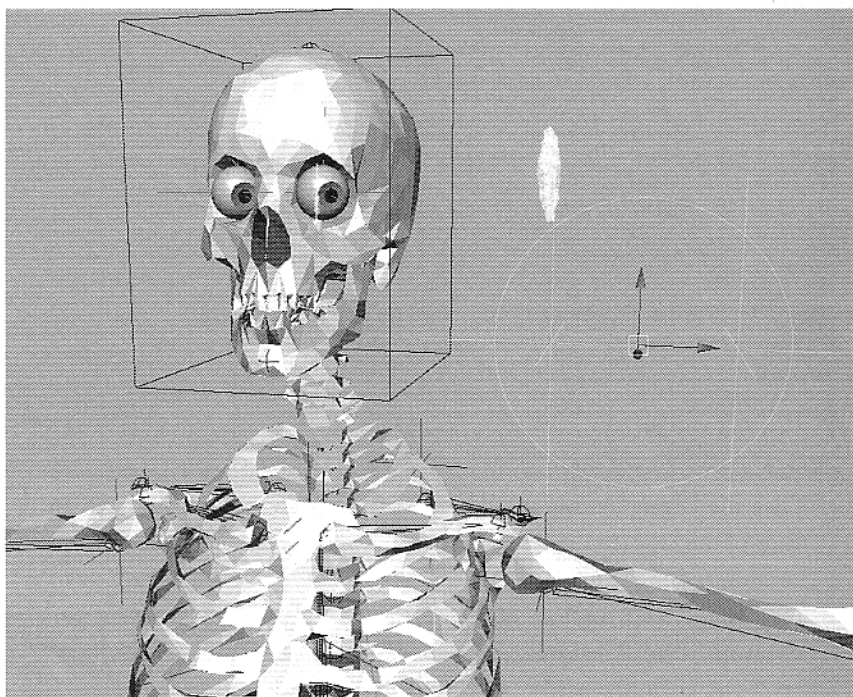
1. Create a new layer named **UnDeformed** for all models in your character that won't be deformed. Place your eyeball models on this layer, as well as any other models such as armor, jewelry, hats, and glasses. In addition, all such objects should be made parent to the joint they should follow—in the same way you parented the polygon reference bones to the appropriate joints. Make sure all the parts of each eyeball can be moved together by parenting them under the white part of the eyeball. Then, to make sure the eyeballs follow the head motion, make the white part of each eye child to the HeadRoot joint.
2. To make a control icon for the eyeballs, choose Create, NURBS Primitives, Circle  on the top menu bar. In the resulting options box, make sure the circle is facing forward by setting the Normal Axis to Z. Click the Create button, and translate the circle so that it sits directly in front of the character's eyes. Then, create two locators by choosing Create, Locator, and move them so each sits close to the edge of the circle, while also being directly in front of each eyeball. Freeze the circle and locators, and name them **EyesLook**, **LtEyeLook**, and **RtEyeLook**. Make the EyesLook circle the parent of both locators and the child of the Head box.

After you have made the eye controls, you need to constrain the eyeballs to them with an Aim constraint. Do this with the left eye by selecting the LtEyeLook locator, Shift-select the model for the white part of the left eye, and choose Constrain, Aim . In the Constraint options box, set the Aim Vector based on the center orientation of your eye model. Unless you made your eyeball model with the X-axis pointing forward, the default setting will not work correctly. If you made the eyeball parent so that its center is oriented according to the global axis, where Z is pointing forward, it

is the Z-axis that should be constrained. Change the Aim Vector fields so they read 0, 0, 1. This will constrain the Z-axis of the eye model to always point at the locator in front of it. Leave the rest of the fields in their default settings, and click the Add/Remove button. Then constrain the other eyeball to the locator in front of it.

When both eyeballs are constrained, test your controls by translating the circle around. Your character's eyes should track the circle icon (see Figure 3.30). In addition, you can scale and rotate the circle to get more cartoonish effects, such as crossing the eyes or making the eyes wobble. You can even animate the locators themselves to create a wandering-eye effect. In addition, some character setup artists make cone icons in front of their character's eyes to more easily see where the character is looking. Do this by just creating two cones from EP curves, and make each cone child to the white of each eye. ■

- 3.30 Create eye controls by aim-constraining each eyeball to a locator. Manipulating the parent circle icon makes the eyes move around.



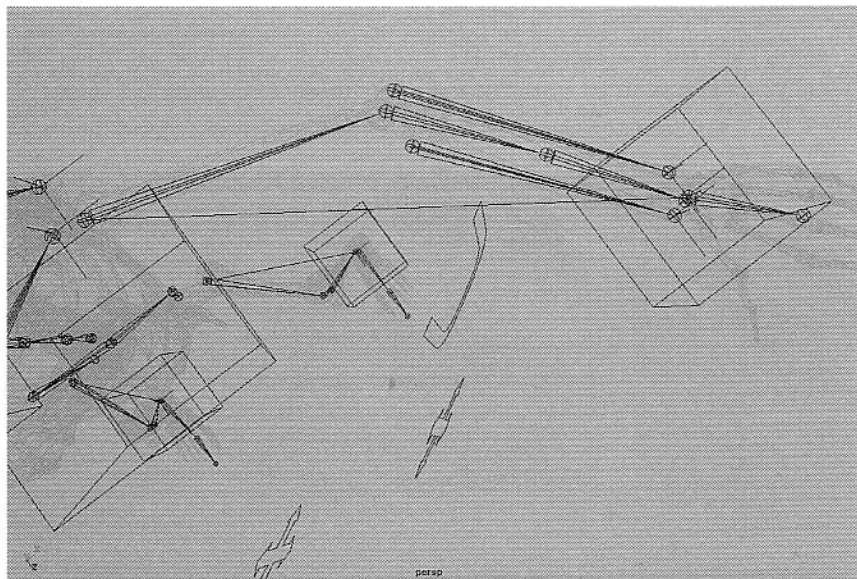
## REFINING THE LOWER ARMS AND HANDS

Although the basic controls are there, your rig still needs a little work to finish the arms and hands. In a real arm, the rotation of the radius and ulna bones makes the forearm twist. You add two IK joints in your lower arm hierarchy to get closer to how this works. In addition, you set an order to the wrist rotations by creating parented group nodes, and create some finger joints to finish your character's hand.

## EXERCISE 3.6

## ADDING FOREARM, HAND, AND FINGER CONTROLS

1. To create a radius and an ulna bone for your character's left arm, use your polygon reference bones as a guide, and draw two skeletons with IK turned on. The radius should go from slightly below where the inside of the arm bends and down to the thumb side of the wrist. The ulna should begin slightly below the elbow and end on the outside of the wrist. After drawing the skeleton, translate and rotate the root joints as needed to place them correctly. Make sure both skeletons are on either side of the main arm skeleton, because the **LtArmTurn** joint will be the axis they need to turn around (see Figure 3.31). Name the joints for the skeleton that is on the elbow side of the arm **LtUlnaRoot** and **LtUlnaEnd**. Name the joints for the skeleton on the other side of the arm **LtRadiusRoot** and **LtRadiusEnd**. Name the IK handles **LtUlnaIK** and **LtRadiusIK**.



**3.31** Create a radius and ulna skeleton on each side of the main arm skeleton.

2. In the hypergraph view, parent the radius and ulna root joints under the **LtArmLow** joint. This makes the two new skeletons follow the main arm skeleton whenever the left-arm box is moved. Not only do the radius and ulna joints need to move with the arm, they also need to move with the **LtArmTurn** joint when it rotates in X. To make this work the way it works in your own body, the radius should turn all the way up its axis, whereas the ulna should turn only at the wrist.

In a real body, the radius bone rotates all the way up its axis when the forearm twists. This is the reason the bicep's muscle, which is attached to the

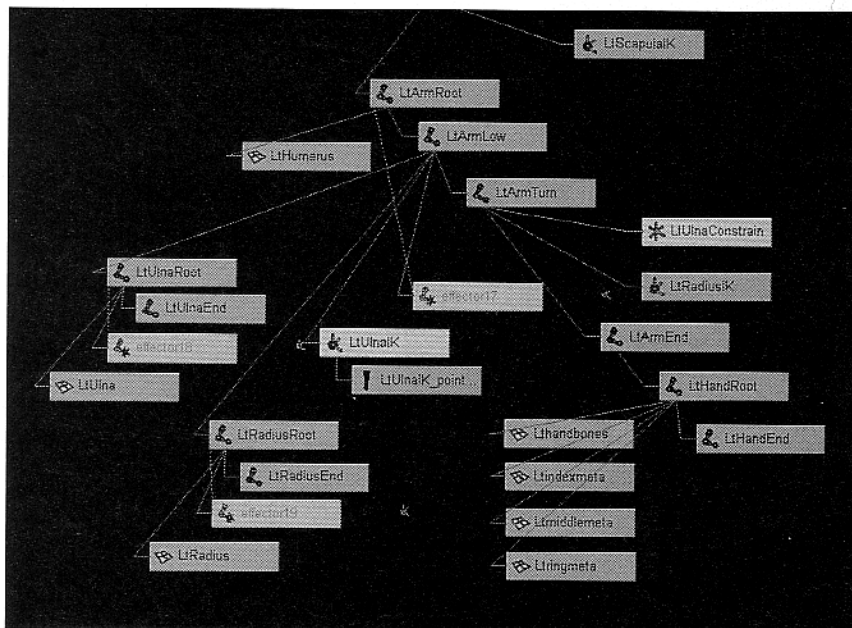


radius bone, elongates and contracts when the forearm is rotated. You can easily do this on your character by making the left radius IK handle child to the LtArmTurn joint. Make the polygon radius bone child to the radius root joint to better see the effect.

If the ulna turned all the way up its axis, however, the joint where the ulna attaches to the elbow would break. To avoid this, you cannot just make its IK handle child to the arm turn joint. Instead, you must filter out the rotation information to use only the translation information, by using a point constraint. Create a locator to do this, and move it right on top of the ulna IK by snapping or constraining, as shown earlier. Name the new locator **LtUlnaConstrain**.

To finish the forearm, constrain the ulna IK to the LtUlnaConstrain locator by selecting the locator, Shift-select the ulna IK handle, and choose **Constrain, Point**. Once constrained, make the LtUlnaConstrain locator child to the LtArmTurn joint, and make the LtUlnaIK child to the LtArmLow joint (see Figure 3.32). To see the effect on the ulna, parent the polygon ulna bone under the ulna root joint, and try rotating the LtArmTurn joint in X. Notice the radius turns completely, whereas the ulna only turns at the wrist.

- 3.32 To keep the ulna from breaking at the elbow joint, constrain the UlnaIK to a locator that is made child of the arm turn joint.

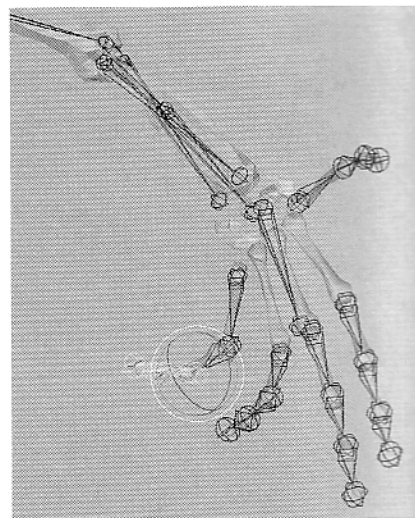


3. Create all the FK finger skeletons for the left hand. Draw the skeletons in the view that enables you to fit them properly into the finger geometry. All the fingers can be four-joint skeletons except the pinky. The pinky should have an extra fifth joint that starts close to the wrist, and represents the pinky metacarpal. The metacarpals for the other three fingers do not require skeletons because they don't move much, and the hand bone can represent them for animation. However, an extra bone for the pinky metacarpal enables you to create a cupping pose on the palm of the hand (see Figure 3.33). The thumb skeleton should also start close to the wrist, but only has four joints. When drawn, rotate and translate the thumb root joint into place. Name the thumb joints **LtThumbRoot**, **LtThumb2**, **LtThumb3**, **LtThumbEnd**, and then name all the other finger joints appropriately.

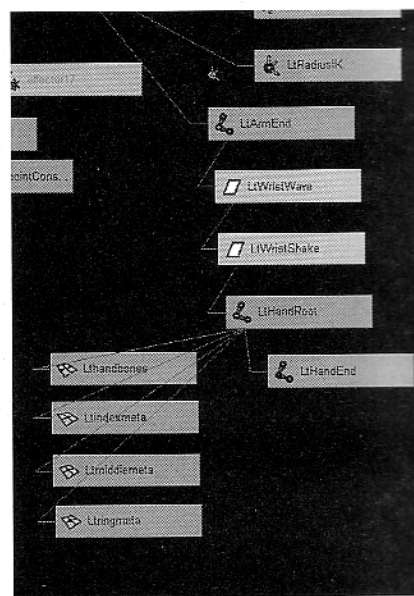
All the finger roots will be child to the end of the hand joint. Begin this by parenting the thumb root under the **LtHandEnd** joint. Notice that a bone is drawn going from the end of the hand skeleton to the start of the thumb skeleton. To keep this from happening, with the thumb root selected, group twice. Name the parent group node **LtFingerPad1**, and the child group node **LtFingerPad2**. Again, this is just to keep a lot of unnecessary bones from being drawn. Parent all the rest of the finger roots under **LtFingerPad2**.

4. After parenting all the finger skeletons, make sure that all the polygon reference bones for the left hand are made child to the appropriate joints. The small bones of the hand, as well as the metacarpals for the index, middle, and ring fingers, can all be made child to the hand root joint. When you are done, try rotating the **LtArmTurn** joint in X. You should see all the fingers rotating with the wrist.

Although rotating the **LtArmTurn** joint in X is the main rotation for the wrist, two other rotations can still be done at the wrist. You obviously do not do these movements by rotating the **LtArmTurn** joint, but by rotating the hand root joint in Z and Y. Even though you can do both these rotations on the same joint, however, it is a better idea to create two group nodes between the end of the arm and the hand root to do the wrist rotations. Do this by selecting the **LtHandRoot** joint and group twice. Name the top group node **LtWristWave** and the bottom group node **LtWristShake** (see Figure 3.34).



3.33 Creating an extra finger joint for the pinky metacarpal enables you to cup the hand.

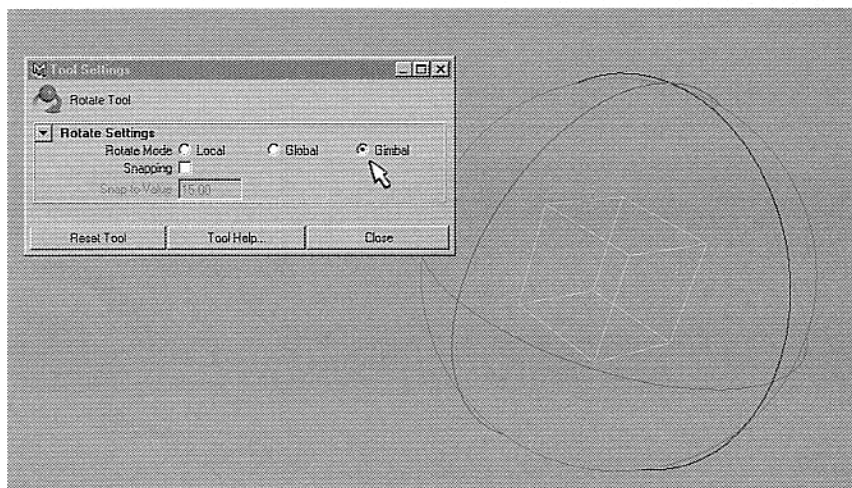


3.34 Create two group nodes between the end of the arm and the hand's root joint.

When rotating an object in more than one axis, you must consider something called the *rotation order*. Create a cube, and open the Attribute Editor to see what is set under Rotate Order in the transform node. The default rotation order on an object is always XYZ, which sets the Z-axis as the most important rotation axis, followed by Y and then X. You should set the rotation order so it reflects how you will be animating a particular object and to reduce a rotation problem called *Gimbal lock*. Gimbal lock occurs when you rotate an object in all three axes until it stops being able to rotate in the least-important axis, as set by its rotation order. You can also get variations on this problem when your object doesn't rotate cleanly in an axis but wobbles in a weird way when you set keyframes.

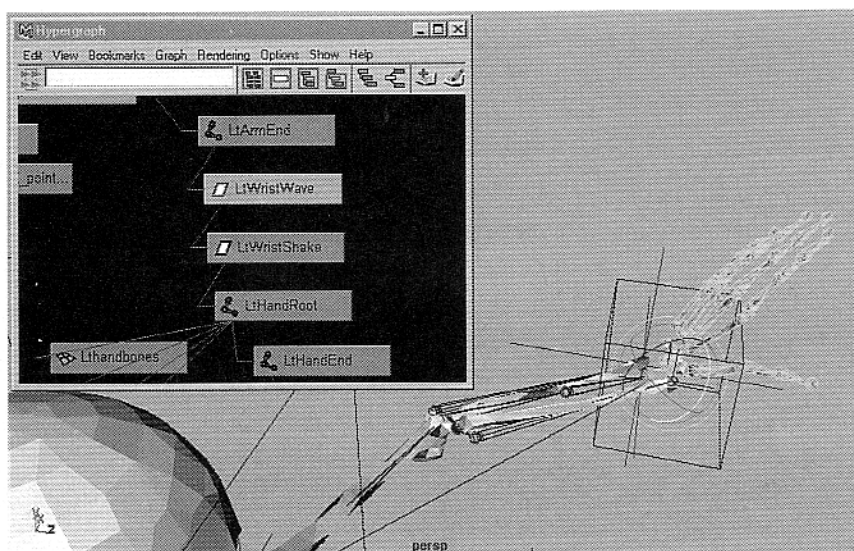
To reduce the problem of Gimbal lock, character setup artists often set the order of rotation on objects according to how that object usually moves. To see how this works, double-click the Rotate tool to set it to Gimbal mode (see Figure 3.35). This is a special rotation mode that shows you how each axis moves in relation to each other. With your cube set to an XYZ rotation order, try rotating each axis. Notice that rotating the Z-axis rotates all the other axes. Then notice that rotating the Y-axis only rotates Y and X, while the Z stays still. Rotating the X-axis doesn't rotate the X or Z. If you rotate the Y-axis 90 degrees, notice that the local X-axis of the cube is no longer available. This problem is commonly referred to as Gimbal lock. To get a better idea of how rotation order works, try switching the Rotate Order setting, and rotate the cube in Gimbal mode some more. Be aware of the rotation order on your controls when setting up your rig so that you can reduce Gimbal lock occurrences on your controls.

3.35 Set the rotation tool to Gimbal mode to see how rotation order affects an object.



Another way of setting an order of rotation is through parenting. Separating all the rotations of a control onto different nodes sets the rotation order by making the parent nodes more important than the child nodes. This keeps the rotations from conflicting with each other. For instance, use the two group nodes you created between the end of the arm and the hand joint to separate the wrist rotations. Rotating the LtArmTurn joint already does the most important rotation on the wrist, which twists the hand and forearm. Rotating the LtWristWave node in Z does the next important rotation, making the hand flap up and down. Whereas rotating the LtWristShake node in Y does the least important rotation, which is a small side-to-side rotation (see Figure 3.36). Make sure that the pivot points are correctly placed in the wrist for each rotation. Use insert mode if they need to be adjusted.

When finished with the left arm, do the same for the right-forearm and right-hand setup. If you try to mirror the entire arm hierarchy to the other side, make sure you first disconnect all polygon reference bones. Using the Joint Mirror command on geometry does not work well. In addition, mirroring a hierarchy often causes the IK to get disconnected on the new hierarchy, so you may have to reset all the IK. Sometimes it is better just to mirror individual skeletons and IK handles as you create them, and then parent them all again on the other side. ■



**3.36** By doing the hand rotations on separate group nodes, you can reduce the problem of Gimbal lock.

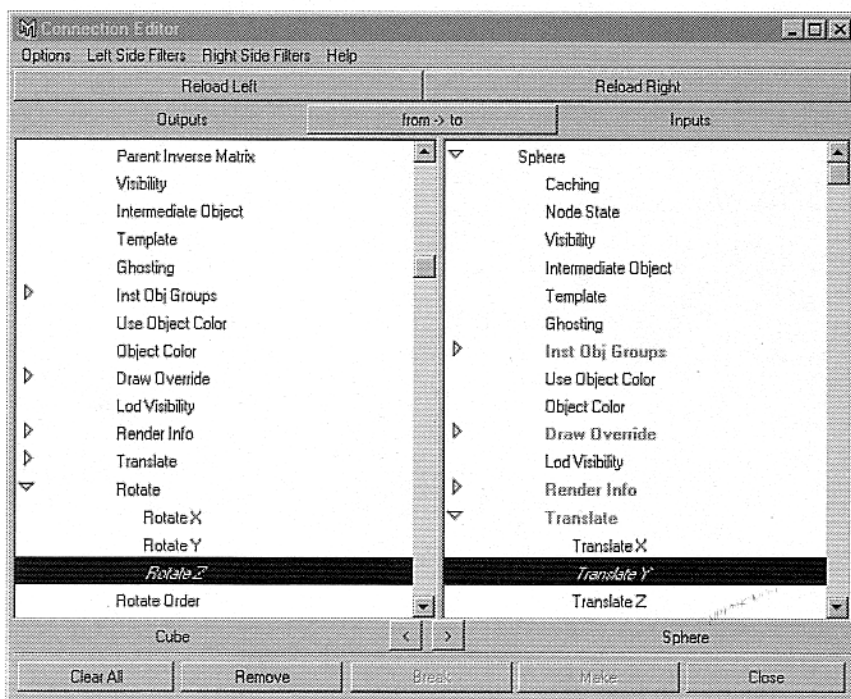


## CONNECTING CHANNELS

Throughout this chapter, you learn several ways to control the channels of objects with the channels of other objects. You have already done this through the use of constraints, such as point and aim constraints. However, these kinds of constraints do not enable you to constrain individual channels of different types, such as constraining a single translation channel to a single rotation channel. To do this, you must use either the Connection Editor, the Expression Editor, or set driven keys.

The most basic way to connect channels is to use the Connection Editor. Open the Connection Editor by choosing Window, General Editors, Connection Editor. You use the Connection Editor by loading the constraining object in the left Outputs side, and loading the object to be constrained in the right Inputs side (see Figure 3.37). The two objects have all their channels listed, and to connect two channels, all you have to do is click them in each window. Keep in mind that the constraint does not go both ways. Only the channel on the object loaded into the right side of the Connection Editor is constrained. Once constrained, the channels have a one-to-one connection based on their values displayed in the channel bar. This kind of connection is pretty simple. To make a more complex connection, where you can vary how the objects are connected, you have to insert utility nodes into the connection.

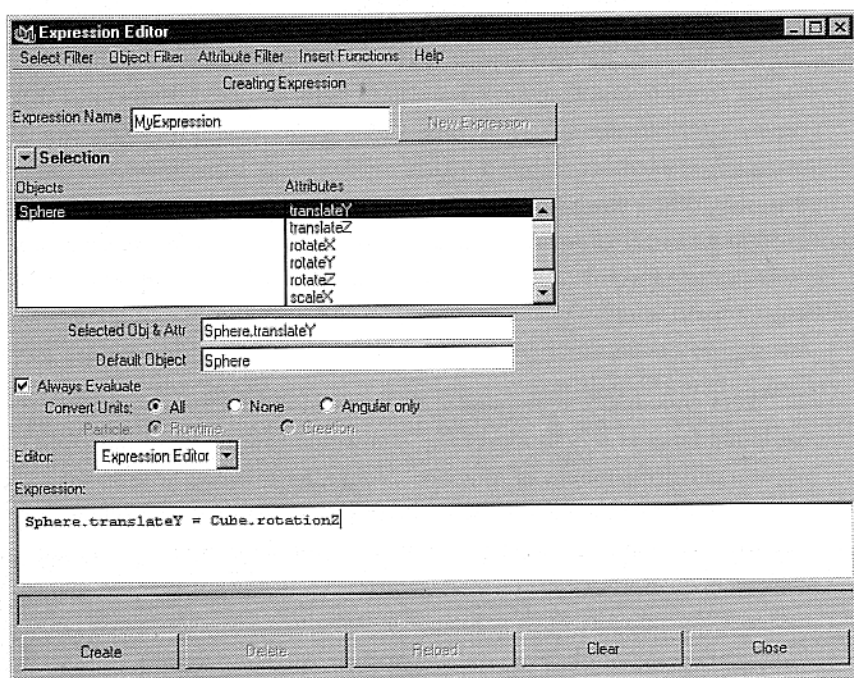
3.37 Load two objects in the Connection Editor to connect their channels.



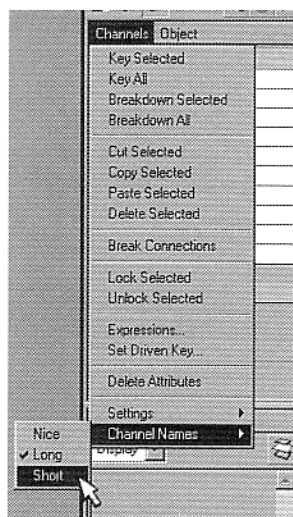
Another way to create connections between channels is by writing mathematical expressions in the Expression Editor, located under Window, Animation Editors. You type your math expressions in the white text field in the lower half of the Expression Editor (see Figure 3.38).

Expressions can constrain channels in the exact same way as the Connection Editor, but are written using math signs. Some of the basic signs used are: + (plus), - (minus), \* (multiply), and / (divide).

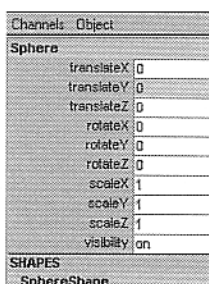
Object names must be correctly written in each expression, and their channels must be specified. It is a good idea, therefore, to name your objects with short, logical, descriptive, names. Everything in an expression is case sensitive; so make sure you are consistent in how you use capitalization in the names of your objects. All channel names are also case sensitive. In fact, you should be aware that the default names in the channel bar are not how the actual channel names should be written. If you choose Channels, Channel Names on the channel bar, three choices display. The default setting of Nice is not accurate. It capitalizes the first letter in the name, and puts a space between parts of the channel name. Using this syntax in your expressions will give you an error message. Instead, switch the Channel Names setting to either Long or Short (see Figure 3.39). The syntax for both of these settings is accurate, and will work in your expressions. If you want to speed up your workflow by typing less, use the short versions of the channel names. After you create the expression, however, Maya converts the names to the long versions.



3.38 To connect channels in the Expression Editor, write math expressions in the white text field that specify how to connect the objects.



- 3.39 In the Channel box, change the channel names to either the long or short versions when writing expressions.



- 3.40 Connecting two channels in the Connection Editor creates a constraint on the channel of the object in the Inputs field.

The syntax for writing a basic expression that creates a one-to-one constraint between channels is written like this:

```
ConstrainedObject.channel = ConstrainingObject.channel
```

Keep in mind that only the object on the left side of the equals sign is constrained. The other object's channels are not affected at all. This kind of expression is extremely simple, but adding some simple math symbols can refine it.

### EXERCISE 3.7

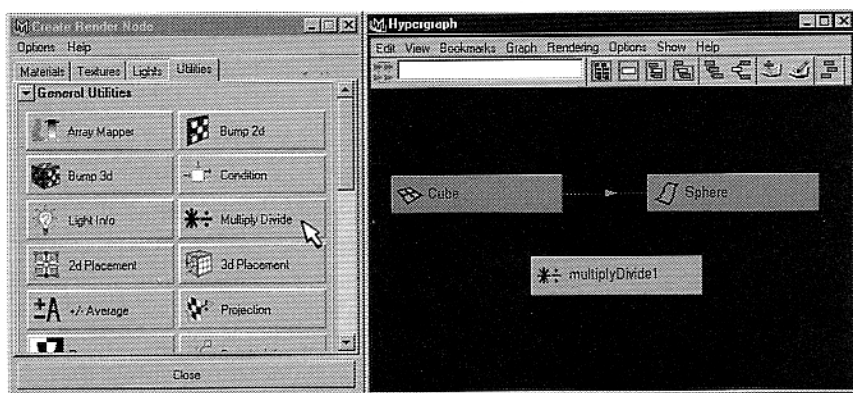
#### CREATING BASIC CHANNEL CONNECTIONS

1. To get some experience connecting channels, in this exercise you constrain channels using both the Connection Editor and Expression Editor. First, in a new empty scene, create a polygon cube and NURBS sphere under the Create menu. Name the objects **Cube** and **Sphere**, and translate the cube in X so that it is not sitting on top of the sphere. Then open the Connection Editor by choosing Window, General Editors, Connection Editor. Select the cube and load it into the left Outputs side of the Connection Editor by clicking the Reload Left button. When loaded, find the channel named Rotate and click the arrow beside it to show the individual rotation channels. Click the Rotate Z channel.

Next, select the sphere and load it into the right Inputs side of the Connection Editor by clicking the Reload Right button. To connect the channels between the two objects, click the arrow beside the channel named Translate, and then click Translate Y. Notice in the Channel box that the Y channel for the sphere becomes colored. This means the channel is being constrained (see Figure 3.40). If you select the cube, however, notice that none of the channels are constrained. Rotate the cube in Z to see the sphere move up and down in Y. Also notice that when you rotate the cube in a negative direction, the sphere moves negative in Y. It also moves the exact same amount of grid units that the cube rotates in degrees. This is a one-to-one connection between channels.

The kind of connection you just made between the cube and the sphere has limited uses as it is. In most cases, you would have to adjust this connection either in amount or direction. Notice, for instance, how far the sphere goes when the cube rotates. This is because you are controlling translation, which is based on grid units, with rotation, which is based on degrees. A quarter turn of the cube, which is 45 degrees, makes the sphere translate 45 grid units. Such a distance is going to be way too much for most character controls. Even on a large character, most controls do not move more than a few grid units.

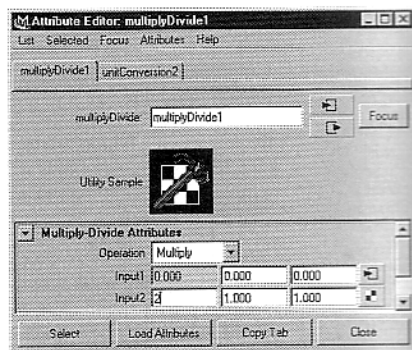
2. To change the amount that a channel constrains another channel in the Connection Editor, you must use a utility node. Open a hypergraph view, and with both the cube and sphere selected, click the window icon for Input and Output Connections. Then choose Rendering, Create Render Node in the Hypergraph window menu bar. In the Render Node options box, click the Utility tab, and choose the Multiply Divide node under General Utilities (see Figure 3.41). Load the cube into the left Inputs side of the Connection Editor, and select the Multiply Divide node in the hypergraph to load it into the right Outputs side of the Connection Editor. Connect the Rotate Z channel of the cube to the Input1 X channel on the Multiply Divide node. Then load the Multiply Divide node into the left side of the Connection Editor, and load the sphere into the right side. Connect the output of the Multiply Divide node's Output X channel to the inputs of the sphere's Translate Y channel. ■



3.41 In the hypergraph view, create a utility node to adjust a constraint created in the Connection Editor.

If you refresh the hypergraph view, you will see connection arrows going from the cube node to the utility node to the sphere node. Right-click the Multiply Divide node, and choose Attribute Editor. The way the Multiply Divide node works is that the value in Input2 is performed on Input1. So if you set the operation to Divide, for instance, and you set the value of Input2 to 10, the rotation of the cube is divided by 10 (see Figure 3.42). If you then rotate the cube 45 degrees, the sphere translates in the Y-axis only 4.5 grid units, rather than 45 grid units. Try experimenting with other utility nodes, such as the Reverse node, which enables you to reverse the direction of the control.

You can create the same kind of connection between the cube and sphere using a mathematical expression. First, delete the Multiply Divide node in the hypergraph view, which deletes the constraint on the sphere's Translate Y channel. Then open the Expression Editor and type the following in the Expression field:



3.42 Use the Multiply Divide node to increase or decrease the result of the connected channels.



```
Sphere.translateY = Cube.rotateZ
```

After you have written this, click the Create button in the lower-left of the Expression Editor. Select the cube and rotate it in Z to see the sphere move. Notice that the connection is exactly the same as the basic connection that you previously did in the Connection Editor. If the expression disappears from the Expression field, you can locate it again by switching the Select Filter setting from By Object/Attribute Name to By Expression Name, and choose it in the List field (see Figure 3.43). In general, it is better to work in the By Expression Name filter because changing selections on objects will not affect the current expression you are writing. Your new expression should have the default name Expression1. You can then edit the expression to change the name, and adjust how the cube affects the sphere.

You can do several simple things to fine-tune your expression. For instance, the preceding expression implies the following:

```
Sphere.translateY = 0 + Cube.rotateZ
```

This expression reads “the sphere’s translation in Y is equal to zero plus the cube’s rotation in Z.” The zero in the expression represents the start number or the current value in the sphere’s Y channel. Change this number so that the expression looks like this:

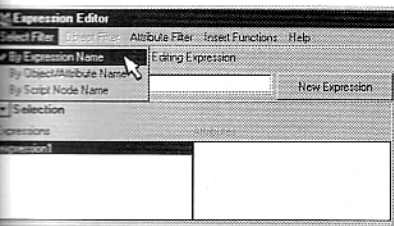
```
Sphere.translateY = 5 + Cube.rotateZ
```

When you click the Edit button in the Expression Editor, notice that the sphere moves 5 units up in Y. Nothing else is changed about the expression except the start position of the sphere. If you change the number to -5, the sphere will be set to -5 in Y as its start position. The other thing you could change is the direction of the constraint. Change the expression to read as follows, and then click the Edit button:

```
Sphere.translateY = 0 - Cube.rotateZ
```

When you rotate the cube, notice that the sphere goes in the opposite direction than it did before. Changing the plus to a minus makes the sphere move in a negative direction when the cube rotates in a positive direction, and vice versa. If you want to reduce the amount of the constraint effect, as done previously with the Multiply Divide utility node, use a division symbol like this:

```
Sphere.translateY = 0 - Cube.rotateZ / 10
```



43 In the Expression Editor, change the selection filters to edit an expression.

Or you could multiply the amount like this:

```
Sphere.translateY = 0 - Cube.rotateZ * 10
```

The expression syntax for a basic channel constraint can be summarized as follows:

```
ConstrainedObject.channel = Default#(Start) +-  
(Direction) ConstrainingObject.channel */#(Amount)
```


As you can see, it is relatively easy to create a very specific constraint using mathematical expressions. I personally prefer using expressions to using the Connection Editor with utility nodes to constrain channels. Utility nodes do evaluate a little faster than expressions, however, and some professionals prefer using them for their controls. For the rest of this chapter, you use expressions for doing all constraints that involve math operations. Keep in mind, however, that in most cases you could also use utility nodes and the Connection Editor to do the same kind of constraints. It would just take you a little longer to set up.

## CONTROLLING THE BACKBONE WITH MATH EXPRESSIONS

You may have noticed that the basic FK backbone on your character cannot be selected very easily. To animate it bending, you have to manually select and rotate each one of its three main joints. Because this is not very efficient, you create some controls to make animating the backbone easier. Instead of rotating each joint individually, you use math expressions to constrain the rotation of all the joints to a single control icon. This is much easier for you to select and animate.

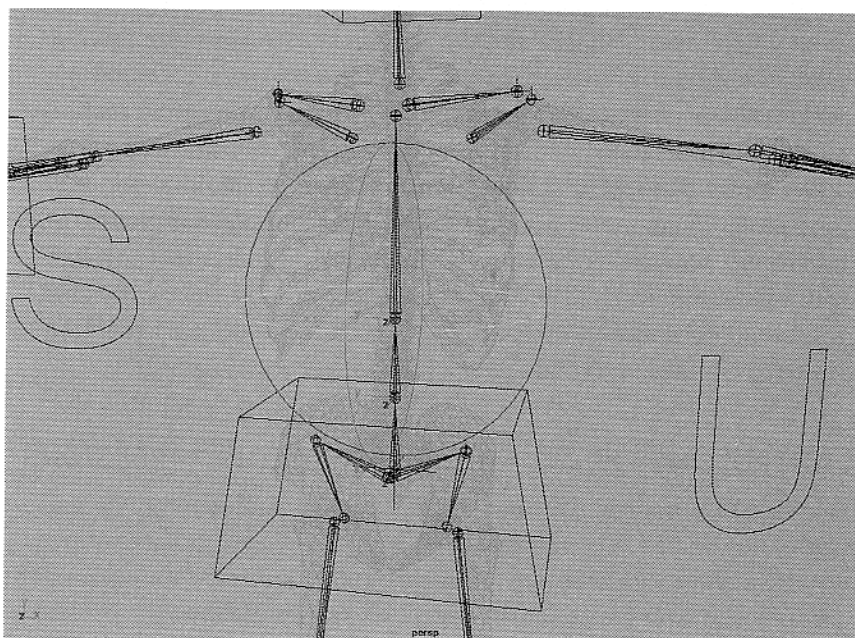
### EXERCISE 3.8

#### MAKING BASIC BACKBONE CONTROLS

1. Open the scene that contains the basic character rig you have been building. To create some control icons for rotating the backbone joints, choose Create, NURBS Primitives, Circle . In the Circle options box, set the Object Normal to Z, and click Create. Scale the circle so that it is slightly larger than the width of your character's torso, and then freeze its transforms. Name this circle **BackBend**. Then duplicate the BackBend circle, and name it **BackBow**. Rotate the BackBow circle 90 degrees in Y, and also freeze it. Finally, duplicate the BackBow circle, and name it **BackTwist**. Rotate the BackTwist circle 90 degrees in Z, and freeze it.

At this point, you should have three circles sitting at the global origin. Select all three circles and group them, naming the parent group node **BackControls**. In the hypergraph view, parent the BackControls node under the UpperBody icon. You may also want to translate the BackControls to better position them in relation to your character (see Figure 3.44).

- 3.44 Place three circle icons around the middle of your character's torso to control the backbone skeletons.



2. When you have some control icons for the backbone, you can begin connecting channels in the Expression Editor. Keep in mind that your expressions must be based on how the centers of your backbone joints are oriented. If you created your backbone in the front view, Z should be pointed forward, Y to the side, and X oriented toward the next joint in the skeleton. With this orientation, BackBend controls the Z-axis, BackBow controls the Y-axis, and BackTwist controls the X-axis on the joints.

To check how the centers on your backbone joints are oriented, select them and choose Display, Component Display, Local Rotation Axes. Specifically, look to see whether all the centers have the same orientation. If you didn't draw the backbone perfectly straight, some of the centers on the joints may be flipped 180 degrees in X. This occurs whenever you change directions when drawing a skeleton. Because this changes how your expressions affect the joints, it is a good idea to set all the centers so that they have the same orientation. Do this by switching to component mode, and after turning on the question mark (?) symbol, select a center axis that needs to be fixed. Do not type any values in the transform channels to

rotate the center. Instead, activate the Rotation tool, and manually rotate the center a little in X. You won't be able to tell how far the center was rotated, but that doesn't matter. Open the Script Editor and find the last line in the gray History field. It should look like this:

```
rotate -r -os 23 0 0;
```

Highlight this line in the Script Editor, and use the middle mouse button to drag it down into the white scripting field. Then change the first number in the code to **180**. It should look like this:

```
rotate -r -os 180 0 0;
```

After you have done this, highlight the code again, and use the middle mouse button to drag it to your shelf. This action creates a shelf button to flip your centers around 180 degrees in X. You can open the Shelf Editor to give the button a short name such as RotX. Close the Script Editor, and press Z to undo the preceding rotation. Then, with the center still selected, click your new shelf button to rotate the center exactly 180 degrees in X. You can use the button to rotate any other centers. Then turn off the question mark symbol, and switch back to object mode. When all the centers are oriented correctly, begin connecting your backbone controls by typing the following expression into the Expression Editor:

```
BackRoot.rz = 0 + BackBend.rz;
```

Name the expression **BackRotate** and click the Create button. Notice that this expression uses the short versions of the channel names, and ends with a semicolon. The semicolon terminates a line in an expression, and is necessary if you are going to write more than one line. After you create the expression, try rotating the BackBend circle in Z. You should see the first joint of the backbone rotating side to side in Z.

Switch to the Expression Name selection filter if you have not already done so, and click the BackRotate expression to edit it. Highlight the line you just wrote, and copy it by pressing Ctrl+C. Then click the Enter key to go to the next line. Paste the line down by pressing Ctrl+V. Copying and pasting makes it easier for you to create new lines when you need to make similar expressions. Do this twice, and then change the backbone names so that you have the following three lines in your expression:

```
BackRoot.rz = 0 + BackBend.rz;
Back2.rz = 0 + BackBend.rz;
Back3.rz = 0 + BackBend.rz;
```

3. After you click the Edit button, try rotating the BackBend circle again. You should see all the backbone joints rotating in the same direction as the circle. This should make the backbone bend in a smooth manner from side to side. To fine-tune this motion, you can divide the effect on some of the



joints. The lower part of a real backbone doesn't bend as much as the upper part, so you may want to adjust your expressions in a similar way to the following:

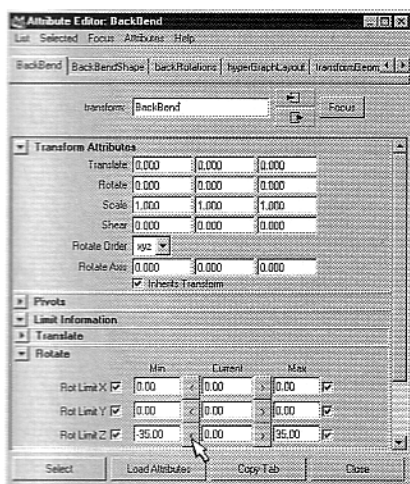
```
BackRoot.rz = 0 + BackBend.rz / 4;
Back2.rz = 0 + BackBend.rz / 3;
Back3.rz = 0 + BackBend.rz;
```

You can divide or multiply your expressions as needed for your character. Keep in mind that you do not have to get the rotations perfect at this stage. Try to get them close to how they should rotate, and then you can fine-tune them after you bind the models to your character, when you can better see the effect of the rotations on the skin. To finish your backbone connections, type expression lines for the rest of your backbone channels, and click the Edit button. Your final expressions will look similar to the following:

```
BackRoot.rz = 0 + BackBend.rz / 4;
Back2.rz = 0 + BackBend.rz / 3;
Back3.rz = 0 + BackBend.rz;
BackRoot.ry = 0 - BackBow.rx / 4;
Back2.ry = 0 - BackBow.rx / 3;
Back3.ry = 0 - BackBow.rx;
BackRoot.rx = 0 + BackTwist.ry / 4;
Back2.rx = 0 + BackTwist.ry / 3;
Back3.rx = 0 + BackTwist.ry;
```

Notice that the second set of expressions for the backbone's rotation in Y uses a minus sign rather than a plus sign. If your backbone centers are facing negative rather than positive in Y, you must adjust your expression accordingly. You can start by just using a plus sign, and if your joints are rotating in the opposite direction from your circles, just change the plus sign to a minus sign. Use this trial-and-error method when writing all your expressions.

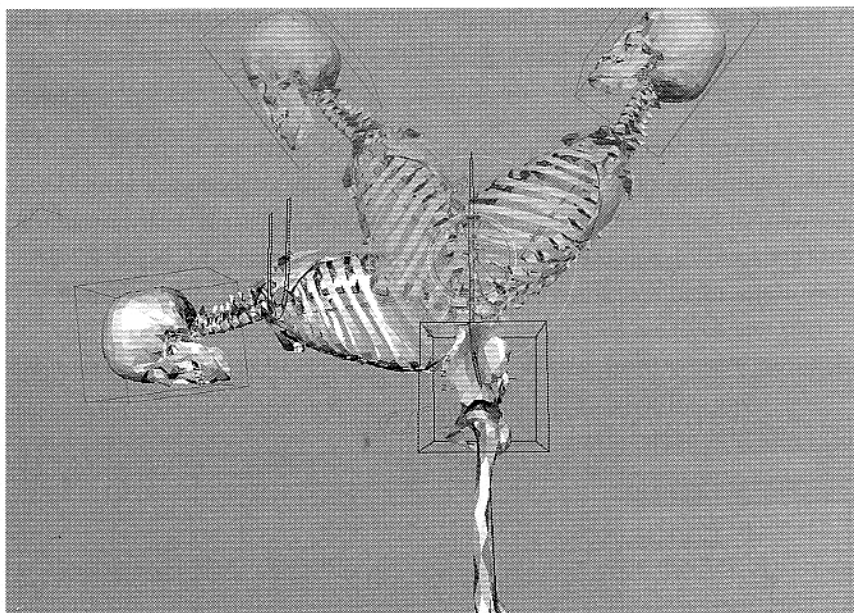
- One last thing you should do is set some rotation limits on your circle icons. A general rule to keep in mind when setting up your rig is to put limits only on the objects you are going to animate directly. So in this case, you should place limits on the circle icons, and not on the joints they control. Right-click the BackBend circle, and choose the Attribute Editor. In the Transform tab named BackBend, click the Limit Information drop-down arrow to see the rotation limit channels. The BackBend circle should rotate only in Z, so click the arrows beside the Min and Max for X and Y, which should set their values to 0. Click the empty boxes to set the limits. Then rotate your BackBend circle positive in X until it is as far as you want your character's backbone to bend to the side, and click the arrow next to Max in the X limits. Copy the Max value, turn on the Min limit, and paste the same value with a negative sign added (see Figure 3.45). In



**3.45** Set the BackBend circle's rotation limits in the Attribute Editor under the Transform tab.

addition, make sure you click all the empty boxes to turn on the limits for each axis. Test your BackBend circle to make sure it rotates only in Z the amount you set.

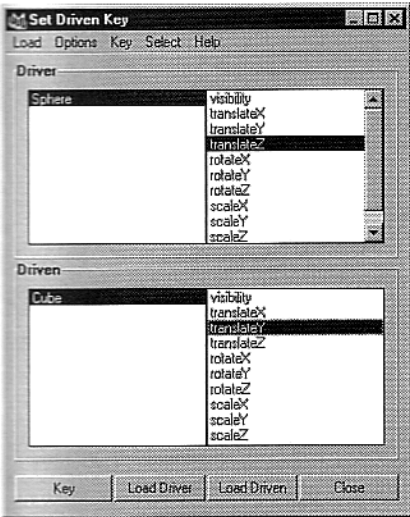
Follow this same process to set limits on the other two circles. The BackBow circle should rotate only in X, and should rotate more to the front than to the back (see Figure 3.46). The BackTwist circle should rotate only in Y. When you have finished setting the limits, you may do one more thing to the circles. You may want to color the icons to cue the animator how they should be animated. For instance, you could make each circle the color of the channel it should be rotated in. Do this in the Shape tab of the Attribute Editor for each circle, by turning on Enable Overrides in the Object Display, Drawing Overrides section. Adjust the Color slider to set the color. Make BackBend blue, BackBow red, and BackTwist green. ■




**3.46** Set the limits so that rotating the BackBow circle makes the backbone bend more forward than backward.

## SETTING DRIVEN KEYS FOR SHOULDER MOTION

The third main way of connecting channels in Maya is to set driven keys. These are special keys that are not based on the timeline, but instead are based on the connected channels' relationship to each other. The object with the constraining channel is the *driver*, and the object with the channel constrained is the *driven*. Setting driven keys has some distinct advantages over other methods for creating basic constraints between channels. Driven keys are easy to set, and are extremely flexible. If you need to do math operations in your controls, however, you will still have to use expressions or the Connection Editor.



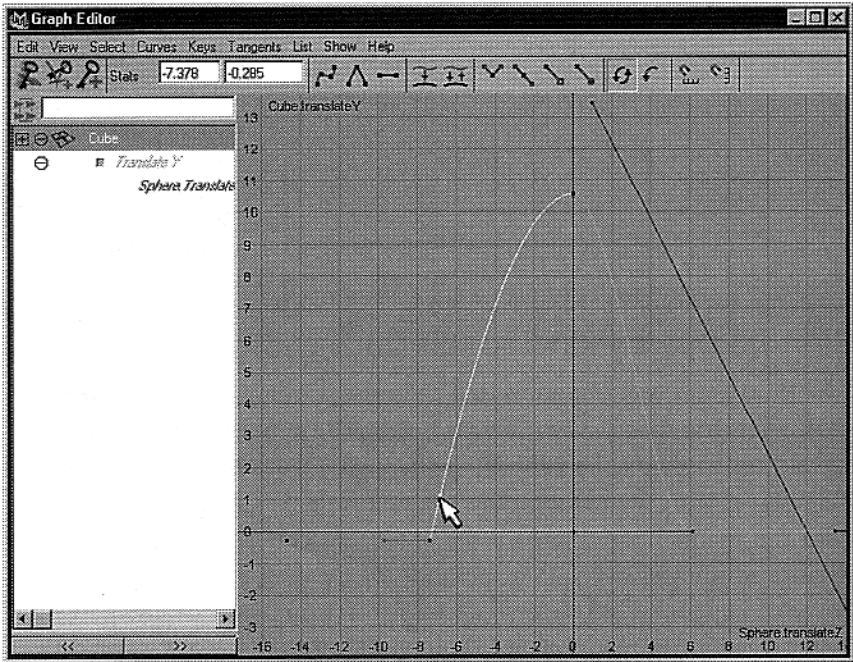
3.47 Set driven keys to connect channels by loading objects into the Driver and Driven sections of the Set Driven Key options box.

To set driven keys, choose **Animate, Set Driven Key, Set** . In the Set Driven Key options box, you load the driver and driven, and click the channels you want to connect (see Figure 3.47). Press the Shift key to select multiple driven channels. Once loaded, you manipulate the channels in relation to each other, and click the **Key** button for each new position. After the keys are set, whenever the driver's channel changes, the driven channel responds.

One main difference between setting driven keys and other connection methods is that driven keys create animation curves that can be edited in the Graph Editor. This enables you to adjust the timing of a constraint in ways that would be difficult using other methods. An expression constraint occurs in a constant manner, for instance, which would be equivalent to a linear animation curve in the Graph Editor. A driven object, however, can respond at a variety of rates to a driver's motion. A driven object can speed up or slow down over the course of a move, which is equivalent to a spline-based animation curve in the Graph Editor (see Figure 3.48).

An additional difference from other constraint methods is that driven objects can have multiple drivers. The driven key's channel is constrained, so you can't set regular animation keys on it manually, or attach another kind of constraint to it. But you can connect additional drivers to it. Being able to connect multiple drivers to a single driven allows

3.48 A driven key connection creates spline-based animation curves that you can edit in the Graph Editor.



you a great amount of flexibility when setting up your character controls. For instance, in the next exercise, you control the raising of your character's individual shoulders with three drivers. This creates automatic shoulder movement based on the position of the arm box and elbow icon, and allows you to use a custom channel for manually controlling the raising of each shoulder.

### EXERCISE 3.9

#### SETTING DRIVEN KEYS ON THE SHOULDER

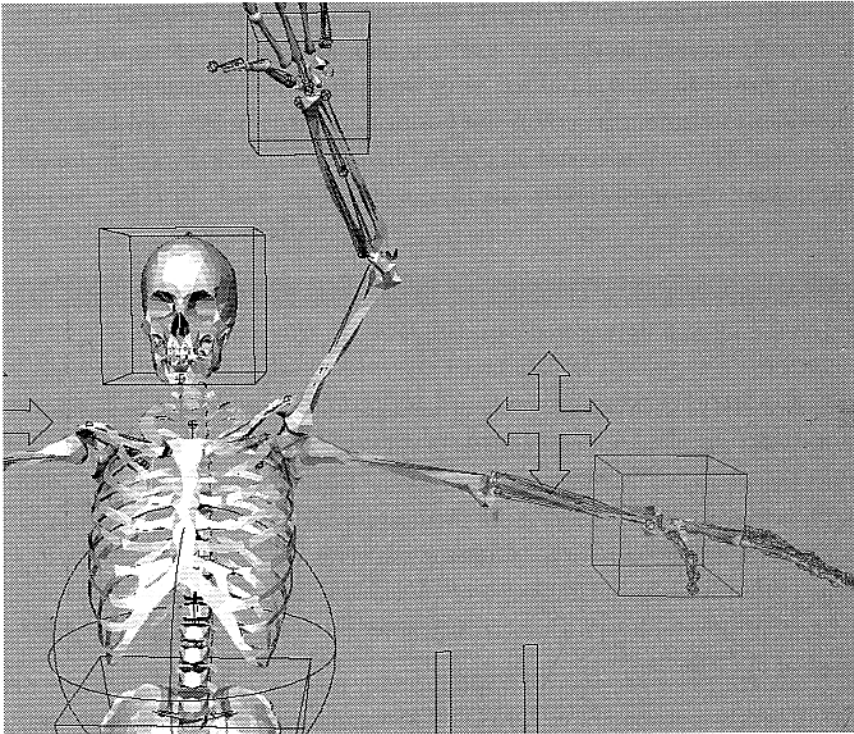
The first driven keys you are going to set make the left shoulder rise automatically when the left-arm box translates in Y. The shoulder in a real body is where the clavicle, scapula, and humerus bones come together. This area rises automatically when the elbow moves above shoulder level. This occurs because there is a notch on the ball joint of the humerus that clicks into place on the clavicle when the arm rises, which then pushes the clavicle up. You set driven keys to make this happen on your character whenever the arm box is raised past the level of the shoulders.

1. Begin by opening the Set Driven Key options box, and with the left-arm box selected, click the Load Driver button. Choose the Translate Y channel of the arm box in the Driver Channels list. Then load the LtClavicleIK as the driven object, and also choose its Translate Y channel. Before continuing, select the arm box and press the S key to set keys for its default position at frame 1 on the timeline. Keep in mind that this key on the arm box has nothing to do with the driven key connection. You move the arm box around a lot while setting driven keys, and this enables you to easily get the arm back to its default pose when you have finished.

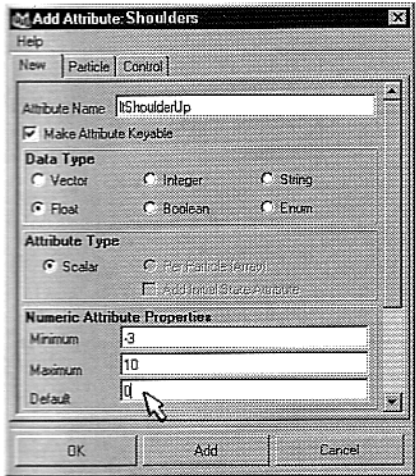
The first driven key to set is the default position of the clavicle. Place the arm box on the same level in Y as the clavicle, and click the Key button in the Set Driven Key options box. Then translate the arm box above your character's head until the elbow locks out. Select the clavicle IK and translate it up in Y slightly. The arm should bend a little as the clavicle is raised. Set another driven key. To check the control, translate the arm box up and down to see whether the clavicle moves smoothly in Y (see Figure 3.49). Afterward, click the timeline to force your arm box back into its default position.

2. Select the elbow arrow icon, and load it as a second driver for the clavicle IK. Choose its translation in Y as the driving channel. To see how this control should affect your character, rotate your own elbow forward and upward, so your palm faces behind you. Notice your clavicle goes up.

3.49 Set driven keys on the clavicle IK to make the shoulder move up and down when an arm is raised.



Before setting driven keys, as with the arm box, press the S key to set a key for the elbow icon's default position. Make sure the elbow icon is directly behind the lower arm joint, and set a driven key. Then raise the elbow icon in Y so the arm rotates about 80 degrees forward. Also raise the clavicle IK a little bit, and set another driven key. When your keys are set, test the control, and then click the timeline to reset your arm controls into their default position. After testing the automatic shoulder controls, remove the keys you set on the arm and elbow controls by selecting their channels in the channel bar, and right-click to break the connections.



3.50 Create a custom channel on the Shoulders icon to manually drive the clavicle IK up and down.

3. You may have noticed a problem after setting your automatic controls for the left shoulder. After you set driven keys, you cannot manually set translation keys on the clavicle IK. This can still be done, however, by creating a channel that manually drives your left shoulder. The best place to create such a channel is on the Shoulders icon. Select the Shoulders icon and choose Modify, Add Attribute on the top menu bar. In the Add Attribute options box, name the new channel **ltShoulder** (see Figure 3.50). Make sure that you are using a Float data type, which gives you smooth transitions in your new channel. Then set the Minimum field to -3, the Maximum field to 10, and the Default field to 0. Setting the Numeric Attribute Properties creates the limits on the channel. The reason you set the minimum to a smaller amount than the maximum limit is because a shoulder mostly moves up, rather than down.



After you have a left-shoulder channel on your Shoulders icon, you can make it drive the manual translation in Y of your character's clavicle IK. Load the Shoulders icon into the Driver section of the Set Driven Key options box, and choose the new *ltShoulder* channel. Make sure the driven channel is set to the translation in Y of the left-clavicle IK. Make sure the driver channel is at 0, and the clavicle IK is in its default position, and click Key. Then change the driver channel to 10, raise the clavicle IK in Y, and click Key again. Finally, change the driver channel to -3, and lower the clavicle IK to slightly below its default position, and key it.

When all the driven keys are set, select the *ltShoulder* channel in the channel bar, use the middle mouse button as a virtual slider, and scrub through the driven keys. As the channel changes, you should see your character's left shoulder moving up and down accordingly. Try out all three of the shoulder controls to see how they work together. Keep in mind that you can set more driven keys to refine this motion. You may want to drive some X translation channels in addition to the Y translation. You may also want to drive the arm root joint's translation to make it rise with the clavicle IK. When the left shoulder is working well, set similar driven keys on the right shoulder. ■

## CREATING ADDITIONAL SDK CONTROLS

In the preceding exercise, you made a custom channel to control the translation in Y of the clavicle IK. Doing this enabled you to set manual translation keys on a channel that was already being controlled by driven keys. Even if there were no other driven keys on the clavicle IK, however, creating a custom channel on the Shoulders node would still be desirable.

You may have noticed that some objects in your rig do not have easily selectable controls. If you made icons for every control in your character, your interface would soon get cluttered. Some objects in your rig that are difficult to select are the individual shoulder IK handles, the arm turn joints, wrist pivots, finger joints, and the jaw joint. The best place to put control channels for these objects is on the closest main control icon. For instance, arm, finger, and hand channels can be placed on the appropriate arm box, and jaw controls can be placed on the Head box. You can then connect all these channels to the objects they should control by setting driven keys.

## EXERCISE 3.10

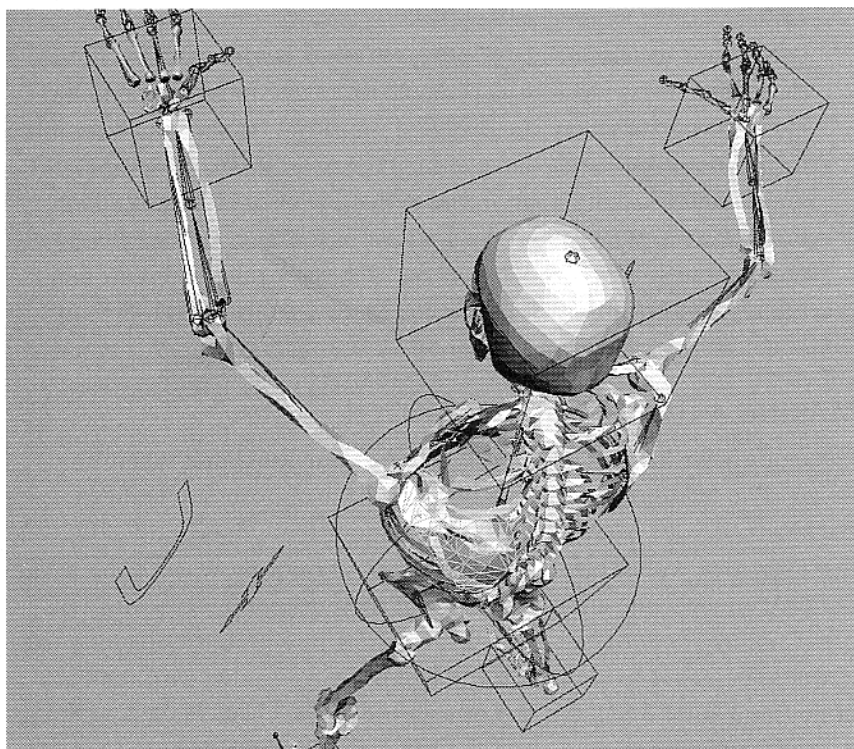
## SETTING DRIVEN KEYS ON OTHER PARTS OF THE BODY

1. Another automatic motion you can add to your shoulder is movement in Z when the arm moves forward and backward. This involves the Z translation of the clavicle IK, and some X and Z translation of the scapula root joint. Load the translation in Z of the arm box as driver, and load the Z translation of the clavicle IK as driven. Set driven keys with the arm box out in front of your character, and back behind your character about 30 degrees.

Driving the clavicle IK in Z rotates the scapula a little because the scapula IK is child to the clavicle IK. When a real arm moves forward and backward, however, the scapula actually floats around the rib cage to some degree. You can best experience this motion by stretching both your arms backward, and then notice how your scapulas press together. To make the scapula translate around the rib cage, load the scapula root joint as driven, with the translation in Z of the arm box still as the driver. Hold down the Ctrl key to select the translation in X and Z of the scapula root joint. Then key the translations of the scapula to make it move in a small arc around the rib cage as the arm box moves forward and backward (see Figure 3.51).

2. You can add several channels to each arm box to control your character's forearms, hands, and fingers. Begin by making three custom channels on the left-arm box named **wristTurn**, **handWave**, and **handShake**. Set the minimum for these channels to -10, and the maximum to 10. Open the Set Driven Key option box and load the wristTurn channel of the arm box as the driver. Load the LtArmTurn joint's X rotation channel as driven. Then set a driven key with the arm turn joint in its default position, and the wristTurn channel at 0. Then change the channel to 10, rotate the arm turn joint forward in X about 45 degrees, and set a driven key. Keep in mind that the degree to which a real forearm turns is very little. Most of the turning on the arm occurs from the shoulder, and should be done by translating the elbow icons. Change the wristTurn channel to -10, rotate the arm turn joint backward in X about 100 degrees, and set a key. Scrub the wristTurn channel to see the forearm turning.

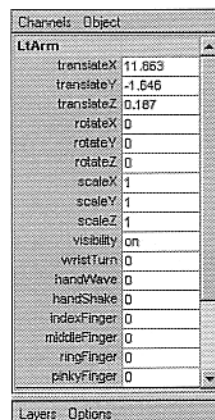
Next, use the same method to make the other two custom channels on the arm box drive the two group nodes that are parent to the hand joint. Make the handWave channel drive the up-and-down rotation of the LtWristWave node, and the handShake channel drive the side-to-side rotation of the LtWristShake node. Depending on which way the hands of your character are facing, the two driven channels will probably be Z and Y. Keep in mind when setting driven keys that the wave rotation is larger than the shake rotation. After you have all the driven keys set, you should be able to use the three custom channels to place your character's hand into any pose.



**3.51** Also use driven keys to make the scapula translate around the rib cage in an arc when the arm is moved forward and backward.

To set driven keys for the fingers, create five channels on the arm box named **indexFinger**, **midFinger**, **ringFinger**, **pinkyFinger**, and **thumb** (see Figure 3.52). Because fingers only go in one direction, you may not need much of a minimum value on the channels. This depends on whether your character's fingers are straight or relaxed in its default pose. If they are relaxed, you will have to put a minimum value that is large enough to straighten them out. Set the maximum value for each finger channel to 10.

Then load the **indexFinger** channel of the arm box as the driver. Hold down the Shift key in the hypergraph view to select **LtIndexRoot**, **LtIndex2**, and **LtIndex3**, and then load all three index finger joints in as the driven. Select with the Shift key all the finger joints in the Set Driven Key options box, and choose their rotation in Z as the driven channel. Set a key in their default position, with the driving channel at 0. Then change the driving channel to 10, rotate all the index finger joints into a closed position using the Up and Down Arrow keys to move through the finger hierarchy, and set another key. Set the driving channel to the minimum value, rotate all the index finger joints until they are straight, and set a final key. Repeat this process with all the finger joints. In addition, you can create **fingerClosed** and **fingerSpread** channels on the arm box that drives all the fingers at once. The **fingerClosed** channel should drive the Z rotations of all the finger joints, and the **fingerSpread** should drive the Y rotations of the finger root joints to spread the fingers away from each other.



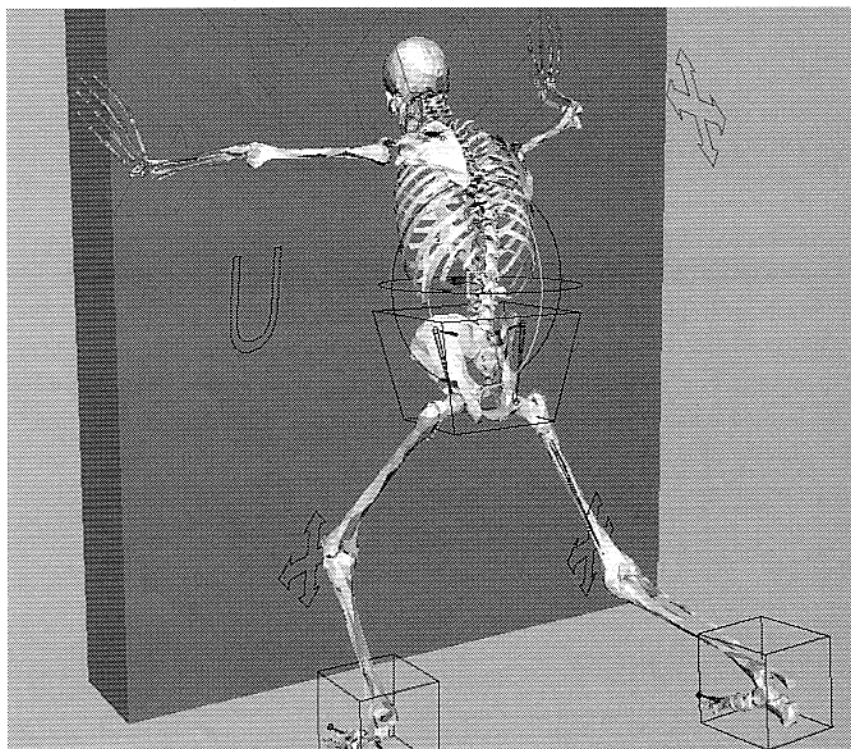
**3.52** Create several custom channels on each arm box to control the forearm, hand, and finger motions.

3. Create a few custom channels on the Head box to move the jaw and neck. Name the channels **jawOpen**, **jawGrind**, **jawJut**, **neckBend**, and **neckTilt**. The **jawOpen** channel should rotate the jaw joint in Z so the mouth opens and closes. The **jawGrind** and **jawJut** channels should translate the jaw from side to side, and translate the jaw forward and backward. Set the minimum and maximum values to the appropriate values to achieve these movements. Then load the jaw joint in as driven, and key the appropriate channels. The **neckBend** and **neckTilt** channels should make the NeckRoot joint rotate side to side and forward and backward. ■

## MAKING A HANDS-FOLLOW SWITCH

Currently your hands should be following your backbone when it moves. The reason this occurs is because the arm boxes are child to BackPad2. If your hands were parented under the UpperBody icon, however, they would move with the torso, and not move with the backbone. At times while animating, both of these parenting solutions will prove useful. If your character was pushing a heavy object, for instance, it would be easier to animate if the hands did not move with the backbone (see Figure 3.53). On the other hand, if the character was walking, it would be easier to animate if the hands follow the backbone. Actually, both these solutions are possible if you use constraints.

3.53 It is easier to make your character push a heavy object if the hands do not follow the backbone.



## EXERCISE 3.11

## USING CONSTRAINTS TO CREATE A HANDS-FOLLOW SWITCH

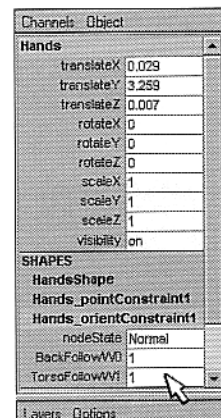
1. Create a locator named **Hands**, and make the Hands locator the parent of both arm boxes. Create two other locators named **BackFollow** and **TorsoFollow**. Make sure all three locators are sitting directly on top of each other. The BackFollow locator should be the child of BackPad2, and the TorsoFollow locator should be the child of the UpperBody icon. The trick to creating a switch for how the arms follow the backbone is to create two constraints that make the Hands locator switch between following BackFollow and TorsoFollow.

Select the BackFollow locator, Shift-select the Hands locator, and choose Constraint, Point on the top menu bar. With both still selected, go to the Constraint menu again and choose Orient. This places both a point and orient constraint on the Hands locator. Select the TorsoFollow locator, Shift-select the Hands locator again, and create both point and orient constraints. If you select the Hands locator, you should see two point constraints and two orient constraints in the Inputs section of the Channel box (see Figure 3.54). Having these two constraints is the key to creating a switch. Notice that each constraint has a weighting that goes from 0 to 1. When the channel is at 0, the constraint is turned off, and when it is at 1, the constraint is fully turned on.

2. To create a way of switching the constraints, you must create a custom channel that will be connected to the weight channels of each constraint. Do this by selecting the UpperBody icon, and choose Modify, Add Attribute. Create a custom channel called **armsSwitch**, with limits from 0 to 1, and a default value of 0. You can use the Connection Editor, expressions, or set driven keys to create a connection between the armsSwitch channel and the constraint channels.

The armsSwitch channel should make one of the constraint channels follow it exactly, and make the other constraint channel follow it in the opposite direction. This has the effect of turning one constraint on, while the other constraint turns off. For example, you can do this with an expression like this:

```
Hands_pointConstraint1.BackFollowW0 =
↳ 1 - UpperBody.armsSwitch;
Hands_pointConstraint1.TorsoFollowW1 =
↳ 0 + UpperBody.armsSwitch;
Hands_orientConstraint1.BackFollowW0 =
↳ 1 - UpperBody.armsSwitch;
Hands_orientConstraint1.TorsoFollowW1 =
↳ 0 + UpperBody.armsSwitch;
```



3.54 Use a custom channel to control the weight channels of two point constraints to make the arms appear to switch parents.



3. When the channels are constrained, try rotating the BackBend circle, and then scrub the armsSwitch channel on the UpperBody icon. You should see your hands smoothly switching between staying still and following the backbone. The hands should still follow the upper body, even when they don't follow the backbone. If you didn't want the hands to follow anything, you would do the same process, only using a locator that is child to the Rig node, rather than the UpperBody node. ■

## ADVANCED ANIMATION CONTROLS

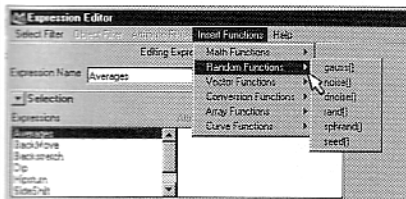
In this section, you add some more complicated controls to your character rig. This process involves using math functions to create automatic torso motions, setting driven keys to create a rolling foot control, creating an advanced backbone with spline IK and clusters, and creating color indicators that warn when a control has been moved too far. Although these controls require more time to set up, they make it easier and faster for you to animate your character.

Some of the expressions used in this section use expression functions. An *expression function* is a predefined command that can be used in an expression instead of explicitly writing the complicated math associated with the command. Some common expression functions used in Maya are abs, trunc, cos, rand, and time. You can view a list of functions in the Expression Editor under the Insert Functions menu (see Figure 3.55). To view a complete list with detailed explanations of each function, look in the Expressions section in the online documentation of the Help menu.

## CONSTRAINING THE TORSO TO THE FEET

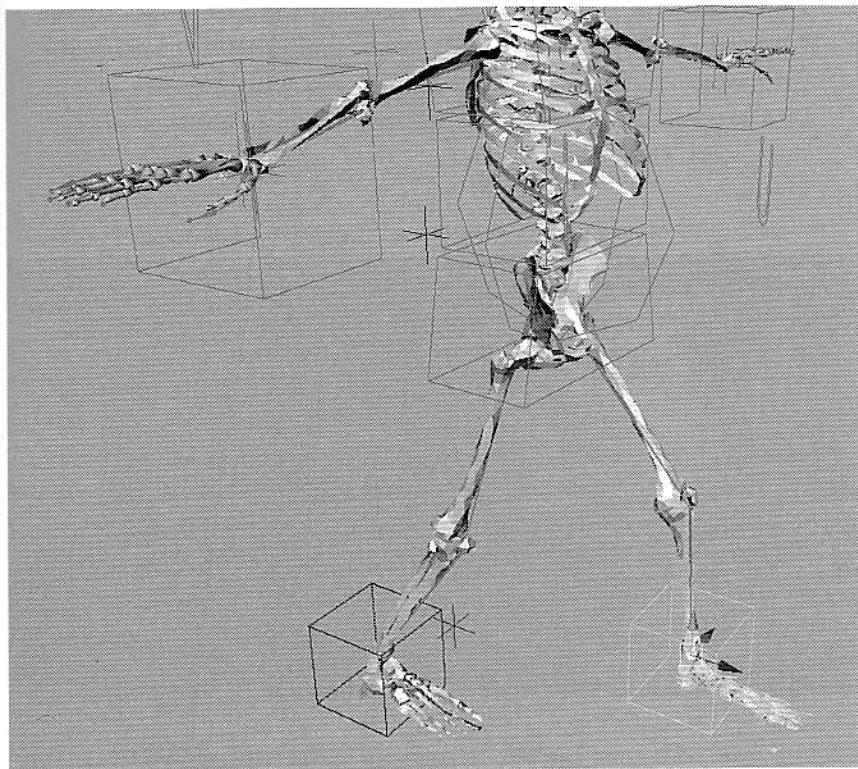
One thing that can be useful for easily moving your character around in a scene is to connect the torso to the feet with an average expression. This always keeps the UpperBody icon between the two leg boxes whenever they move in X and Z. In addition, you create an expression to make the torso automatically dip in Y each time a foot takes a step. The amount of the dip is based on the size of the step (see Figure 3.56).

Keep in mind that although these expressions make certain actions easier to animate, they may not be appropriate for all scenes. Sometimes you will want to disable these expressions and animate these motions manually. This is especially true if your character is going to rotate extensively from the Rig node, such as to perform a 360-degree flip. This is because the Rig and Feet nodes, which are parents of the nodes involved in the



3-55 You can find a list of the advanced expression functions on the menu bar in the Expression Editor.

expressions, cannot be affected by the upper body and leg boxes. The reason for this is that having a child node affect a parent node causes a cycle, which generates an error message from Maya. Because this is undesirable, using the expressions causes the Rig and Feet nodes to get left behind as soon as you begin moving your character around using the average expressions. In turn, this causes the pivot points to be off for any kind of body rotation using these nodes.



**3.56** Add expressions to make your character's torso automatically move when each foot takes a step.

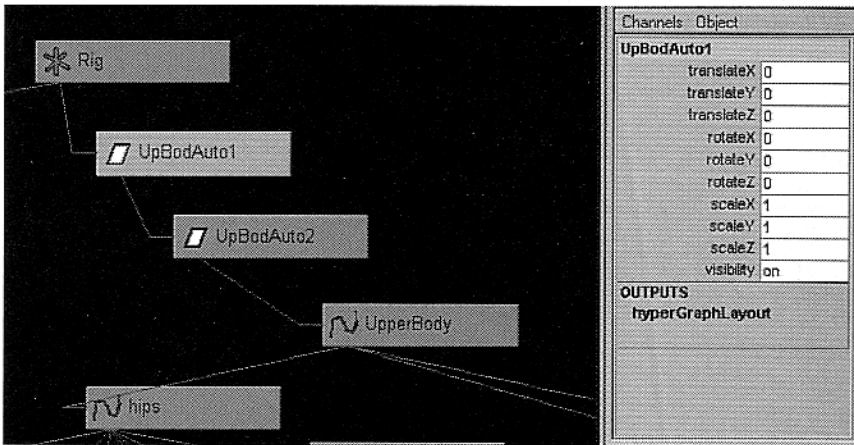
### EXERCISE 3.12

#### CREATING TORSO AVERAGES AND DIP

1. The first step in constraining the torso to automatically move when the legs move is to create a couple of group nodes that are sitting directly on top of, and are the parents of, the UpperBody icon. If you constrain the channels of the UpperBody icon directly with these expressions, you will no longer be able to manually animate the upper body of your character. Instead, you constrain the channels on some parent group nodes that propagate the movement to the upper body through the parent child relationship. You can then manually add keyframes to the UpperBody icon in addition to the automatic movements. The term often used for keeping the channels of a child node open for setting manual keyframes is a *freedom node*.

In the hypergraph view, select the UpperBody icon, and press Ctrl+G twice. Name the top group node **UpBodAuto1**, and the lower group node **UpBodAuto2** (see Figure 3.57). If your UpperBody icon was originally placed right on top of the Rig node, your two group nodes should be sitting right on top of the UpperBody icon. If not, go into insert mode and move accordingly. Because you are going to write expressions, you want the channels of your group nodes to have 0 values in translation and rotation, and 1 in scaling. If they are sitting right on top of the Rig and UpperBody nodes, this should be the case. Keep in mind that in versions of Maya before 4.5, you should not freeze the transforms of nodes that are parented in your skeleton hierarchy. Doing so changes the orientations of your skeleton centers and dislocates any IK handles that are child to these nodes. This problem seems to be fixed in the latest version of Maya, however.

3.57 Making two group nodes as parents of the UpperBody icon creates extra channels that can be constrained in your expressions.



2. After you have some channels to constrain on the UpBodAuto nodes, you can begin writing the expressions for averaging the torso between the leg box positions. One important thing to check before writing the expressions is that the Feet node is directly between the leg boxes. Keep in mind that the information in the channels of the leg boxes is based on the parent node's position. You want, therefore, to make sure that the Feet node is directly between the leg boxes in X, and lined up with the center of the boxes in Z. In addition, before writing the expressions, check the channels of the objects involved for any irregularities in the rotation or scaling channels. They should not, for instance, have Scaling values other than 1, or Rotation values other than 0. The first expression you write should constrain the X translation of the UpBodAuto1 node with an average. Adding a number of values, and dividing the result by the same number of values, creates an average. Use the following syntax:

$$(A + B) / 2$$

Replace the letters *A* and *B* in the preceding expression with each leg box's X translation to average the X translation of the UpBodAuto1 node. Do this on your rig by opening the Expression Editor, and type the following expression:

```
UpBodAuto1.tx = (LtLeg.tx + RtLeg.tx) / 2;
```

Name the expression **Averages**, and click Create. Test the control by translating one of the leg boxes in X. Your character's torso should remain directly in between both leg boxes. To constrain the Z translation in the same way, edit the expression by copying the first expression, and paste it on the next line, changing each X-axis to Z. The second line in your expression should look like this:

```
UpBodAuto1.tz = (LtLeg.tz + RtLeg.tz) / 2;
```

One other average constraint that you may want to use on your character is placed on the Y rotation channel. It is written like this:

```
UpBodAuto1.ry = (LtLeg.ry + RtLeg.ry) / 2;
```

3. One thing you may notice when you test your average constraints is the leg boxes tend to pull the feet out of their sockets when a foot steps. The reason this happens is because the torso doesn't dip when your character steps in X or Z. In a real body, when you take a step, your torso dips. The bigger you step, the more your torso dips. If your torso didn't dip, your feet would come out of your sockets too!

So to make your character's torso automatically dip when your character takes a step, you can just create another expression. In the Expression Editor, click the New Expression button, and type in the following:

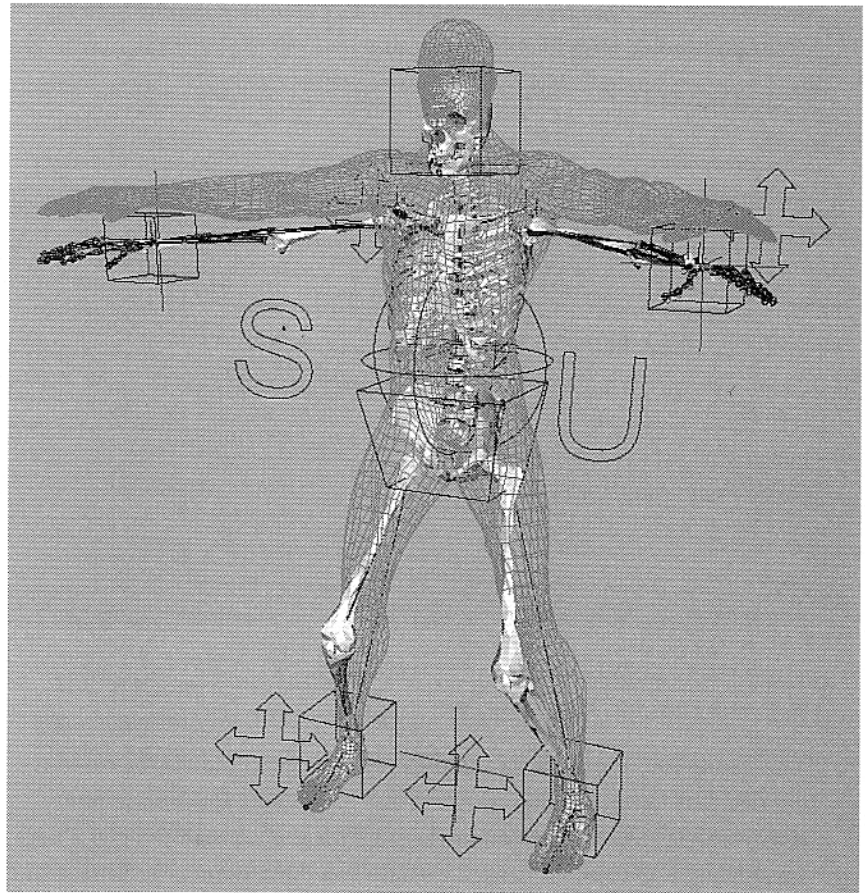
```
UpBodAuto1.ty = 0 - ((abs(LtLeg.tx - RtLeg.tx)
➡+ abs(LtLeg.tz - RtLeg.tz)) / 6);
```

This expression finds how far the distance is between the feet by subtracting one foot from the other in X and Z. The absolute function removes negative signs from the expression, because there is no such thing as a negative distance. Dividing is needed because you don't want your character to dip as far as it steps. Name the expression **Dip**, click the Create button, and move your leg boxes to test the dip.

You probably need to divide the effect by more than 6 to produce less of a dip. In addition, your character has probably moved out of its default position in Y (see Figure 3.58). This is because the expression is activated when the feet move apart, and the feet are already apart in their default pose. You can type in a number greater than 0 as the start number to adjust the character back up in Y. Another way you can make adjustments to your character's

default position is by just translating your freedom nodes. The only channel that will be constrained on UpBodAuto2 is the X translation channel. So you can make any needed adjustments to the Z or Y translation of your character on this node. Also notice that although the expressions force the torso to move with the feet automatically, you can animate the UpperBody icon to compensate or add to the movement as needed. This enables you to animate basic movement of your character quickly, while adding subtle variations to the movement by manually animating the freedom node. ■

- 3.58 Check to see whether the expressions constraining the torso caused your character to move out of its default position.



## WRITING CONDITIONAL EXPRESSIONS

Another kind of advanced expression is a conditional expression, which uses the `if` and `else` commands. These commands enable you to set up a condition of some kind, and do one of two expressions depending on whether the condition evaluates as true or false. This is a good way, therefore, to have your character controls switch between two opposing actions. The automatic controls you create with conditional expressions on your



character simulate the torso shifting weight when a foot is raised, and cause the hips to rotate in Y when a foot steps forward and backward. The general syntax for a conditional expression in Maya is as follows:

```
if (condition) Expression evaluated if the condition is true;
else Expression evaluated if the condition is false;
```

The condition used can be any mathematical statement that evaluates as true or false; for the examples in this book, however, you use greater than (>) or less than (<) statements. If the condition evaluates as true, the If expression runs. If it evaluates as false, on the other hand, the Else expression runs. Keep in mind that you can use the If command without the Else command, but the Else command always requires the If command be used before it. If you don't use an Else command, the expression does nothing if the condition evaluates as false.

### EXERCISE 3.13

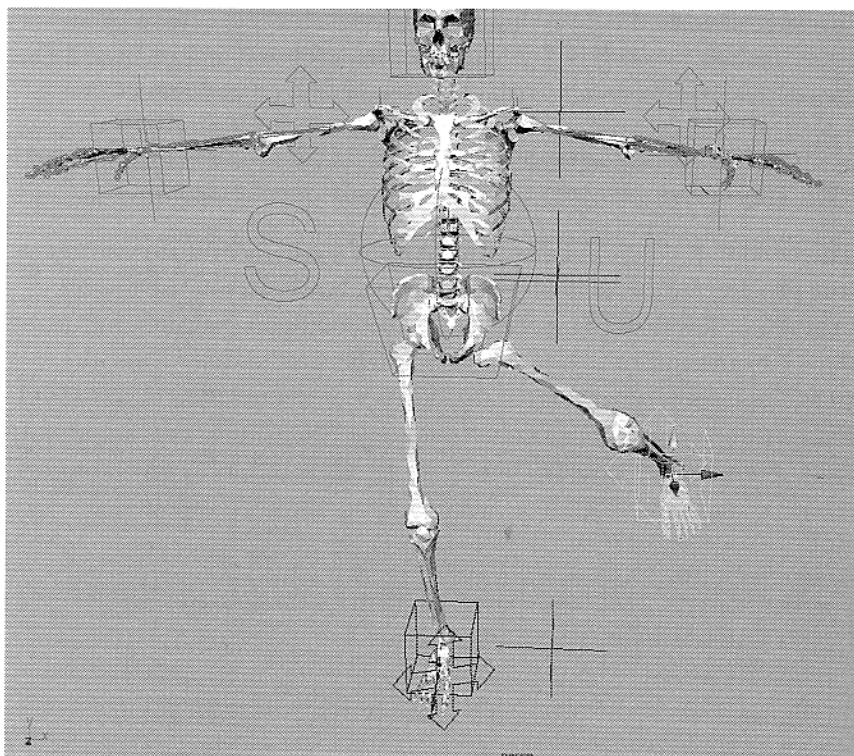
#### USING CONDITIONAL EXPRESSIONS ON YOUR RIG

1. To make the torso automatically shift from side to side when a foot is raised, the translation in X of the UpBodAuto2 node must be constrained with a conditional expression. You must place this expression on the UpBodAuto2 node because the X translation channel on the UpBodAuto1 node is already being used by the average expression. This expression also uses the absolute function, because it involves calculating the distance each foot is raised to set the shifting amount. This amount is then divided at the end of the expression to make the movement subtler. In the Expression Editor, create the conditional expression in the following way:

```
if (LtLeg.ty > RtLeg.ty)
UpBodAuto2.tx = 0 - (abs (LtLeg.ty - RtLeg.ty) / 2);
else
UpBodAuto2.tx = 0 + (abs (LtLeg.ty - RtLeg.ty) / 2);
```

It's okay to separate the expressions onto different lines by pressing the regular Enter key, if this makes them easier to read. Maya still sees only one expression command until it reaches a semicolon. The above expression can be read, "If the left leg rises above the right leg, shift the torso to one side; otherwise, shift it to the other side." Name the expression **Conditionals**, and click Create. If you then raise one of the leg boxes, you should see the torso translate over the opposite leg as if its weight shifted to keep the character balanced (see Figure 3.59). Notice that the only difference between the If and Else expression is the plus and minus signs. If you find your character shifting in the wrong direction, reverse the signs.

- 3-59 Use a conditional expression to make your character shift its torso over the opposite foot when a leg is raised.



2. A similar conditional expression can be done to make the Hips box rotate in Y when a foot steps in Z. To do this, and still be able to manually animate the Hips box, you must make a group node parent to the Hips box. Do this by selecting the Hips box, press Ctrl+G, and name the new node **HipsAuto**. In insert mode, move the pivot of HipsAuto to sit on top of the Hips center. It doesn't matter whether the HipsAuto node has any numbers in its translation channels, but make sure the number in the rotation channels is 0. When finished, copy the previous conditional expression, and paste a second conditional expression in the same window. Then change the second expression to look like the following:

```
if (LtLeg.tz > RtLeg.tz)
HipsAuto.ry = 0 - (abs (LtLeg.tz - RtLeg.tz) * 3);
else
HipsAuto.ry = 0 + (abs (LtLeg.tz - RtLeg.tz) * 3);
```

When finished, click Edit to update the Conditionals expression. If you then move a leg box in Z, you should see the hips rotating in the same direction. Adjust the plus and minus signs if necessary to get the correct rotation. The reason the division should be changed to multiplication in the second expression is because the translation of the legs is affecting the rotation of the hips. For instance, 5 units of translation on the leg boxes makes the hips rotate only 5 degrees, which is not very noticeable. Multiplying the expression makes the hips rotate a noticeable amount. ■

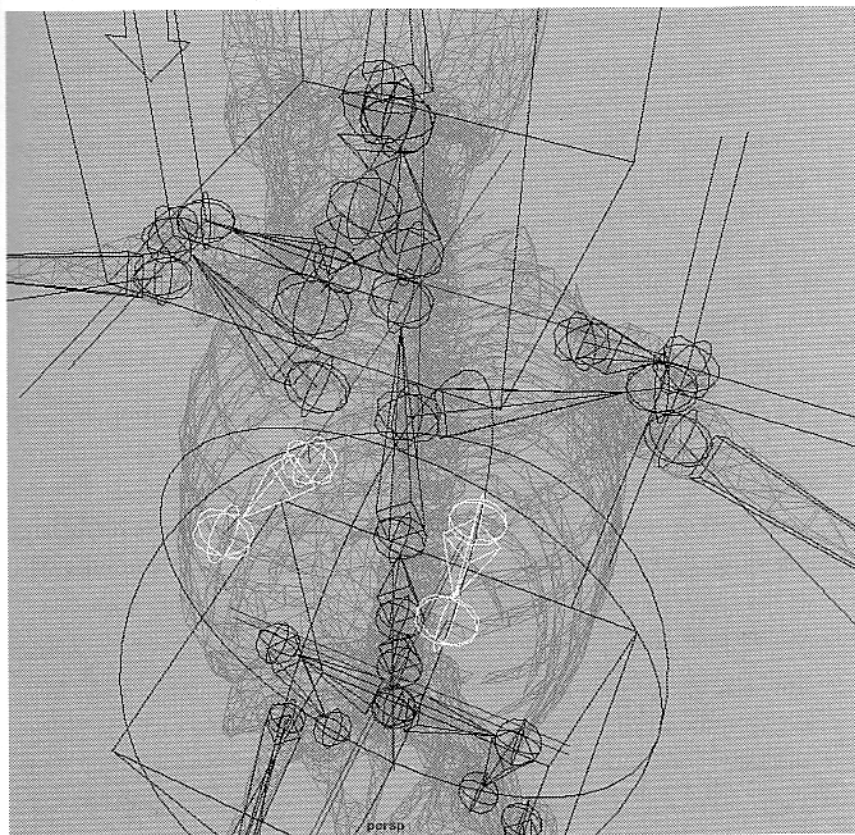
## OTHER ADVANCED MATH FUNCTIONS

A couple of other interesting expressions you can use on your character involve creating a breathing control, an eye-jitter control, and a tail-whip control. The breathing expression involves using a cosine function and a time function that references the timeline. The eye-jitter control involves using a randomize function and custom channels. Finally, the whipping tail expression also involves referencing the timeline, and introduces the use of MEL commands in an expression to create a delay effect.

### EXERCISE 3.14

#### USING FUNCTIONS TO CREATE ADDITIONAL CONTROLS

1. To create some breathing controls, in the top view draw a two-joint FK skeleton in the left chest area of your character. The skeleton should start inside the chest and end at the surface of the chest (see Figure 3.60). Translate and rotate the root joint as needed to place the skeleton correctly in the chest area. Name the joints **LtBreathRoot** and **LtBreathEnd**.



**3.60** Draw two skeletons in the chest area of your character to use as breathing controls.

Mirror the skeleton to the right side, and name the right skeleton appropriately. Select both root joints and group twice, naming the group nodes **BreathPad1** and **BreathPad2**. For now, make **BreathPad1** child to **Back3**, so the breath joints move with your character's backbone. You use an expression to control the X scaling of the skeletons. This causes the front of the chest to move forward and backward.

On the **UpperBody** icon, create two custom channels named **breathSpeed** and **breathSize**. Set the minimum and maximum limits of both channels to .1 and 10, with a default of 1. In the previous expressions shown in this chapter, you divided and multiplied your expressions by specific numeric values to control the amount of movement produced. Doing this sets a static amount of control that cannot change during your animation. In this expression, you divide by a channel that can be animated to change over time. This enables you to vary the movement produced by the expression over the course of your animation. Type the following lines in the Expression Editor to create the breathing effect:

```
LtBreathRoot.sx = 1 + (cos (time / UpperBody.breathSpeed)
➡/ UpperBody.breathSize);
RtBreathRoot.sx = 1 + (cos (time / UpperBody.breathSpeed)
➡/ UpperBody.breathSize);
```

Name the expression **Breathing**, and click **Create**. This expression uses the cosine function in conjunction with the time function. A cosine function creates a wave, which is useful for making the X scaling of the breath joints go up and down. Using the time function makes the scaling happen automatically whenever the timeline is played. Scrub or play the timeline to see the breath joints scaling continuously. Change the breath speed and size channels to change the rate and scale of the breathing. Try setting the **UpperBody.breathSpeed** channel to .5, for instance, and the **UpperBody.breathSize** channel to 8.

You may want to create an additional breath skeleton in the stomach cavity to simulate deep breathing. This skeleton should be drawn from the middle of the abdomen to the belly button, and named **LowBreathRoot**. Use the same kind of expression as used on the chest skeletons to create the breathing effect. Bear in mind that these skeletons will later be bound to the skin of the torso to make it deform in a subtle manner as the joints scale continuously in X.

2. To create an eye-jitter control, you must first parent two group nodes in between the **EyesLook** circle and the locators constraining each eye. Select the **LtEyeLook** locator, and press **Ctrl+G** twice. Name the top group node

**EyeJitterPad**, and bottom group node **EyeJitter**. Then make the **RtEyeLook** locator child to **EyeJitter**. The translation channels on the **EyeJitter** node should all be at 0, and they should be sitting directly on top of the center of the **EyeControl** circle. Freeze the nodes if necessary to set the translation channels to 0.

After you have created the two new nodes, make three custom channels that will control the jitter. On the **EyeJitter** node, create a **jitterMin** and **jitterMax** channel. The **jitterMin** channel should go from -10 to 0, with a default setting of 0. The **jitterMax** channel should go from 0 to 10, with a default setting of 0. Then on the **EyesLook** icon, create a **jitterControl** channel that goes from 0 to 10, with a default setting of 0. After you have made all the channels, type the following expression into the Expression Editor:

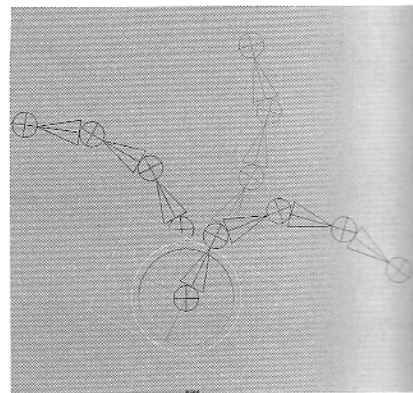
```
EyeJitter.jitterMax = 0 + EyesLook.jitterControl;
EyeJitter.jitterMin = 0 - EyesLook.jitterControl;

EyeJitter.tx = 0 + rand (EyeJitter.jitterMin,
➡EyeJitter.jitterMax) / 8;
EyeJitter.ty = 0 + rand (EyeJitter.jitterMin,
➡EyeJitter.jitterMax) / 8;
```

Name the expression **JitteryEyes**, and click the Create button. The way this expression works is that the **jitterControl** channel controls both the **jitterMin** and **jitterMax** channels. When the **jitterControl** channel is set to 5, for instance, the **jitterMin** channel is set to -5, and the **jitterMax** channel is set to 5. These two channels are then used to define the minimum and maximum amounts that the randomize function requires, which is then used to control the translation channels of the **EyeJitter** node. Try increasing the **jitterControl** channel and notice that the eyes start to shake. The shaking continues as long as the timeline is played.

3. The next expression creates a delay on the rotations of the joints in a tail to create a whipping effect (see Figure 3.61). Because Maya doesn't have a delay function, you have to use some MEL scripting to make the expression work.

Draw an FK skeleton for a tail in the top view, starting in the middle of the hips and going straight back behind your character. For this example, the tail has five joints. Name the joints **TailRoot**, **Tail2**, **Tail3**, **Tail4**, and **TailEnd**. Point-constrain the root joint of the tail skeleton to the Hips box. When the tail skeleton is placed so that it starts in the Hips box and goes straight back behind the character, type the following lines in the Expression Editor:



**3.61** Use an expression that creates a delay on the rotation of joints to make a whipping effect on a tail.



```

int $time = 'currentTime -query';
int $delay10 = 'getAttr -time ($time - 10) Hips.ry';
int $delay20 = 'getAttr -time ($time - 20) Hips.ry';
int $delay30 = 'getAttr -time ($time - 30) Hips.ry';
int $delay40 = 'getAttr -time ($time - 40) Hips.ry';
TailRoot.rotateZ = 0 + Hips.rotateY;
Tail2.rotateZ = 0 + $delay10;
Tail3.rotateZ = 0 + $delay20;
Tail4.rotateZ = 0 + $delay30;
TailEnd.rotateZ = 0 + $delay40;

```

4. Name the expression **TailWhip**, and click Create. This expression uses MEL scripting to create a delay effect on the tail joints. Because this expression references the timeline, it is necessary to set keyframes on the Y rotation channel of the Hips box to see the whipping effect. Rotate the Hips box in Y, and right-click the Y channel in the channels box to set keys at several frames. If you then scrub or play the timeline, you should see a whipping effect on the tail.

This expression uses two MEL commands: The `currentTime` command finds out what frame the timeline is on; the `getAttr` command enables you to get the value for the hips' rotation in Y at a particular frame. Both of the results for these commands are being placed in the variables `$time`, `$delay10`, `$delay20`, and so on. These variables are then used in the expressions to constrain each tail joint to rotate with an increasing delay. This creates the whipping effect on the tail. Don't worry if you don't entirely understand the MEL scripting used in this expression. You learn more detail about creating variables and using MEL commands in Chapter 6, "Scripting MEL Character Controls." ■

## CREATING A ROLLING FOOT

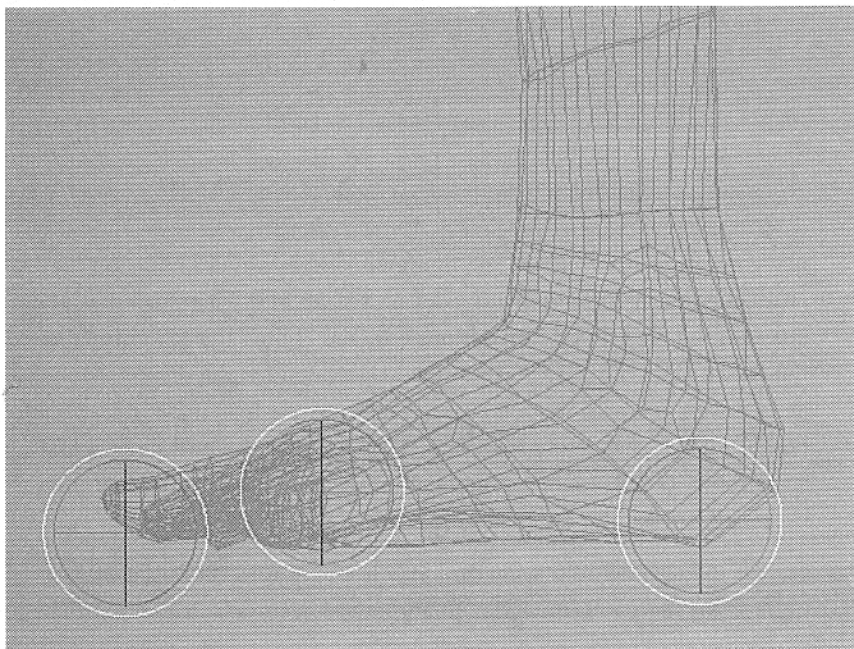
Next you create a more complicated foot hierarchy and set driven keys to drive a rolling foot control. The new foot setup involves creating group nodes that are used as pivot points for your current foot skeletons. A custom roll channel is created on each leg box to drive the pivot points so that each foot can roll forward and backward. When a foot rolls backward, it should pivot back on the heel. When it rolls forward, it should pivot up on the ball of the foot and then up onto the toe.

## EXERCISE 3.15

## ADDING FOOT CONTROLS

1. Open a hypergraph view, and zoom in on the feet section of your character rig hierarchy. Hide the visibility of the RtLeg box, so the right foot isn't in the way when you are working on the left foot. Disconnect the left-leg IK handle and polygon reference bones from your hierarchy by selecting them and pressing Shift+P. Be careful not to disconnect or change the parenting on the LtLeg box, or your torso may be affected through the expression connections. Then make sure your left-foot root joint is child to the ankle box. To create the new pivot points, select the foot root joint, and group three times. In the hypergraph, this creates three group nodes in the hierarchy that are between the left-leg box and the left-foot root joint.

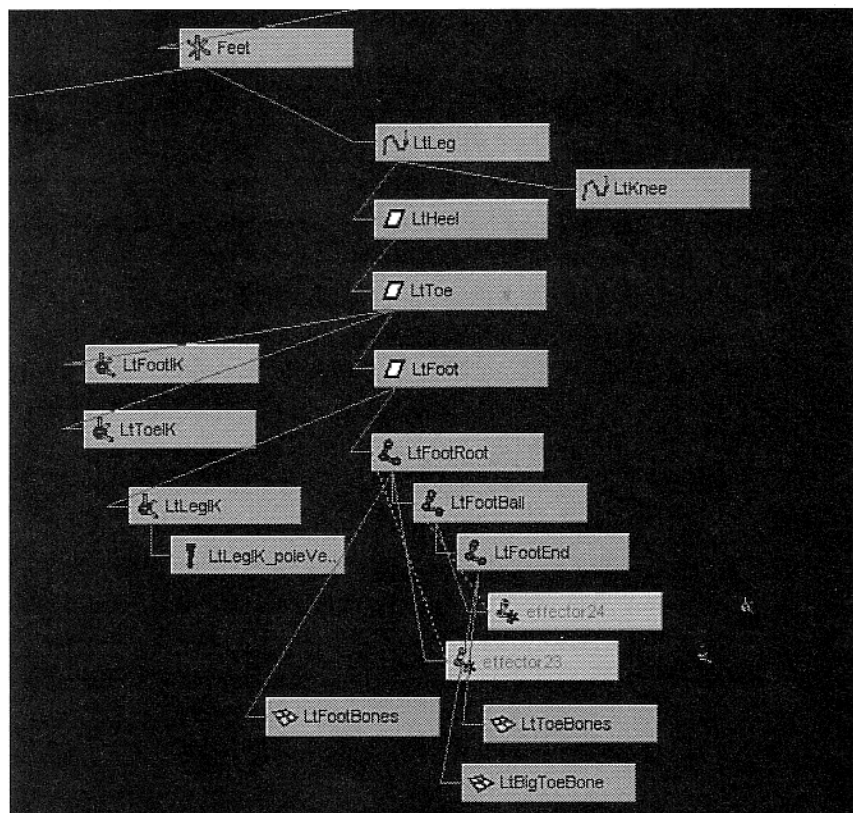
Name the three new group nodes from top to bottom as **LtHeel**, **LtToe**, and **LtFoot**. Press the Insert key, and move each group node's pivot to the appropriate place on the foot. The heel pivot should be where the heel touches the ground, the toe pivot should be where the toe touches the ground, and the foot pivot should be at the ball of the foot (see Figure 3.62). Then constrain the foot joints with IK twice. Using the IK handle tool, create IK from LtFootRoot to LtFootBall, and then create another IK from LtFootBall to LtFootEnd. Name the two new IK handles **LtFootIK** and **LtToeIK**. If your joints moved a little when you added the IK handles, translate the handles to put them back in place.



3.62 Create three new group nodes for each foot to use as the pivot points in a foot roll.

To finish the left-foot setup, parent the IK handles. Make the LtLegIK handle child to LtFoot, and make both the LtFootIK and LtToeIK child to LtToe. Then reparent the polygon foot bones under the foot root joint, and the polygon toe bones under the LtFootBall joint (see Figure 3.63). When everything is parented, test the three new pivot points by rotating them in X.

3.63 Make all the foot skeletons and IK handles child to the new group nodes.



2. To create a single control that drives all three of the foot pivots, create a custom channel on the left-leg box named **roll**. (Channel names are normally not capitalized.) Make the minimum and maximum limits on the channel go from -5 to 10, with a default setting of 0. Then load the LtLeg box into the Set Driven Key options box, and choose the roll channel as the driver. Load the X rotation of the LtHeel node as the driven channel. Set a driven key with the roll channel at 0, and the heel node in its default position. Then change the roll channel to -5, rotate the heel node backward about 25 degrees in X, and set another key. This should create the backward rotation on the heel part of the foot roll.

Set the roll channel to 0 again, and load in the X rotation of the LtFoot node as the driven channel. Key them both in their default positions. Then

change the roll channel to 5, rotate the LtFoot node forward in X about 30 degrees, and set a driven key. This should make the foot roll up on the ball of the foot (see Figure 3.64). Finally, set the roll channel to 5, load in the X translation of the LtToe node, and set a driven key for its default position. It is important to do this at 5 on the roll channel, so the foot doesn't start at 0 to roll forward onto the toe. Then change the roll channel to 10, and rotate the toe node forward about 30 degrees in X, and set another key. Also at 10, load the LtFoot node in again, and rotate it backward about 20 degrees, and set a final key. This causes the foot to uncompress at the ball of the foot when it rolls up on the toe.



**3.64** Set driven keys on the group nodes to make the foot roll from the heel up onto the ball of the foot, and then onto the toe.

3. When all the driven keys are set, test the roll channel. You should see your character's foot rolling backward when the roll channel goes in a negative direction and roll forward when the roll channel goes in a positive direction. One thing you should add to this setup is a node for manually animating the toes' rotation. Do this by selecting the toe IK, and press Ctrl+G. Name the group node LtToeRot, and move its pivot point to the ball of the foot. Then create a custom **toeRotate** channel on the left-leg box, and connect it with driven keys to the X rotation of the LtToeRot node. Keep in mind that you can still manually rotate your whole foot by just rotating the leg box. ■

## BUILDING A FLEXIBLE BACKBONE

Although the current FK backbone skeleton in your rig can bend in all directions, it has some limitations. It lacks fine controls for posing the backbone in more arched positions. It is also not easy to shift areas of the back—for instance, to stick your chest out—without causing problems to the joint centers. To upgrade your character's backbone controls to be more flexible, you create a new backbone skeleton that contains spline IK. This new backbone structure is added to your original backbone skeleton, so that you have two levels of backbone controls. In addition to this, you create a stretching quality to the backbone that can be used for both subtle or cartoon effects.

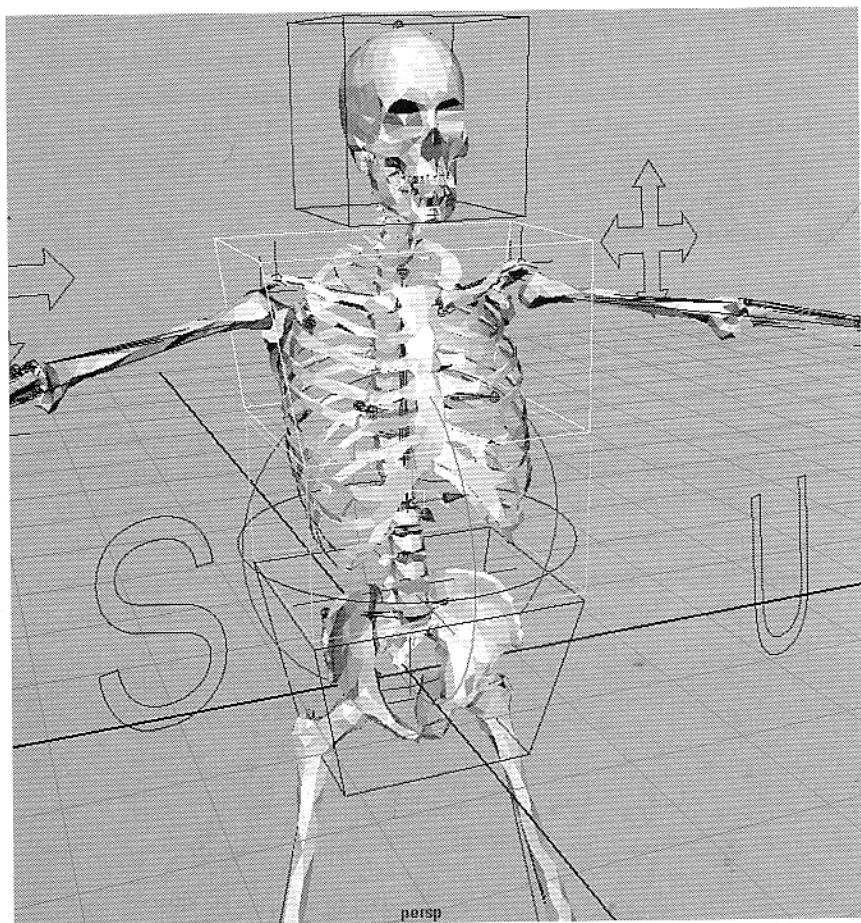
### EXERCISE 3.16

#### CREATING A SPLINE IK-BASED BACKBONE

1. Begin by creating two new box icons named **MidBack** and **UpperBack**. Move and scale the MidBack box so it is right above the Hips box and covers the abdomen area of your character. Move and scale the UpperBack box so that it surrounds the upper chest and shoulders area of your character (see Figure 3.65). You can also go into component mode and scale vertices to create a better shape. When in place, freeze the transforms on both boxes.
2. To create a more flexible backbone, you must draw a new backbone skeleton, with joints that follow the curve of the back. This doesn't mean, however, that you have to create as many joints as exist in a real backbone. You can create half the number of joints in a real backbone and still get enough flexibility. The fewer joints you have, the easier it is for you to add controls for each joint. Before drawing any joints, however, hide the BackPad1 node and the thorax polygon bone. This should give you a clear view of your polygon backbones in the side view. Also, make sure that IK is turned off in the Joint Tool options box before drawing joints.

Draw the new backbone skeleton starting where the waist bends up to where the neck starts to bend. On real skeleton vertebrae, this is from between L2 and L3 to between T1 and C7. Use your polygon reference bones as a guide, drawing straight up the middle of the bones, placing each joint between every second vertebrae. In addition, do not draw a joint for every backbone, but instead draw one for every other backbone. This should create about eight backbone joints. When finished, name the joints **B1** through **B8**.

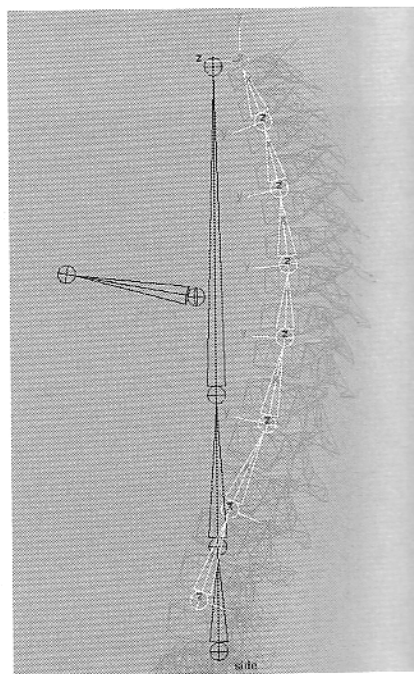





- 3.65 Place a couple of new curve boxes to control the middle and upper areas of the new backbone skeletons.

It is important for all the joint centers to be oriented correctly on your backbone skeleton. Show the centers on your joints, and check to see whether any are flipped 180 degrees in X (see Figure 3.66). This happens when you change directions when drawing a skeleton. If this is the case, use the same method described earlier in this chapter to rotate the joints to the correct orientation. This involves using the question mark (?) symbol in component mode to select the center and clicking the shelf button you created for rotating the center 180 degrees in X (the shelf button should execute the MEL code `rotate -r -os 180 0 0`.)

When your joint centers are all oriented in the same way, finish your new backbone skeleton by parenting all the appropriate polygon vertebrae to the new Maya joints. Each backbone joint should be parent to two polygon vertebrae. The polygon thorax bone should be made child to B5 or B6. You can adjust this later.




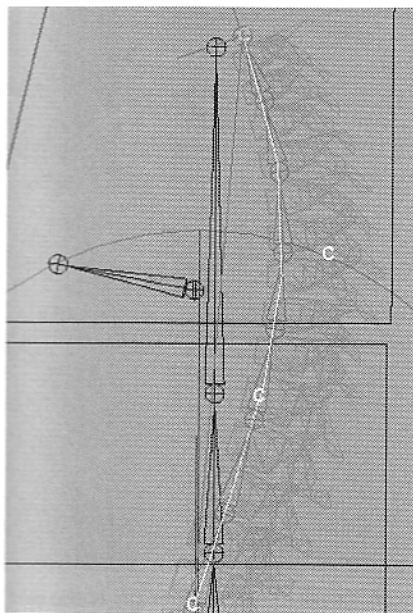
- 3.66 Check to see whether any of the centers on your new backbone joints are flipped around 180 degrees in X.

3. Spline IK uses the shape of a curve to control the orientation of joints in a skeleton. Assign the spline IK to your character's new backbone by choosing Skeleton, IK Spline Handle Tool . Set the spline IK options to the following:

Root on Curve: On	Number of Spans: 1
Auto Parent Curve: Off	Root Twist Mode: Off
Auto Create Curve: On	Twist Type: Linear
Auto Simplify Curve: On	

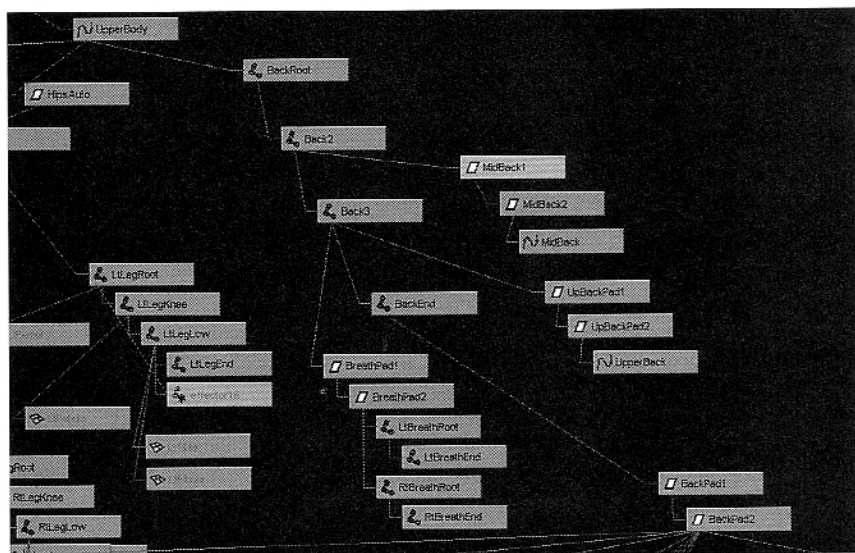
Using these options creates a simple curve that follows the shape of the backbone skeleton. It also creates an IK handle that contains a twist channel used to rotate the joints in X with a linear falloff, so the top twists more than the bottom. Close the options box, and click your backbone skeleton's root joint, and then click its end joint. In the hypergraph view, notice that a new curve and IK handle are created. Spline IK differs from regular IK in that you do not animate the transforms of the IK handle. Instead, select the new curve and switch to component mode. The curve should have four points, and moving the points causes the backbone skeleton to flex. Name the new curve and IK handle **BackCurve** and **BackIK**.

4. To make controlling the points on the backbone curve easier, create cluster handles for each point. Select the lowest point on the curve, and choose Deform, Create Cluster . Make sure the options are at their default settings, with Relative turned off, and click Create (see Figure 3.67). Repeat the process for each point, and then name the clusters from lowest to highest **Bc1** through **Bc4**.
5. Repeat the same process for the hips of your character. Starting around the L1 vertebrae, in the side view draw a four-joint skeleton to the bottom of the pelvis. Because there are not many hipbones, you can draw a joint for each bone. When drawing, make sure you do not connect your hips skeleton to your backbone skeleton. To ensure this doesn't happen, click out to the side of the character, and drag the root joint of the hips over the backbone root joint. Name the hips joints from top to bottom **H1** through **H4**. Once created, assign spline IK from the root to the end of the hips skeleton. Use the same settings as used previously. This action results in a curve with four points, and an IK handle, which you should name **HipsCurve** and **HipsIK** respectively. Create four cluster handles on the hips curve, and name them from top to bottom **Hc1** through **Hc4**. Parent the polygon lower vertebrae and pelvis bone to the appropriate Maya joints.



3.67 Create cluster handles on each point in your backbone curve.

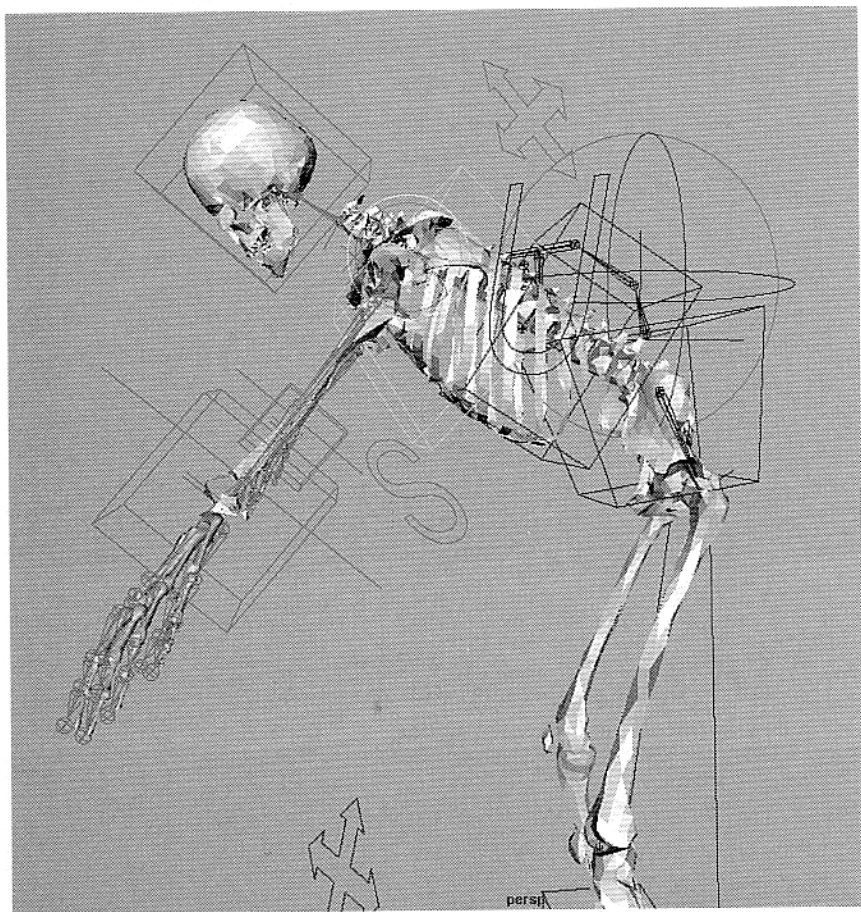
- 3.68** Parent the `MidBack` and `UpperBack` boxes under the original backbone joints.



To make sure everything on the limbs and head moves with the new backbone, make BackPad1 child to the last joint in your new backbone. Depending on how many new backbone joints you made, this will probably be B8. If there are any other children of the original backbones, such as the breathing skeletons, be sure to parent them under the appropriate new backbone joints. In addition, parent the neck clusters under the Head and UpperBack boxes. Make Nc1 and Nc2 child of the UpperBack box, and



three main sections of the backbone (see Figure 3.70). The finest level of controls is reached by translating individual clusters. You can animate the clusters manually, or set driven keys to control them with custom or transform channels on the boxes. This level of detail may not always be necessary, but is available when needed. On the hips, you do not have the FK level of control, but instead just the box and cluster levels.

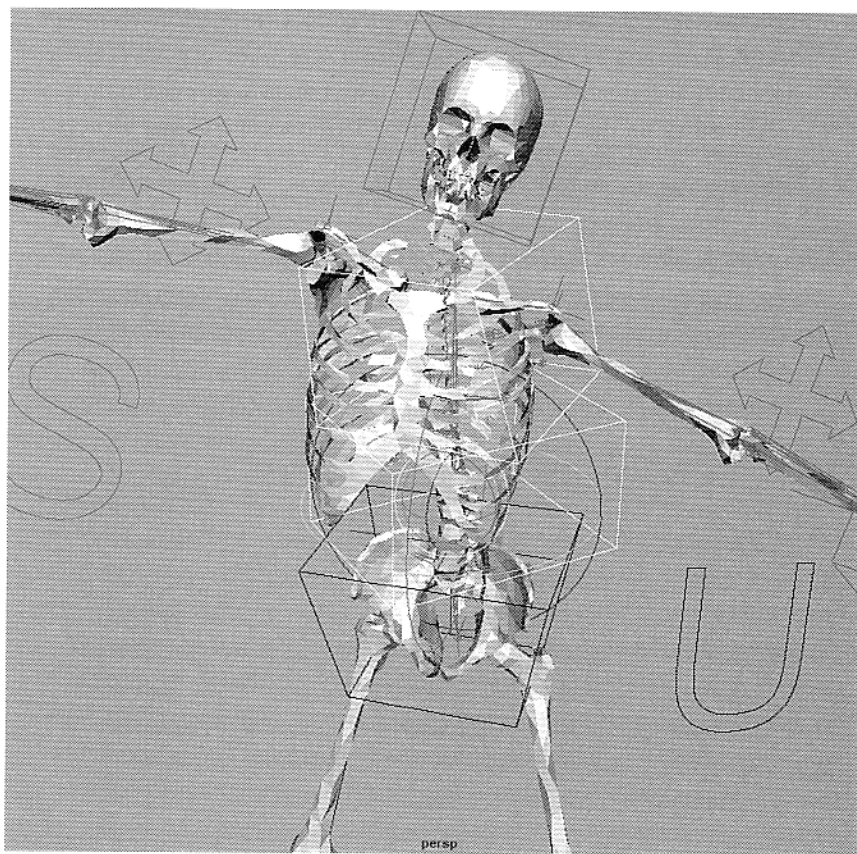


**3.70** Rotate your circle controls to create a basic bend in the backbone, and then adjust the boxes to create more specific poses, such as an arched back.

9. When testing your back controls, you may have noticed a problem. The controls that bend the character to the side, forward, and backward should all work fine, but the controls that twist the back in Y are no longer working right (see Figure 3.71). The character doesn't actually twist, but just wobbles slightly. The reason this is the case is that the spline IK is constraining the twist of the backbone, neck, and hip skeletons. After you put spline IK on a skeleton, you must control the twisting of the skeleton through the twist channel on the spline IK handle.



- 3.71 When using spline IK, your character will no longer twist in Y by just rotating the controls, as seen here. Instead, you must use a separate twist channel.



You can use an expression or set driven keys to control the twist channel on the spline IK handles. To make the rotation in Y of the UpperBack box control the BackIK.twist channel with an expression, for example, type the following line in the Expression Editor:

```
BackIK.twist = 0 + UpperBack.ry;
```

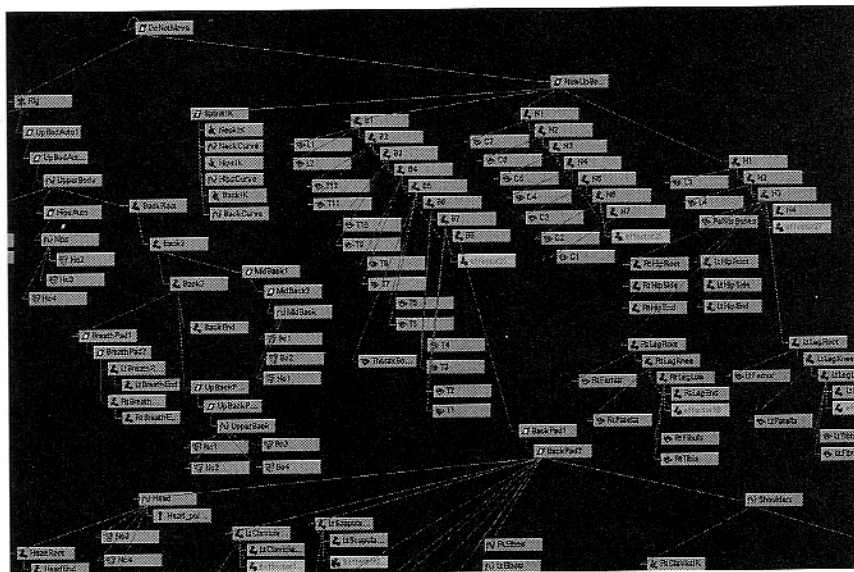
Name the expression **Twists**, and click Create. After creating the expression, try rotating the UpperBack box in Y. If the back is rotating in the wrong direction, change the plus sign to a minus sign. Although you could have made the green BackTwist circle constrain the twist channel, it makes more sense to do it on the level of the boxes that affect the spline IK. Because the BackTwist circle is no longer needed, just delete it. In addition, the Y rotation of the Hips box should control the HipsIK.twist channel, and the Y rotation of the Head box should control the NeckIK.twist channel. These expressions will look something like this:

```
HipsIK.twist = 0 - Hips.ry;
NeckIK.twist = 0 + Head.ry;
```

10. One other channel on the spline IK that you must set controls for is the roll channel. If you try rotating the UpperBody node in Y, you will see a problem similar to the previous twisting problem. The difference between the twist and roll channels is the twist channel has a falloff when rotating the joints, whereas the roll channel rotates all the joints evenly. To fix this, set driven keys on the roll channels of the spline IK for the back, hips, and neck.

The reason you should use driven keys is because you may want to create more than one driver. Load the Y rotation channel of the UpperBody icon as the driver, and use the Shift key to load all the spline IK handles as driven. Again, use the Shift key to select all the spline IK handles in the Set Driven Key options box, and choose the roll channel as the driven channel. Rotate the UpperBody icon in Y, adjust the roll channels as needed to make the skeletons line up with the boxes, and set driven keys. If you are using the Y rotation average expression on the UpBodAuto1 node, load that node as a second driver, and set driven keys in a similar manner.

11. Your character's backbone should be working correctly at this point. There are still, however, several stray nodes that are not parented into your rig hierarchy. Select the root joints for the new backbone, neck, and hips, and press Ctrl+G. Name the parent group node **NewUpBody**. Also select all the spline IK curves and IK handles, create a group node parent named **SplineIK**, and make it child to the NewUpBody node. Finally, create a group node parent of the Rig node named **DoNotMove**, and make NewUpBody and SplineIK child to it (see Figure 3.72). The spline IK curves and handles should never be transformed or animated. Because they are being controlled by constraints, if you move them directly they force a



3.72 Finish your rig hierarchy by parenting all stray nodes. Make sure you parent the spline IK nodes and curves under a node that will never be animated.

double transform on the skeletons. Because this will cause the pieces of your rig to separate, avoid it! There are times, however, when you will need to parent spline IK-controlled skeletons under a control object. This sometimes is required when using spline IK on appendages like ears or a tail. If your character has spline IK on some large rabbit ears, for example, you might see the skeletons twist when the Head box is rotated. This occurs independently of the roll and twist values in the spline IK handle. Parenting the spline IK constrained skeletons under the Head box will fix this problem.

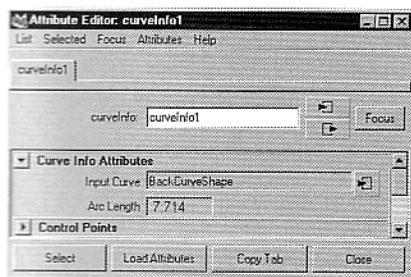
12. To make the backbone be able to stretch, you must get information on the length of the back curve. Do this by selecting the back curve, and type in the following MEL command on the command line or in the Script Editor:

```
arclen -ch 1;
```

Press the Enter key on the numeric keyboard to run the MEL code. Then, with the back curve still selected, graph the node in the hypergraph by clicking the icon for Input and Output Connections. You should see a node called curveInfo1. Right-click this node to open the Attribute Editor, and make a note of the value in the arcLength attribute (see Figure 3.73). This number is the default length of the back curve and is based on the size of your character.

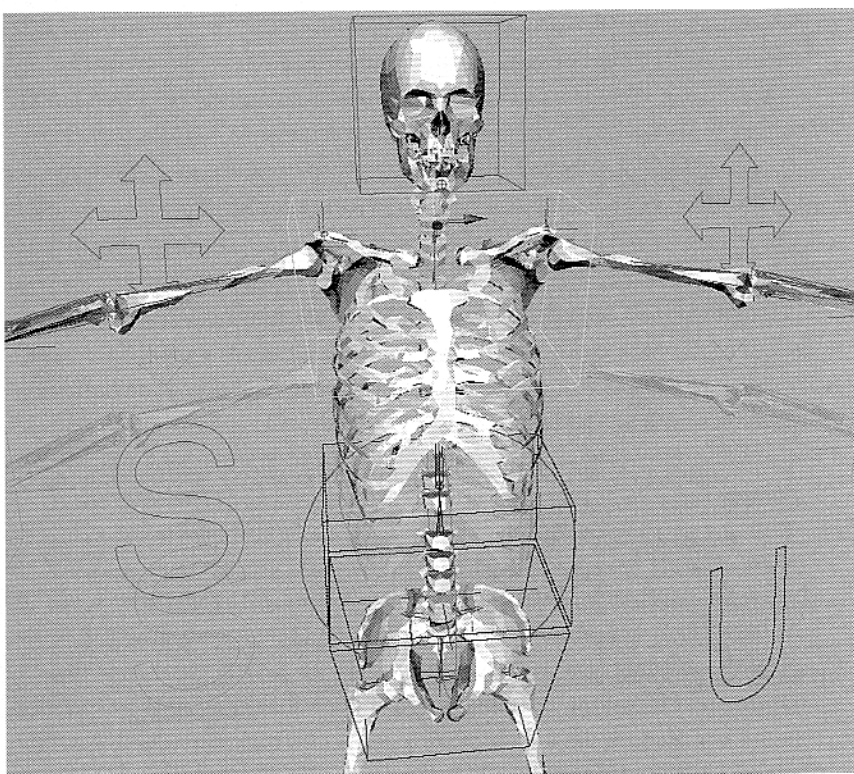
To make the backbone stretch, you will use expressions to make the joints translate in their local X-axis according to the length of the back curve. Translating all the joints except the root joint according to their local X-axis causes the skeleton to get larger or smaller. In the following expressions, you divide the current curveInfo1.arcLength channel value by the original length, as noted previously in the Attribute Editor. When the expression is evaluated, if the lengths are the same, the result is 1, which causes no stretching. If the curve is longer or shorter, however, a result other than 1 is generated to make the backbone stretch. In the expression, the results generated by the curve length are multiplied by each joint's start value. It is important that this number reflect the default X Translation value of each joint. You must, therefore, select each joint to find out which numbers to use in your expression. Based on your character's backbone, your expressions should look something like this:

```
B2.tx = .954 * (curveInfo1.arcLength / 7.714);
B3.tx = .923 * (curveInfo1.arcLength / 7.714);
B4.tx = .9   * (curveInfo1.arcLength / 7.714);
B5.tx = .87  * (curveInfo1.arcLength / 7.714);
B6.tx = .854 * (curveInfo1.arcLength / 7.714);
B7.tx = .8   * (curveInfo1.arcLength / 7.714);
B8.tx = .792 * (curveInfo1.arcLength / 7.714);
```



3.73 Make a note of the arcLength value of the curveInfo1 node in the Attribute Editor.

Name this expression **Backstretch** and click Create. Because this is meant to have the effect of stretching the backbone joints, it is not necessary to write a line for the root joint. Make sure you use the default X Translation value for each joint as its start number. Look in the channels to get the correct value. After creating the expression, try translating the UpperBack box in Y to see the stretch effect on the backbone skeleton (see Figure 3.74). ■



**3.74** Create an expression that makes the backbone stretch when the backbone boxes are moved.


## CREATING WARNING SIGNS

Many controls in your character rig have defined limits. These have been done through setting actual transform limits in the Attribute Editor, or by how you set the minimum and maximum settings on the custom channels of your control icons. Sometimes, however, it is better not to limit the controls, but instead make some kind of warning that tells animators that they have moved a control past its normal settings. Then the animator has the choice of how far to manipulate a control. Occasionally, the animator may want to go beyond the normal limits of a control to create dramatic effects. Such a warning can be done in a variety of ways, from making some text or symbol appear, to making your polygon reference

bones change color. In the next exercise, you place some of these warning signs on your character when the backbone is stretched too far, or when an arm or leg is hyperextended.

### EXERCISE 3.17

#### MAKING TEXT AND COLOR WARNINGS

1. A warning can be any kind of indication to the animator that a control has been moved too far. For instance, you could create a text message that is revealed whenever the backbone is stretched or compressed too far. Choose **Create, Text**  to create something to use as a warning message, such as "Whoa buddy, you're going a bit too far!" Make sure you use curves, and scale or translate the text as needed. Place it beside your character where it will be easy to see. In the hypergraph, name the top group node for the text **WarningText1**. Make the **WarningText1** node child to the **UpperBack** box. The following expression sets the visibility of the warning text based on the length of the back curve. The expression contains an **Or** symbol, known as a double pipe (**|**), which can be found on the Backslash key. Your expression will look similar to the following:

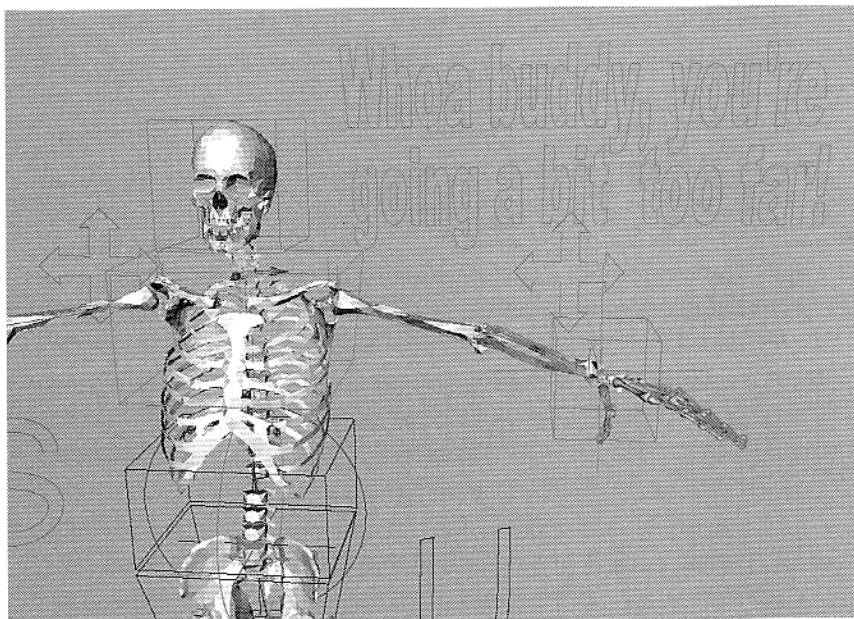
```
if (curveInfo1.arcLength > 8 || curveInfo1.arcLength < 7)
WarningText1.visibility = 1;
else WarningText1.visibility = 0;
```

Name the expression **Warnings**, and click **Create**. The expression reads: "If the back curve gets stretched beyond 8 or gets compressed smaller than 7, reveal the warning text by setting its visibility to 1. Otherwise, hide the warning text by setting its visibility to 0." The values used in the expression should be based on the length of your character's back curve, and may differ from the example. When finished, test the warning by translating the **UpperBack** box in **Y** (see Figure 3.75). You should see the text warning appear whenever the back is stretched beyond the lengths specified.

You can use the same kind of expression to make your polygon reference bones change color when the back curve is stretched too far. Just constrain the incandescence channels on the **Bonecolor** material node rather than the text visibility. Open a hypershade view to select the **Bonecolor** material, and notice three incandescence channels are not being used. To have your bones turn red when the back is stretched, create a similar expression to the following:

```
if (curveInfo1.arcLength > 8 || curveInfo1.arcLength < 7)
Bonecolor.incandescenceR = 1;
else Bonecolor.incandescenceR = 0;
```





**3.75** Create a control that shows a text warning whenever the backbone is stretched too far.

2. Another warning you can create is for when an arm or leg box is translated too far. To do this effectively, you need to calculate the distance between the shoulder and the hand. Keep in mind that the shoulder is the main pivot for the arm, while the hand moves around it in an arc. You want to have Maya calculate when the hand reaches the limit of that arc. To do this, you use the Distance tool by choosing **Create, Measure Tools, Distance Tool**. Then in the front view, click once above your character's left shoulder, and once above your character's left-arm box. This creates two locators that have a distance connection. Name the shoulder locator **Loc1**, and the wrist locator **Loc2**.

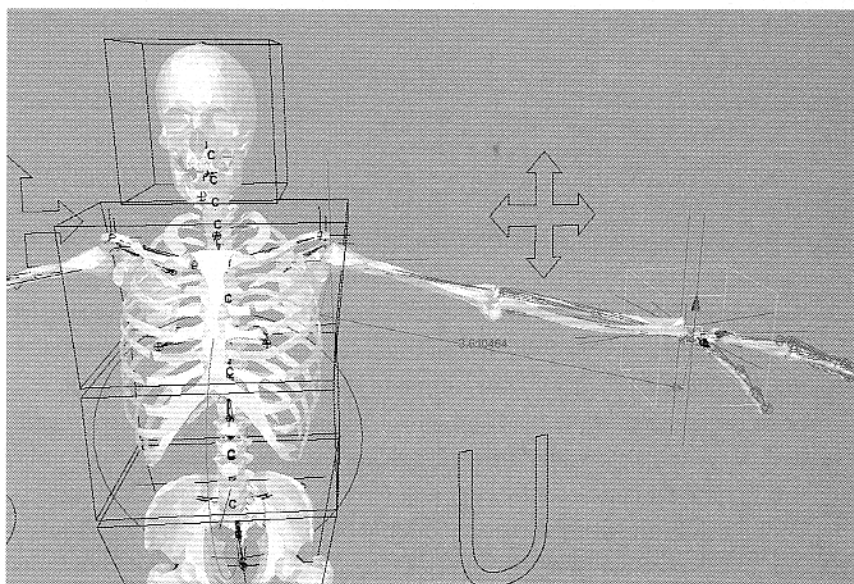
To calculate the distance on the left arm, you need to point-constrain the locators to the arm root and arm box. Select **Loc1** and **Loc2** and group them, naming the parent node **LimbDistances**. When created, point-constrain **Loc1** to the **LtArmRoot** joint, and point-constrain **Loc2** to the **LtArm** box.

Also notice that a **distanceDimension** node has been created in the hypergraph. Right-click the node to open it in the Attribute Editor, and note the distance listed under the **Shape** tab. This distance can be used to create a warning for when the left arm is translated too far. You can write such a warning by creating an expression similar to the following:

```
if (distanceDimensionShape1.distance > 9.2)
Bonecolor.incandescenceG = 1;
else Bonecolor.incandescenceG = 0;
```

This expression makes the Bonecolor turn green whenever the left arm is translated too far (see Figure 3.76). The best way to create warning colors on your character is to create multiple copies of the Bonecolor material and place them on specific areas of your polygon reference bones. This process enables you to specify a material that affects only a single limb, or just the backbone, in your warning expressions. Also keep in mind that you can do all these warnings by setting driven keys, instead of using expressions. This can produce a gradual changing of color on the polygon bones, instead of the color being full on or off. ■

- 3.76 You also can use a distanceDimension node to create a warning color for when a limb is translated too far.



## WRAPPING IT UP

Congratulations! If you made it this far, you should now have a fully functional skeleton rig. You have successfully used many of the skills and techniques that a professional character setup artist uses to create character controls. But wait, the rig is not finished. Although you have created lots of controls for animating the main parts of your character, you still have some important things to add. In the next few chapters, you create some muscle controls, face controls, and scripted MEL controls. Before you get to that, however, in the next chapter you bind your models to your current skeletons to see how they deform your skin.