第2章

标准人体：躯干的设置

在接下来的几个章节中将介绍标准人体设置，包括躯干设置、手臂设置、腿部设置、手指设置以及表情设置。在本章中我们将一起来学习标准人体——躯干的设置。

2.1 认识人体躯干

在学习本小节内容之前我们首先来分析一下如图 2-1 和图 2-2 所示的身体躯干部分的功能，通过观察我们可以清楚的知道：

- 旋转：包括胸腔的旋转，臀部的旋转（卡通角色还带有拉伸和体积的控制），这些功能主要运用 SplineIK 来控制。
- 扭曲：包括胸腔的弯曲，臀部的扭曲，这些主要需要 FK 来控制。

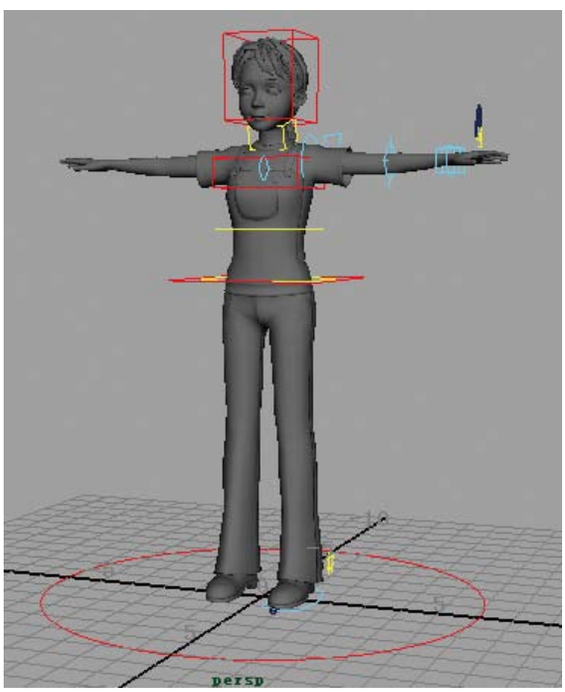


图2-1

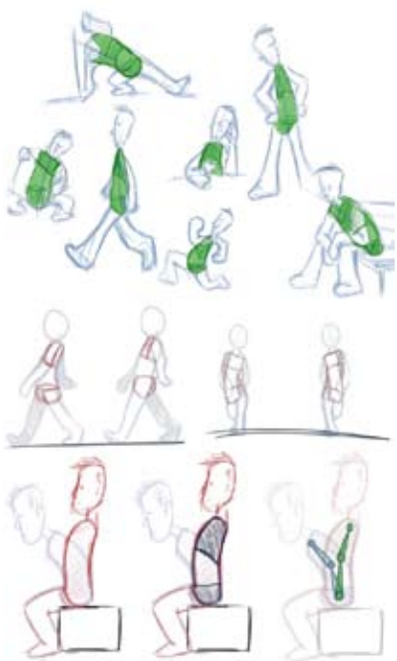


图2-2

下面将带领大家学习如何进行躯干的设置，主要内容如下：

- 线性 IK 的创建
- 线性 IK 的高级旋转
- 线性 IK 的伸缩设置
- 线性 IK 的挤压设置

在接下来的讲解过程中将会分析每一个重要的知识点，通过列举实例来具体阐述完整的制作过程，制作中加入了大量的脚本知识，一方面是设置本身需要的；另一方面希望能够帮助读者学习更多的脚本相关知识。其难点在于躯干设置的伸缩和挤压设置，学习的重点是理解 SplineIK 的各种功能，如图 2–3 所示。

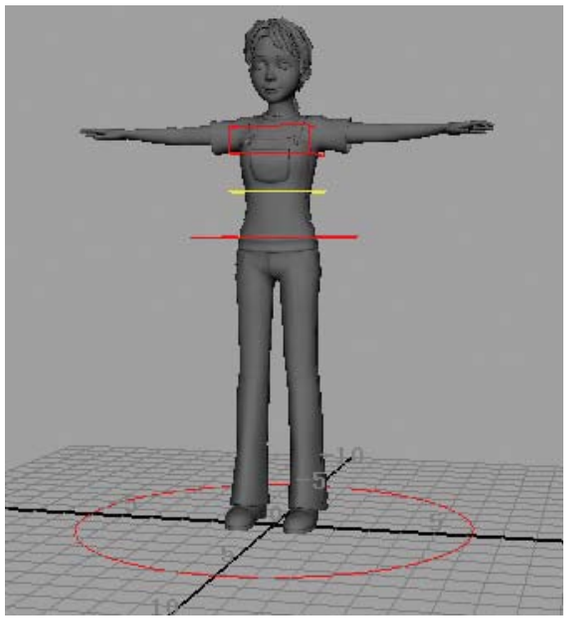


图2-3

2.2 创建线性IK

创建 spine 骨骼，并且创建一些物体模拟躯干组成，需要对这些物体进行命名，以方便制作和观察，骨骼名称和物体名称如图 2–4 所示。

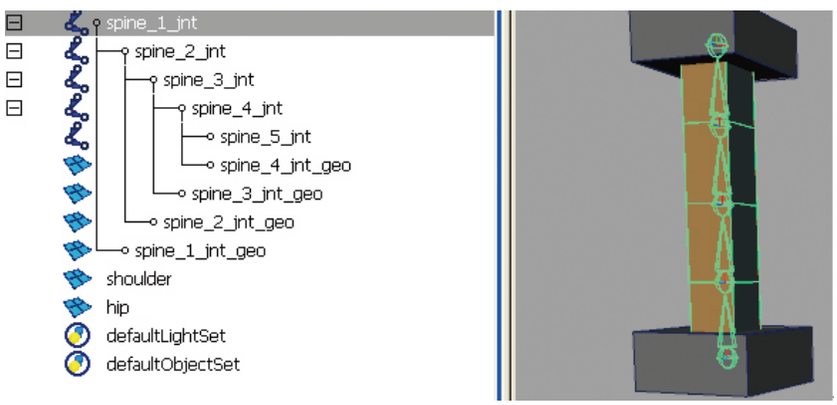


图2-4

注意：

- spine_geo为spine_jnt的子物体。
- shoulder和hip为单独的物体，用于控制。
- spine_jnt每个关节长度一致，方便RIG制作。

打开 IK Spline Handle 创建菜单，在默认情况下 Auto simple curve 选项是被勾选的，在此取消勾选该选项，因为这是自动优化曲线，若勾选 Auto simple curve 选项在创建 SplineIK 骨点时可能会偏移，如图 2–5 所示。

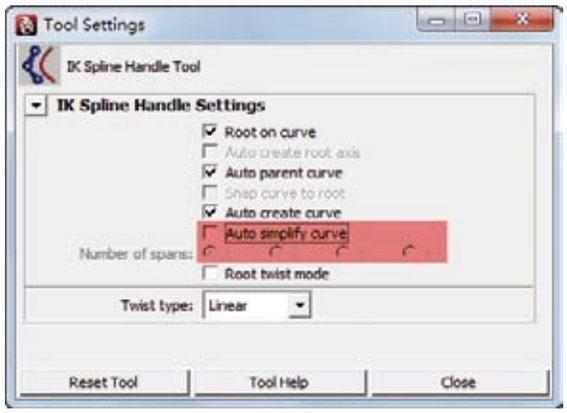


图2-5

选择骨骼 spine_1_jnt 和 spine_5_jnt，创建 splineIK，创建完成之后选择生成的曲线上面的点，如图 2–6 所示。

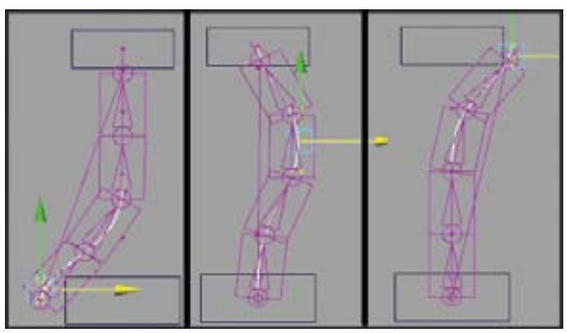


图2-6

在 hip_geo 轴心处创建骨骼 hip_jnt；在 shoulder_geo 轴心处创建骨骼 shoulder_jnt；并重新创建其层级关系，如图 2–7 所示。

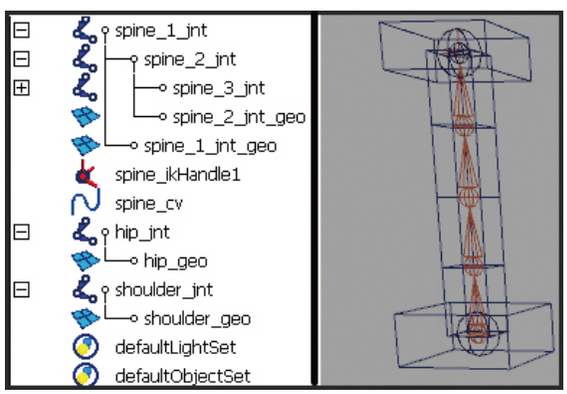


图2-7

为了更好的控制曲线，对曲线进行 Skin。选择 hip_jnt 和 shoulder_jnt，再选择曲线 spine_cv，执行 skin > Bind Skin > Smooth Bind 命令，Smooth Bind Options 设置如图 2–8 所示。

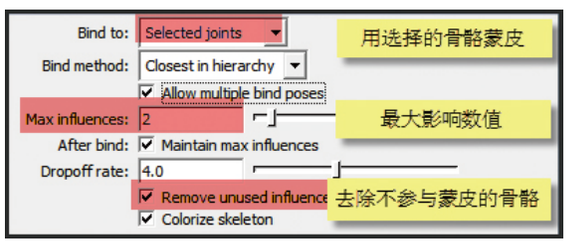


图2-8

注意：

max influences用于控制一个点最多受几个影响体影响，太多时权重分配会乱。通常情况下，我们需要再次对点的权重进行重新调整。

我们对骨骼进行一些调整（移动和旋转），发现已经开始有脊椎运动的感觉了。旋转shoulder_jnt时，spine（脊椎）跟随着一起旋转；旋转shoulder_jnt时，spine一起旋转；移动并旋转shoulder_jnt时，spine可以呈现S效果，是不是有点小蛮腰的感觉了，如图2-9和图2-10所示。

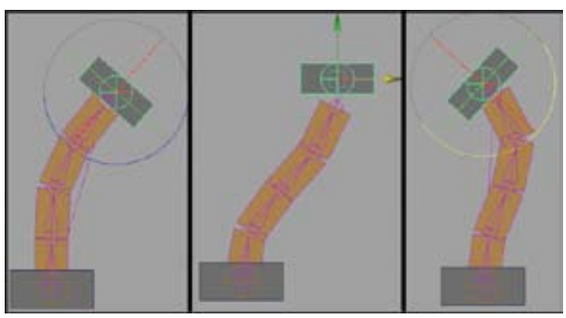


图2-9

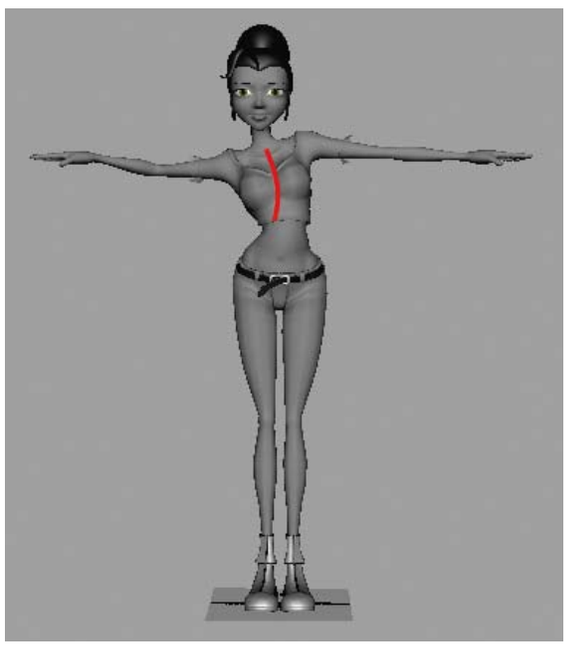


图2-10

2.3 线性IK的高级旋转

splineIK的高级旋转（Advanced Twist Controls）是一个比较特殊的功能，它能够模拟人体胸腔旋转带动脊椎整体跟随旋转的效果。

为了方便查看显示骨骼轴向，围绕Y轴旋转骨骼shoulder_jnt，spine骨骼并没有旋转，这显然不是我们所想要的结果，我们需要的是旋转shoulder_jnt时，整个腰部感觉全部跟随一起转动，如图2-11所示。

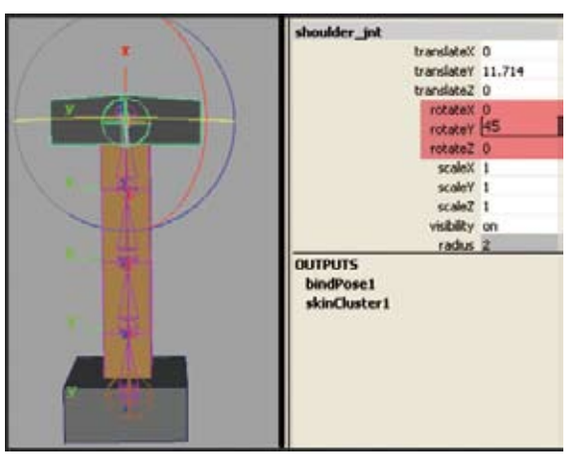


图2-11

选择 spine_ikHandle1 手柄，按组合键 Ctrl+A 打开 splineIK 属性面板，默认状态下 Enable Twist Controls 选项是关闭的，只有勾选该选项才能对其他参数进行设置，如图 2–12 所示。

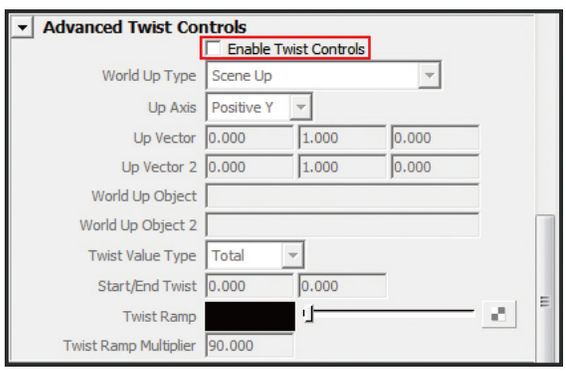


图2-12

为了实现旋转功能，勾选 EnableTwist Controls 选项，并进行设置，设置完毕之后观察左边所有的 spine_geo，发现已经被旋转控制了，如图 2–13 所示。

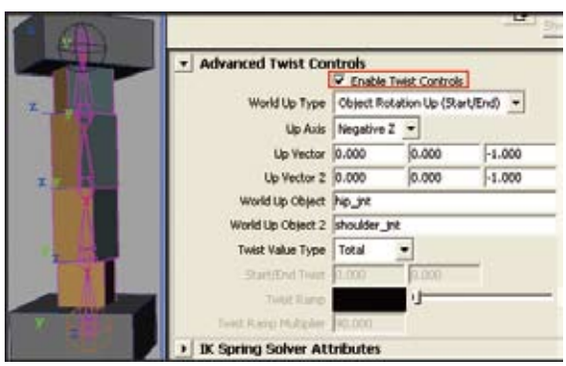


图2-13

为什么改变属性之后 spine_geo 被旋转了呢？为什么按照上述参数进行设置呢？

为了实现旋转选择了 world up type 为 object Rotation Up (Start/End)，因此必须指定所需要的控制物体，在这里就是 hip_jnt 和 shoulder_jnt。另外为了跟世界坐标轴有一个区别，防止一些特殊情况的发生（旋转锁定或者骨骼跳转），设置 up Axis 为 Negative Z（负 Z 轴），如此设置 Up Vector 必须为负 Z 轴，如图 2–14 和图 2–15 所示。

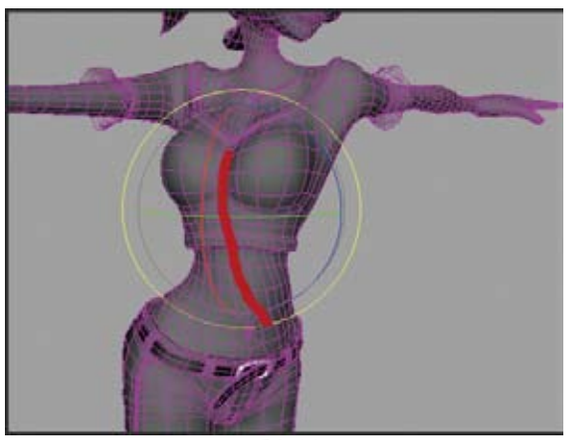


图2-14

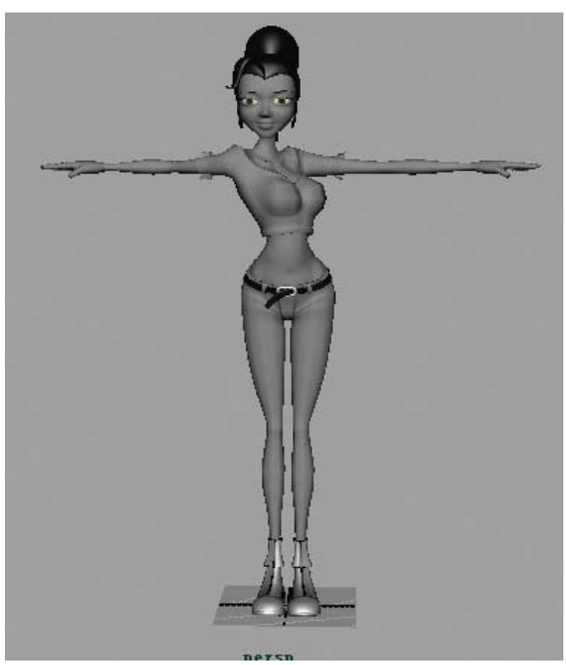


图2-15

2.4 线性IK伸缩

SplineIK 的伸缩设置对初学者来说是一个难点。在本小节中将分两个部分进行阐述，首先给大家分析 SplineIK 为什么这样做；其次给大家介绍怎么制作 SplineIK。其中所牵涉的表达式比较多。

2.4.1 原理

为了让大家更加清楚的理解伸缩的原理，首先运用 Maya 的表达式来制作伸缩，表达式更加能够说明其中的数学意义，如图 2–16 所示。

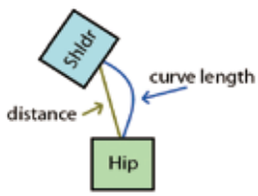


图2-16

1 选择 spine_cv，打开脚本编辑器，输入 arclen–ch 1，为 spine_cv 添加距离节点 curveInfo。打开 Window>hypergraph Input and Output Connections，可以看到已经为曲线添加了长度计算节点，如图 2–17 所示。

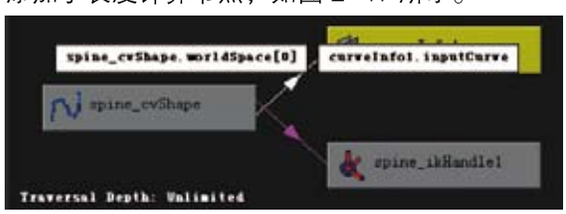


图2-17

2 选择 curveInfo1，按组合键 Ctrl+A 打开属性面板，找到 curveInfo1 选项，已经显示出来了曲线的长度（arclenght = 12.000），如图 2–18 所示。

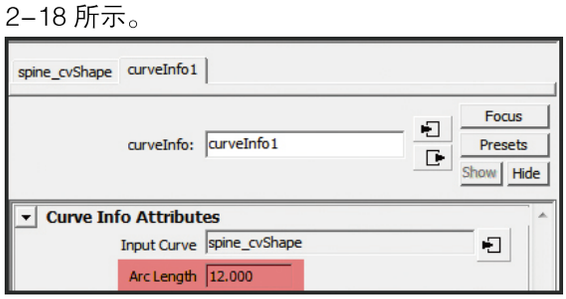


图2-18

3 接下来开始正式分析 SplineIK 缩放的原理。当骨骼创建 SplineIK 后必然会生成一条曲线，当曲线上的点或者曲线本身移动或旋转时，骨骼跟随运动。当曲线被拉长时，调整骨骼的位移即可匹配曲线的长度。那么需要处理曲线长度和骨骼之间的关系，如图 2–19 所示。

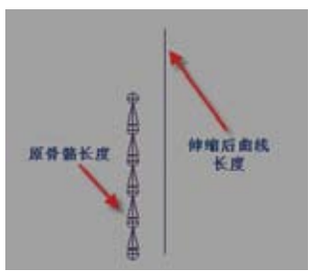


图2-19

伸缩后曲线的长度减去原骨骼长度就可以获取曲线伸缩了的长度：

曲线伸缩了的长度 = 伸缩后曲线的长度 – 原骨骼的长度

stretchDis = arcLength–origLength

说明：

（在这里，伸缩后曲线的长度用arcLength表示，原骨骼的长度用origLength表示）

接着，将曲线伸缩了的长度（stretchDis）平均分配到每节骨骼上面。注意：每根骨骼新的总长度是在每根骨骼原来长度的基础上加上新分配的长度。原始骨骼的位移值即为骨骼的长度。

每节分配的增量：

eachStretch = stretchDis/N

每节得到增量后的新长度：

jointTranslate = origTranslate(骨骼位移值) + eachStretch

● 加法原理

```
// 获取骨骼伸缩了的长度
$length = curveInfo1.arcLength - 12;
// 每一节骨骼伸缩了的长度
$eachStretch = $length/4;
// 第一节关节是不需要伸缩的
//spine_1_jnt.tx = 3 + $eachStretch;
// 每一节的长度，3为骨骼的原始长度，具体针对场景如定
spine_2_jnt.translateX = 3 + $eachStretch;
spine_3_jnt.translateX = 3 + $eachStretch;
spine_4_jnt.translateX = 3 + $eachStretch;
spine_5_jnt.translateX = 3 + $eachStretch;
```

由于乘法是加法演变而来的，所以我们也可以采用乘法公式来阐述伸缩原理。

伸缩后的长度除以原来骨骼的长度就可以获取曲线伸缩的比例：

length = arcLength / origLength

说明：

（在这里，伸缩后曲线的长度用arcLength表示，原骨骼的长度用origLength表示）

接着把总伸长的长度比例值乘以每节骨骼原始长度。

每节得到增量后的新长度：

jointTranslate = origTranslate × length

● 乘法原理

```
// 获取骨骼伸缩比例
$length = curveInfo1.arcLength / 12 ;
// 每一节的长度
spine_2_jnt.translateX = 3 * $length ;
spine_3_jnt.translateX = 3 * $length ;
spine_4_jnt.translateX = 3 * $length ;
spine_5_jnt.translateX = 3 * $length ;
```

2.4.2 制作splineIK的伸缩

在本小节中我们将运用乘法原里来制作splineIK的伸缩。从表达式的长短就可以看出，乘法原理制作更加简单方便，可以最少运用Maya节点，节点运用越少，Maya计算量越小，速度越快，因此在实际生产制作中这种方法是运用最多的。

1 选择spine_cv，复制一条曲线，将其命名为spine_cv_scale。分别为spine_cv和spine_cv_scale创建长度节点（arclen-ch 1；），将其命名为和curveInfo和scale_curveInfo，如图2-20所示。

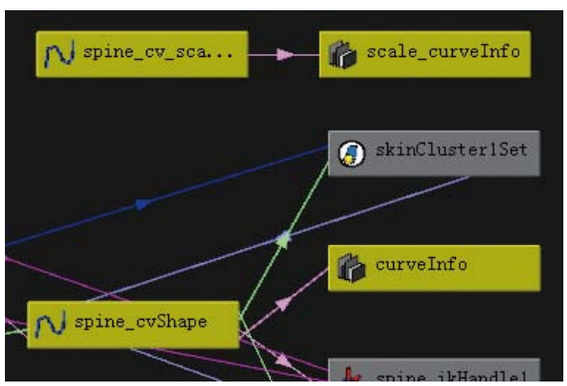


图2-20

2 创建乘除节点，设置运算方式为除法（Divide），通过乘除节点计算处spine_cv曲线相对于spine_cv_scale的缩放比例，如图2-21～图2-23所示。

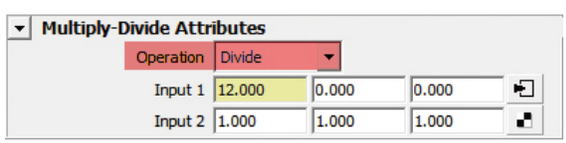


图2-21

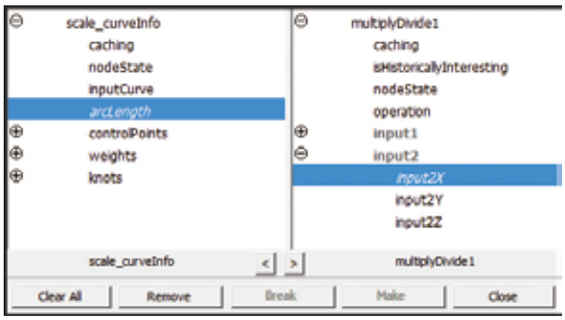


图2-22

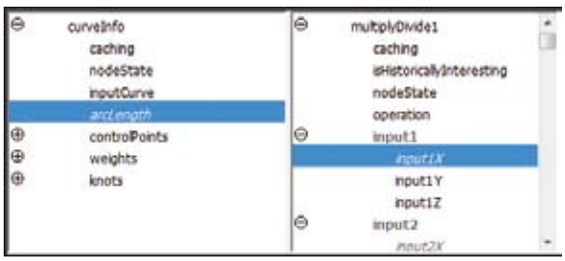


图2-23

3 创建乘除节点，设置运算方式为除法（Multiply），将 input1 设置为 3（3 为骨骼的原始长度，开始制作骨骼时，每个骨骼的长度是一样的），如图 2-24 和图 2-25 所示。

图2-24

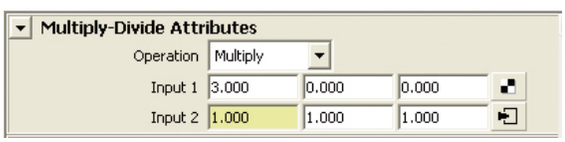


图2-25

4 输出 mltiplyDivide2 的 output 到 spine 骨骼的 tx，输出 mltiplyDivide2 的属性 outputX 到 spine 骨骼的属性 translateX 上，如图 2-26 所示。

Mel 命令：

```
connectAttr -f multiplyDivide2.outputX spine_2_jnt.translateX;
connectAttr -f multiplyDivide2.outputX spine_3_jnt.translateX;
connectAttr -f multiplyDivide2.outputX spine_4_jnt.translateX;
connectAttr -f multiplyDivide2.outputX spine_5_jnt.translateX;
```

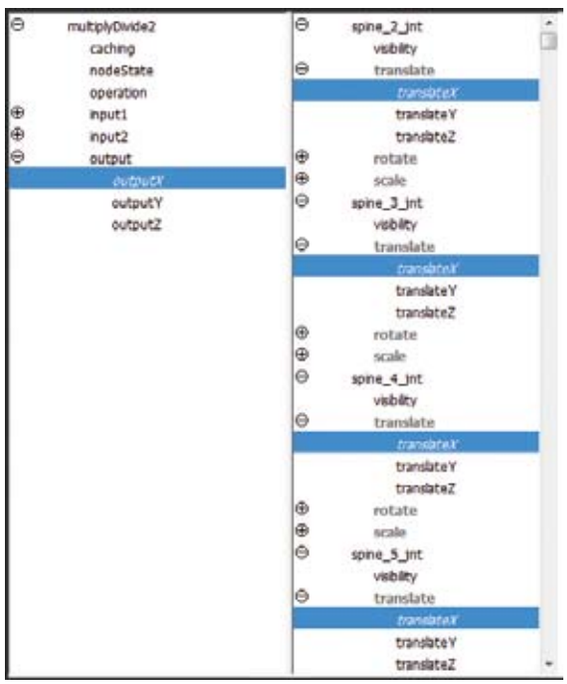


图2-26

5 制作完毕，展开超图查看连接，如图 2-27 所示。

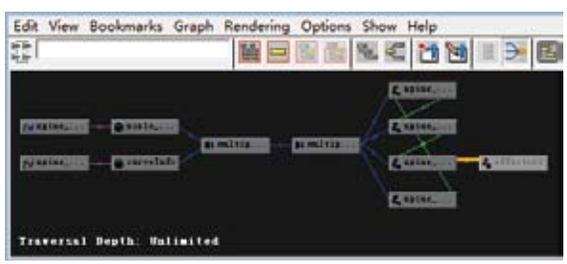


图2-27

6 实例示意图 1 如图 2–28 所示。

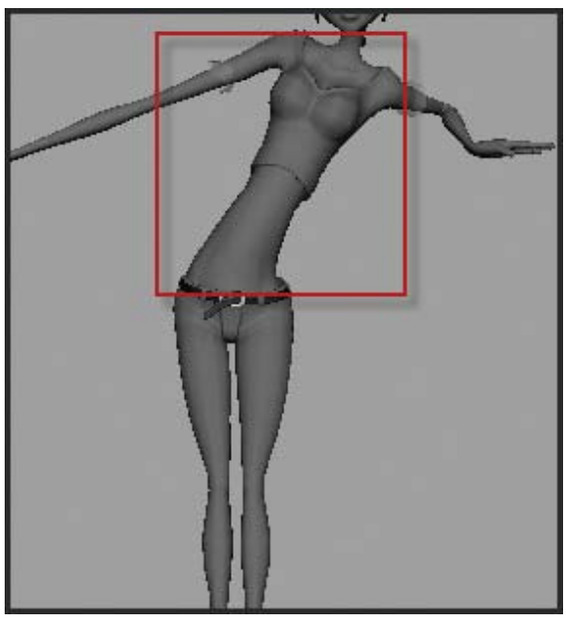


图2-28

7 实例示意图 2 如图 2–29 所示。

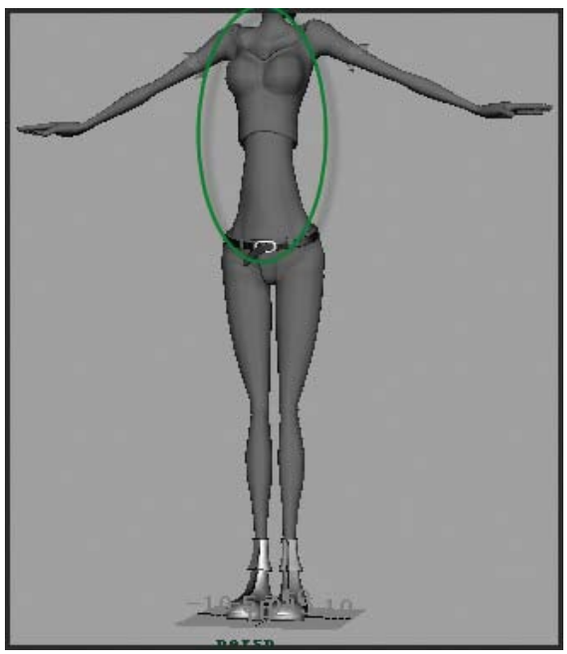


图2-29

2.5 线性IK挤压

在开始学习如何制作线性 IK 挤压变形(Squash)效果之前，使 spine_ref 被 spine_#_jnt 骨骼蒙皮，这样处理之后可以很容易观察到蒙皮变形（ strecth&squash ）的效果，如图 2–30 所示。

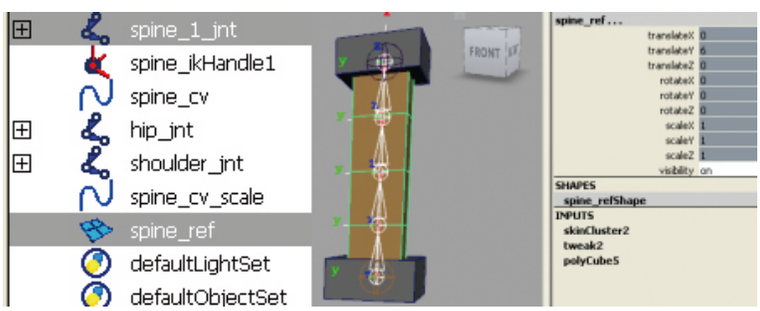


图2-30

注意：

应当注意一下两点：

① #代表“1、2、3、4、5”等序列数

② frameCache节点：帧缓存节点。缓存的浮点型数值“stream”存储关键帧信息。

1 在 shoulder_jnt 上添加 powScale 和 normalizeScale 属性，并且在 powScale 上设置驱动关键帧，并调整关键帧曲线的弧度。曲线的弧度即为 spine_ref 变形结果，如图 2–31 所示。

图2-31

注意：

分别在第1帧和第5帧，K帧创建关键帧，因为初始设置spine有5根骨骼，这点是非常重要的!

2 将关键帧输出到 shoulder_jnt.powScale 上，并且将 spine_cv_scale 和 spine_cv 两条曲线的长度相除后的结果输出到 shoulder_jnt.normalizeScale 上，如图 2–32 所示。

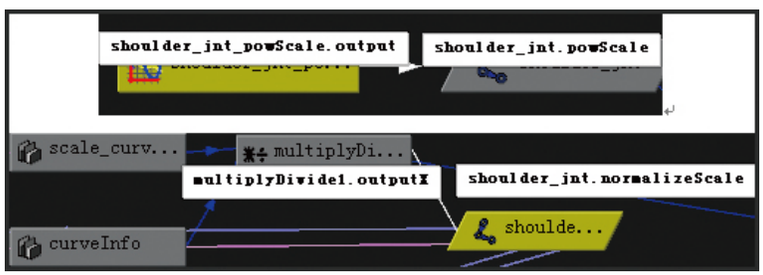


图2-32

注意：

normalizeScale所产生的变化会影响骨骼伸缩后的squash。

1 创建 5 个“frameCache”节点（createNode“frameCache”；）。

2 分别将 shoulder_jnt.powScale 连接到 frameCache#.stream 上。

3 在 spine_#_jnt 上创建 pow 属性，分别进行一下连接。

```
connectAttr -f frameCache1.varying spine_1_jnt.pow;
connectAttr -f frameCache2.varying spine_2_jnt.pow;
connectAttr -f frameCache3.varying spine_3_jnt.pow;
connectAttr -f frameCache4.varying spine_4_jnt.pow;
connectAttr -f frameCache5.varying spine_5_jnt.pow;
```

3 创建以下表达式。$pow 所表现出来的意义是为了通过曲线的伸缩来控制 squash 的变化，如图 2-33 所示。

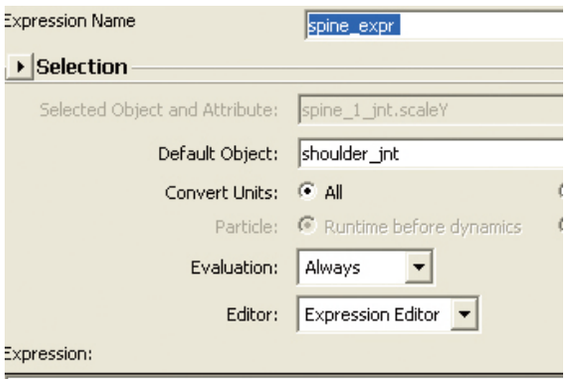


图2-33

制作完毕，查看效果是否是是我们想要的效果，通过调整动画曲线弧度来调整挤压的效果。其中一张图片为示例效果图，另外一张为实例效果图。如图 2-34 和图 2-35 所示为挤压的 squash 效果。

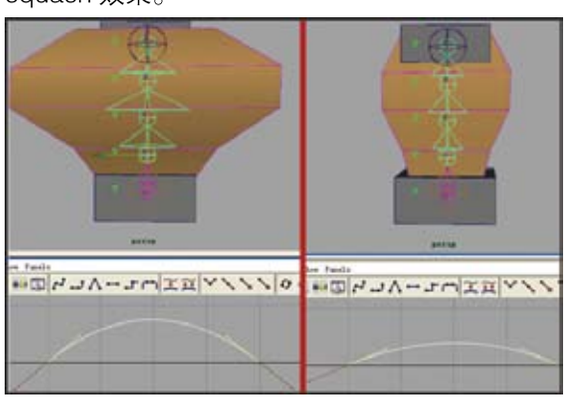


图2-34

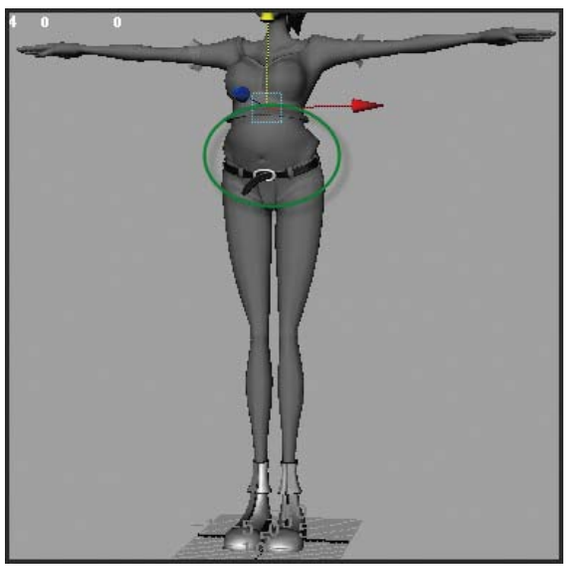


图2-35

图 2-36 和图 2-37 所示为拉长伸缩挤压效果。

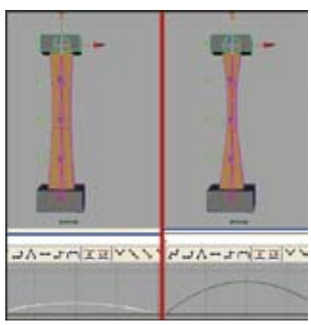


图2-36

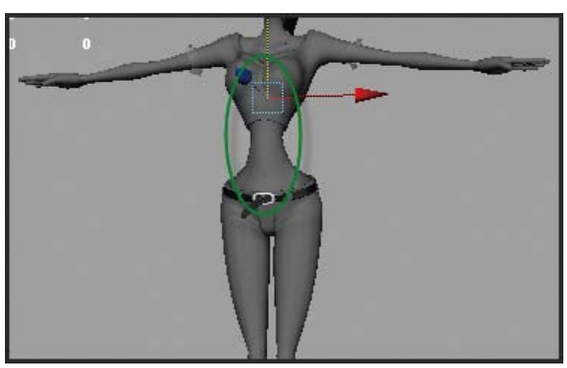


图2-37

2.6 线性IK伸缩脚本

在本小节将向大家阐述一个创建伸缩 splineIK 脚本是如何编写的，以及在编写的时候需要注意哪些事项。希望通过本小节脚本的练习大家能够更好的理解伸缩的原理，也能够更加明白脚本编写的基本知识。

1 演示场景如图 2-38 所示。

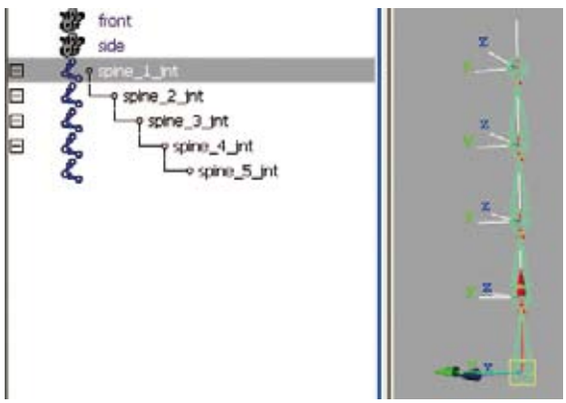


图2-38

2 根据物体创建曲线，目的是在 spine 骨骼上创建曲线，Maya 默认生成的曲线总是不能完全达到想要的效果。我们创建了与骨骼完全匹配的一条曲线，并将其命名为 spine_curve，如图 2-39 所示。

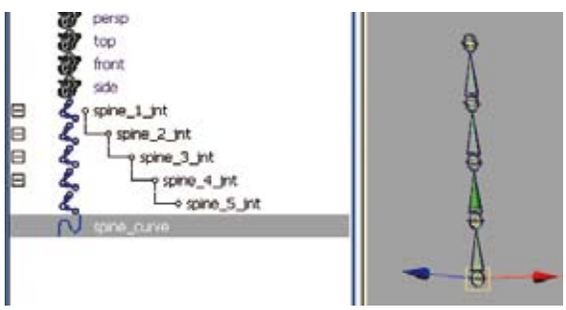


图2-39

在这里讲解一下基本代码：

```
ly_createCurveByJoints(_selJnts,_prefix)
#该函数拥有两个参数，_selJnts是选择的骨骼，_prefix是命名空间前缀。
# 导入Maya的Python模块
from maya.cmds import *
# 该函数定义了两个参数，分别代表这所有选择的骨骼和命名空间
def ly_createCurveByJoints(_selJnts,_prefix):
# 定义一个空数组用来存储坐标位置
 _selJntsPos = []
# for循环，我们一次一般会选择几根骨骼
 for _selJnt in _selJnts:
# 获取单个骨骼的世界坐标的位置
_selJntPos = xform(_selJnt,q=True,ws=True,t=True)
# 将获取的坐标信息添加进定义的空数组
_selJntsPos.append(_selJntPos)
#print _selJntsPos
# 创建曲线
_curveTmp = curve(n=_prefix+'curve',d=3,p=_selJntsPos)
# 返回
return _curveTmp
```

以下这个函数的作用是：通过起始骨骼和末端骨骼来获取中间骨骼，大多数情况下骨骼是未知的，不可能每次都像我们演示的这样名称很规范，因此我们必须这样做。

我们先测试一下写好的脚本所达到的效果：

定义如下：

```
_startJnt = 'spine_1_jnt'
_endJnt = 'spine_5_jnt'
```

执行脚本 ly_getJointByStartEnd(_startJnt,_endJnt) 得到 Maya 返回值为：

```
Result: ['spine_1_jnt', u'spine_2_jnt', u'spine_3_jnt', u'spine_4_jnt', u'spine_5_jnt'] #
```

很明显，我们得到了需要的骨骼，接下来开始讲解这个脚本。完全理解这个脚本，我们就会理解 Maya 究竟是怎么选择层级下面所有物体的。

```
    # 函数定义两个参数，分别为起始骨骼和末端骨骼。
    def ly_getJointByStartEnd(_startJnt,_endJnt):
    # 创建unknown节点作为辅助物体，来识别骨骼相关信息
    _node = createNode('unknown')
    # 为unknown添加属性
    addAttr(ln="selObjects" ,at='message' , multi=1,im=0)
    # 获取_startJnt下面所有骨骼
    _jnts = listRelatives(_startJnt,f=1,ad=1)
    # 分别将jnt的信息连接到unknown上
    for _jnt in _jnts:
    connectAttr(_jnt+'.message',_node+".selObjects",na=True)
    # 列举所有与unknown节点相关联的物体
    _con = listConnections(_node+".selObjects")
    # 定义空数组
    _allJnts = []
    # 将所有与unknown节点相关联的物体添加进_allJnts数组
    for i in range(len(_con)):
    _tempA = listConnections(_node+".selObjects["+str(i)+"]")
    _tempB = _tempA[0].split('|')
    _shortName = _tempB[len(_tempB)-1]
    _allJnts.append(_shortName)
    # 我们所需要获取的是起始骨骼到末端骨骼这些骨骼，以下这些是对数组_allJnts进行处理，获取我们需要的那部分
    _numEndJnt = _allJnts.index(_endJnt)
    _allJnts = _allJnts[_numEndJnt:]
    _allJnts.reverse()
    _allJnts.insert(0,_startJnt)
    # 删除辅助节点
    delete(_node)
    # 返回
    return _allJnts
```

我们准备了那么多了，现在可以开始创建splineIK，制作伸缩了，来测试一下所有脚本编写完成后的效果。

定义一下参数：

```
_startJoint = 'spine_1_jnt'
_endEffector = 'spine_5_jnt'
_prefix = 'spine_'
```

执行命令：

```
ly_createSplineIK(_startJoint,_endEffector,_prefix)
```

调整曲线上面的点，发现骨骼是跟着一起移动的，达到了我们需要的缩放效果，如图 2-40 所示。

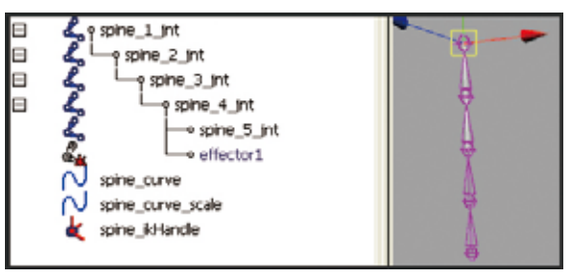


图2-40

```
    # 函数定义了三个参数，分别代表起始骨骼，末端骨骼和命名空间。
    #create spline ik
    def ly_createSplineIK(_startJoint,_endEffector,_prefix):
    # 获取起始骨骼和末端骨骼即中间骨骼
    _allJnts = ly_getJointByStartEnd(_startJoint,_endEffector)
    # 依据骨骼生成对应曲线
```

```
    _spIKCurve = ly_createCurveByJoints(_allJnts,
_prefix)
    # 复制另外一条曲线用于stretch的制作
    _spIKScaleCurve = duplicate(n=_spIKCurve+
'_scale',rr=True)
    # 创建splineIK，这里是通过我们自定义的曲线
创建IK，那么必须curve=_spIKCurve
    _spIK = ikHandle(n=_prefix+'ikHandle',sj=
_startJoint,ee=_endEffector,\
     sol='ikSplineSolver',ccv=0,scv=0,sticky=1,w=1,
curve=_spIKCurve)
    # 创建长度节点来获取权限的长度
    _curveInfoSpIKCurve = arclen(_spIKCurve,ch=1)
    _curveInfoSpIKScaleCurve = arclen
(_spIKScaleCurve,ch=1)
    # 创建乘除节点
    _scaleMD = createNode ('multiplyDivide',n=
_prefix+'scale_MD#')
    # 设置乘除节点的操作属性为除法
    setAttr(_scaleMD+'.operation',2)
    # 连接乘除节点，获取缩放比例
    connectAttr(_curveInfoSpIKCurve+'.arcLength',
_scaleMD+'.input1X',f=True)
    connectAttr(_curveInfoSpIKScaleCurve+
'.arcLength',_scaleMD+'.input2X',f=True)
    # 分别为每个骨骼连接缩放属性
    #create stretch
    for i in range(len(_allJnts)):
    #print _allJnts[i]
    _tx = getAttr(_allJnts[i]+'.tx')
    _scaleMDtx=createNode ('multiplyDivide',n=_prefix+
_allJnts[i]+'_scale_MD#')
    setAttr(_scaleMDtx+'.input1X',_tx)
    connectAttr(_scaleMD+'.outputX',_scaleMDtx+
'.input2X')
    connectAttr(_scaleMDtx+'.outputX',_allJnts[i]+'.tx')
```

到此整个脚本已经制作完成，达到了我们所需要的效果，在超图中检查最终的连接，如图2-41所示。

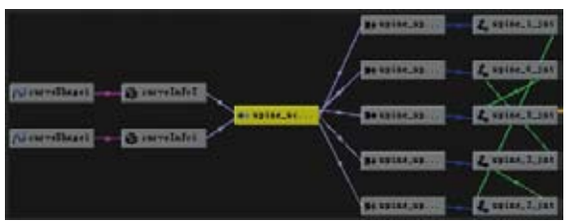


图2-41

大家也许会有疑问，为什么制作那么多的步骤，有那么多的效果，为什么我们上面的脚本不会把所有的工作都完成呢？我们在平时写脚本的时候，应当注意脚本的通用性，可能各个项目对控制的要求不一样，基础脚本写好了，针对不同的项目写出不同的控制，不要一味的去套用脚本，这样可能会出现一些不必要的错误。

2.7 标准人体躯干设置

在前面的5个小节中笔者花了大量的时间来讲解线性IK的知识，目的是为了方便本小节的学习，在实际的项目制作中有一些需求，具体应该怎么实现呢？在这里以标准的模型为例，为大家介绍怎么样搭建一个人体的躯干。

在进行角色设的时候，通常都是从躯干开始制作的，在这里我们以一个女性的人体模型作为绑定的基础模型，如图 2-42 所示。

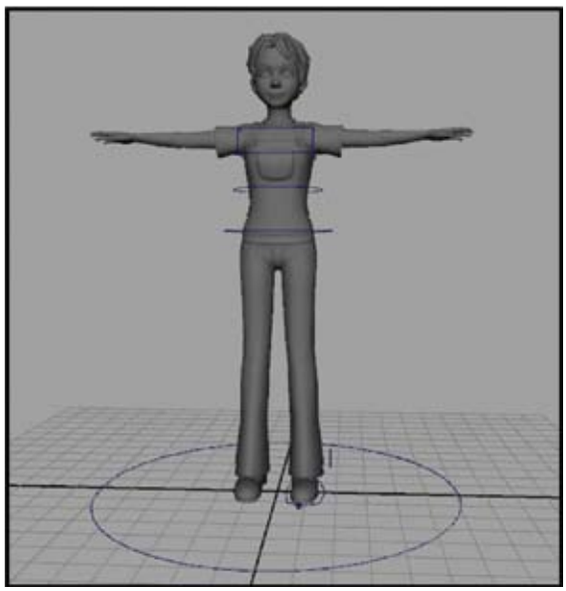


图2-42

2.7.1 创建线性IK

1 创建一个整体控制器，命名为 all_anim，这个控制器用来控制整个角色的位移旋转和缩放，为了更方便的控制缩放，将 scaleY 的名称改为 globalScale，并且将 scaleY、scaleX 和 scaleZ 相关联，隐藏 scaleX,scaleZ 属性，如图 2-43 所示。

修改控制器（all_anim）属性（scaleY）的方法：

- 将 all_anim 的属性 scaleY 的名称改为 globalScale。
- 执行脚本 aliasAttr “globalScale” “all_anim.scaleY”，将属性 scaleY 修改为 globalScale。

修改角色基本属性脚本：

这里将 all_anim 的属性 scaleY 的名称改为 globalScale。

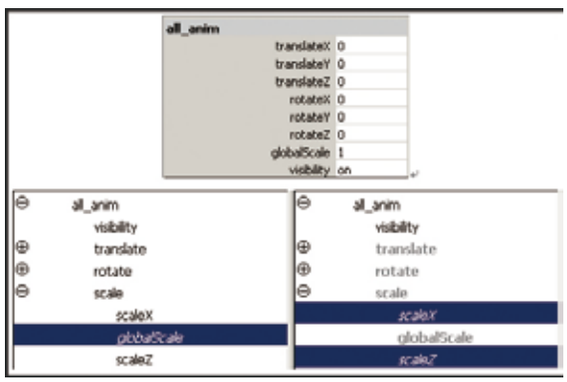


图2-43

2 在 side 视图创建骨骼和控制器作为设置的基础，如图 2-44 所示。

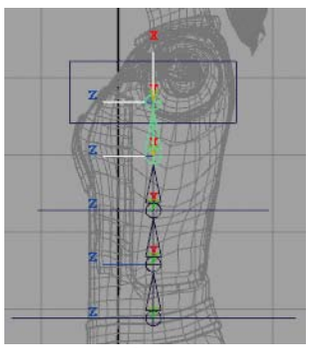


图2-44

3 将骨骼命名，如图 2-45 所示。

图2-45

4 创建控制器，命名如图 2-46 所示。

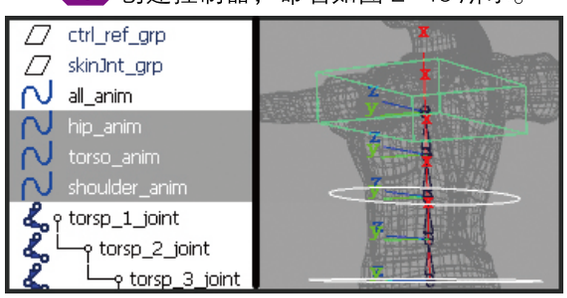


图2-46

5 依附每一个骨点创建一条曲线，作为线性 IK 使用的曲线，将其命名为 torso_splineIK_curve，如图 2–47 所示。

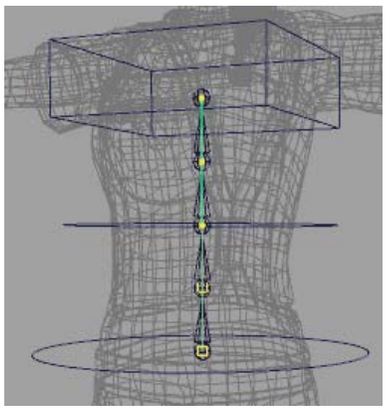


图2-47

6 打开创建 splineIK 属性面板，设置属性如图 2–48 所示。

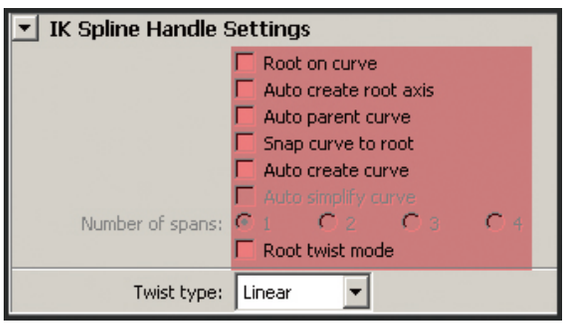


图2-48

7 选择 toro_1_joint、torso_5_joint、torso_splineIK_curve，即可创建 splineIK，命名如图 2–49 所示。

图2-49

2.7.2　创建高级旋转

1 创建 3 个蒙皮骨骼，将其分别命名为 hip_curve_skin、torso_curve_skin、shoulder_curve_skin，骨骼的轴心和控制器相同，如图 2–50 所示。

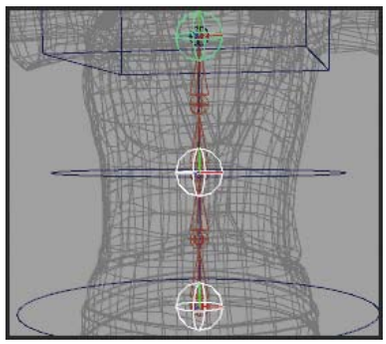
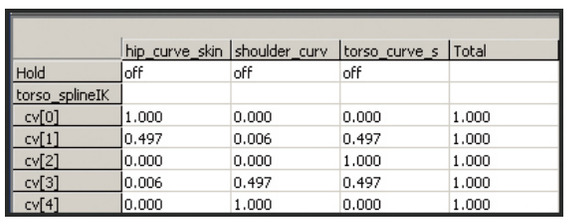


图2-50

2 选择 3 根骨骼 hip_curve_skin、torso_curve_skin 和 shoulder_curve_skin，选择曲线 torso_splineIK_curve，创建蒙皮，在元素编辑器里面查看并且编辑点权重，如图 2–51 所示。



	hip_curve_skin	shoulder_curv	torso_curve_s	Total
Hold	off	off	off	
torso_splineIK				
cv[0]	1.000	0.000	0.000	1.000
cv[1]	0.497	0.006	0.497	1.000
cv[2]	0.000	0.000	1.000	1.000
cv[3]	0.006	0.497	0.497	1.000
cv[4]	0.000	1.000	0.000	1.000

图2-51

3 创建线性 IK 的高级旋转，具体参数如图 2–52 所示。

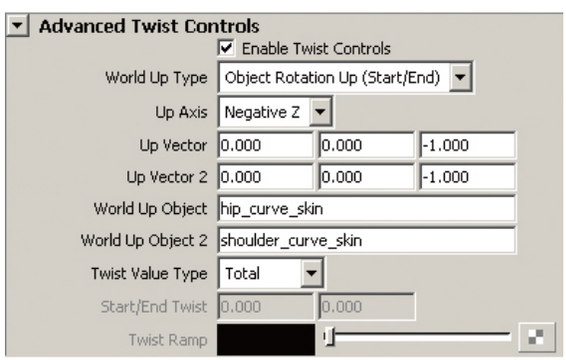


图2-52

4 创建完毕高级旋转之后需要检查旋转是否能用，骨骼的移动是否正常，如图 2–53 所示。

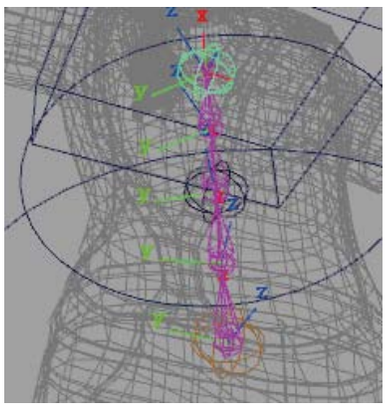


图2-53

2.7.3 创建伸缩IK

1 复制曲线 torso_splineIK_curve，并且将其命名为 torso_splineIK_scale_curve。

为 torso_splineIK_scale_curve 和 torso_splineIK_curve 分别添加长度节点信息 curveInfo，分别命名为 torso_scale_curveInfo 和 torso_curveInfo。创建 curveInfo 节点命令为 arclen –ch 1，如图 2–54 所示。

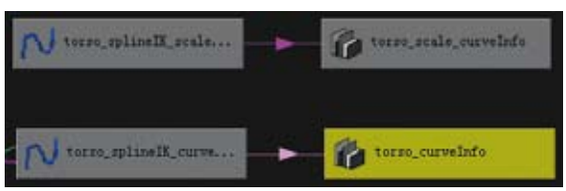


图2-54

2 创建乘除节点，设置操作方式为除法，将其命名为 torso_MD，如图 2–55 所示。

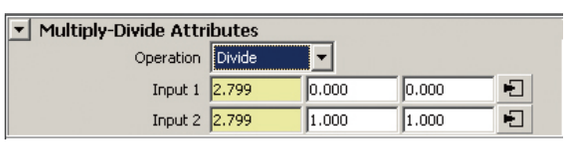


图2-55

3 连接长度节点的属性 arlength 到 torso_MD 的属性 input，如图 2–56 所示。

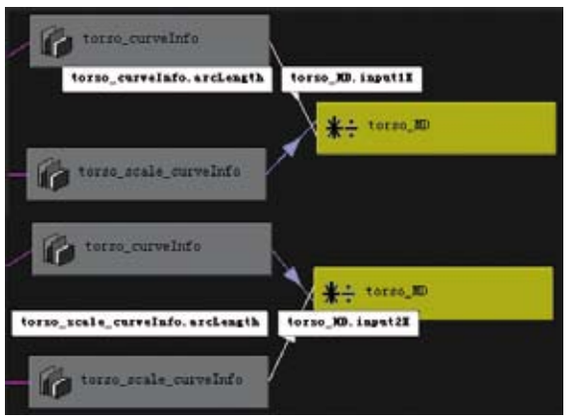


图2-56

4 创建乘除节点，将其命名为 torso_joint_trans_MD，设置乘除节点的 Input1X 为 0.7。乘除节点的连接设置如图 2–57 所示。

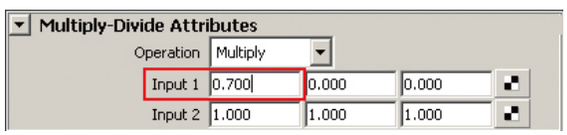


图2-57

提示：

0.7为每个骨点的位移值，初始设置每个torso骨骼是等分的，如图2-58所示。

图2-58

5 输出 torso_joint_trans_MD 到每个骨骼的 translateX 上，连接如图 2–59 所示。

MEL 命令：

```
connectAttr -f torso_joint_trans_MD.outputX
torso_2_joint.translateX;
connectAttr -f torso_joint_trans_MD.outputX
torso_3_joint.translateX;
connectAttr -f torso_joint_trans_MD.outputX
torso_4_joint.translateX;
connectAttr -f torso_joint_trans_MD.outputX
torso_5_joint.translateX;
```

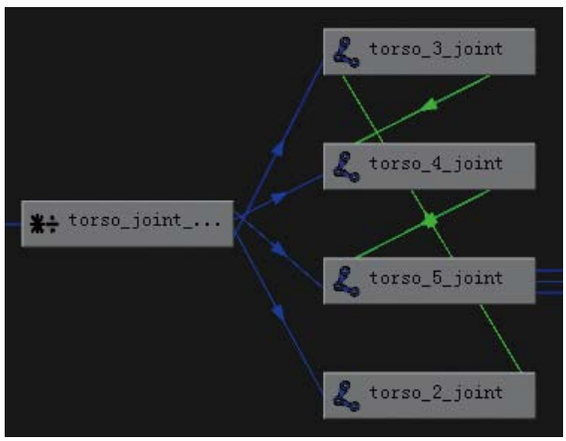


图2-59

6 骨骼的伸缩已经制作完毕，下面为骨骼的伸缩添加开关，在 hip_anim 上添加浮点型属性 autoStretch，最小值为 0，最大值为 1，如图 2–60 所示。

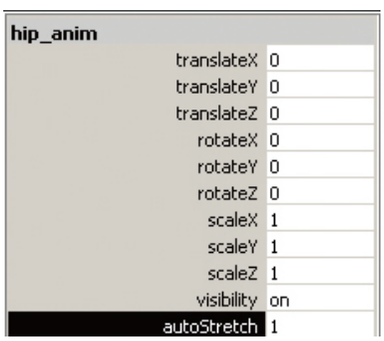


图2-60

7 创建判断节点，通过判断节点来连接制作伸缩开关，并进行命名、连接和设置等，如图 2–61 所示。

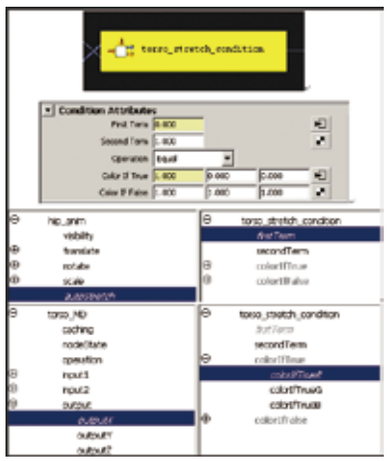


图2-61

8 伸缩制作完毕后对比伸缩开关打开和关闭时骨骼变化的效果，如图 2–62 所示。

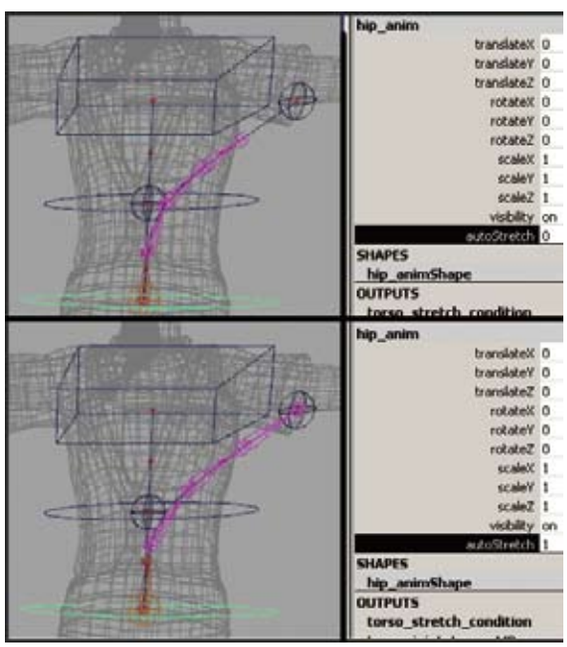


图2-62

2.7.4 创建线性IK挤压

1 在 torso 每根骨骼上添加浮点型属性 pow，做为挤压的变量，如图 2–63 所示。

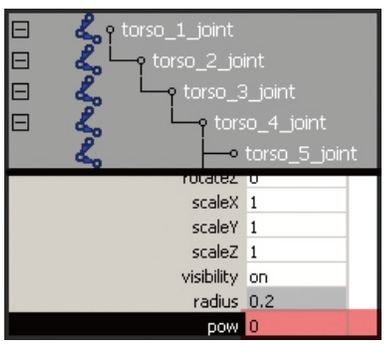


图2-63

2 在 hip_curve_skin 上添加属性 powScale 和 normalize。在 powScale 上创建关键帧 K 帧，第 1 帧，数值为 0；第 5 帧，数值为 0，如图 2–64 所示。

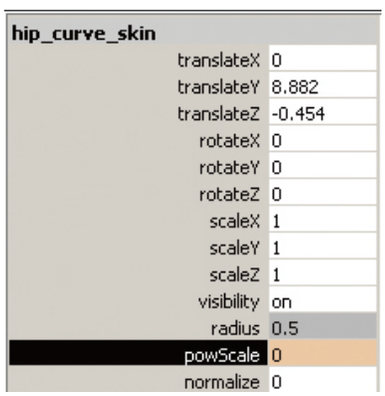


图2-64

3 调整动画曲线的曲率，如图 2–65 所示。

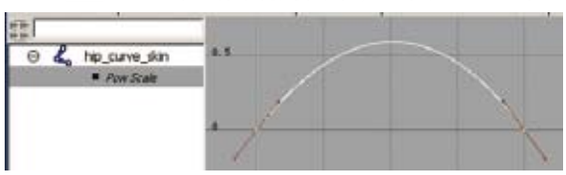


图2-65

4 连接 normalize 属性，如图 2–66 所示。

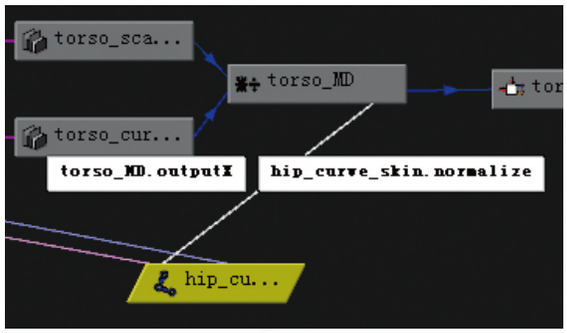


图2-66

5 创建 5 个 frameCache 节点，命名如图 2–67 所示。

图2-67

6 分别将 hip_curve_skin.powScale 连接到 5 个 frameCache 的属性 stream 上；并且连接 frameCache 的属性 varying 值到 torso_joint 的 pow 属性上，torso_frameCache1 如图 2–68 所示；torso_frameCache2 如图 2–69 所示，torso_frameCache3 如图 2–70 所示；torso_frameCache4 如图 2–71 所示；torso_frameCache5 如图 2–72 所示。

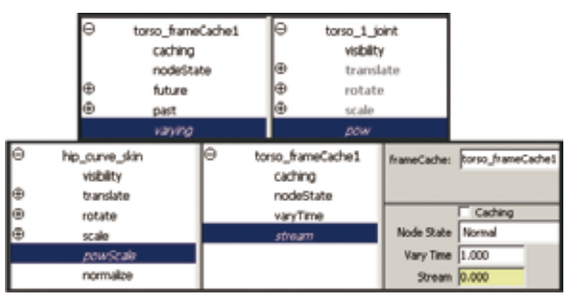


图2-68

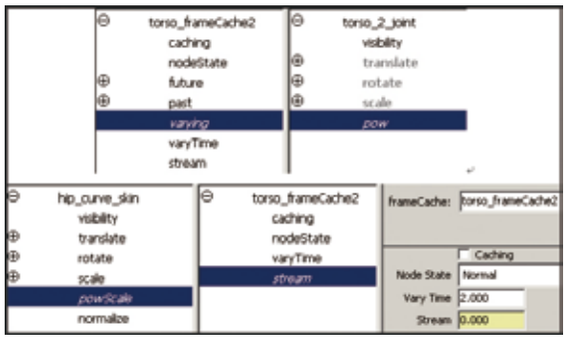


图2-69

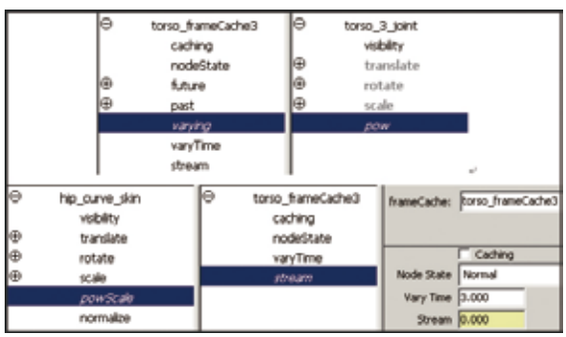


图2-70

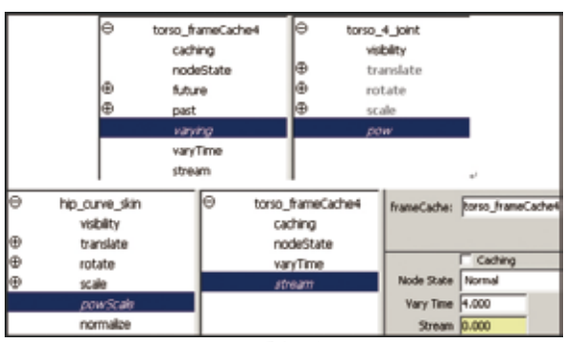


图2-71

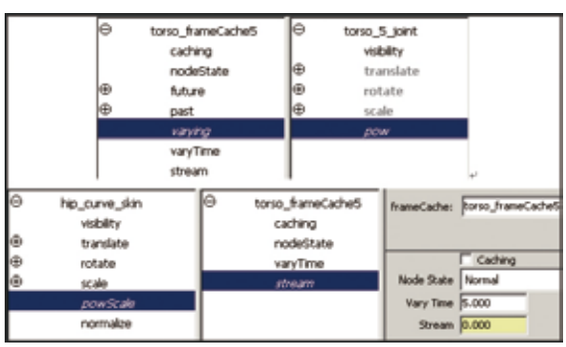


图2-72

7 在 hip_anim 上添加浮点型属性 squash，如图 2–73 所示。

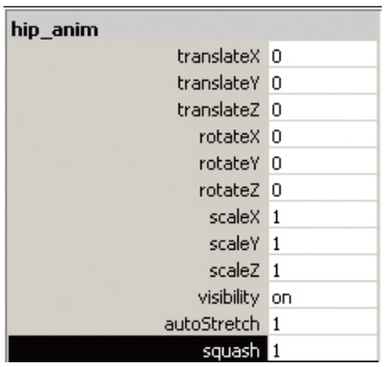


图2-73

8 为 torso_joint 骨骼添加表达式，创建 squash 效果，表达式如下。

```
$pow = sqrt(1/hip_curve_skin.normalize);
$squash = hip_anim.squash;
torso_1_joint.scaleY = pow($pow,torso_1_joint.pow) * $squash;
torso_1_joint.scaleZ = pow($pow,torso_1_joint.pow ) * $squash;

torso_2_joint.scaleY = pow($pow,torso_2_joint.pow) * $squash;
torso_2_joint.scaleZ = pow($pow,torso_2_joint.pow) * $squash;

torso_3_joint.scaleY = pow($pow,torso_3_joint.pow) *$squash;
torso_3_joint.scaleZ = pow($pow,torso_3_joint.pow) * $squash;

torso_4_joint.scaleY = pow($pow,torso_4_joint.pow) * $squash;
torso_4_joint.scaleZ = pow($pow,torso_4_joint.pow) * $squash;
```

```
    torso_5_joint.scaleY = pow($pow,torso_5_joint.pow) * $squash;
    torso_5_joint.scaleZ = pow($pow,torso_5_joint.pow) * $squash;
```

表达式截图如图 2–74 所示。

图2-74

2.7.5 整理躯干RIG

1 躯干 torso 设置完毕之后，接下来整理整个绑定，这一步是非常重要的！合理的 group 会让我们的 RIG 看起来非常棒，很专业，同时也会让以后的修改更加简单，如图 2–75 所示。

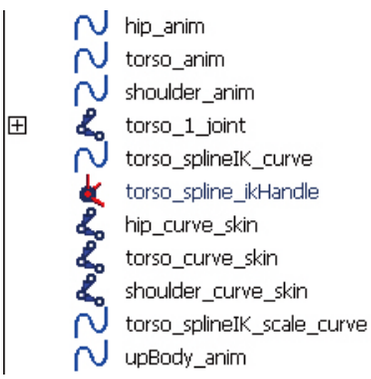


图2-75

2 为控制器 hip_anim、torso_anim 和 shoulder_anim 创建组，调整对应的蒙皮骨骼成为控制器的子物体，并将其命名为 torso_ctrl_grp，如图 2–76 所示。

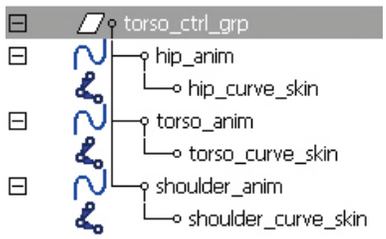


图2-76

3 group 所有和 torso 相关的物体命名为 torso_grp，如图 2–77 所示。

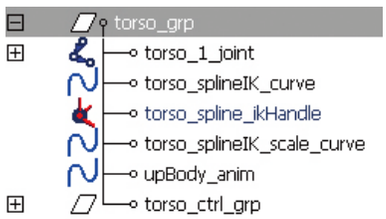


图2-77

4 为每个控制器创建 group，group 的轴心即为控制器的轴心，如图 2–78 所示。

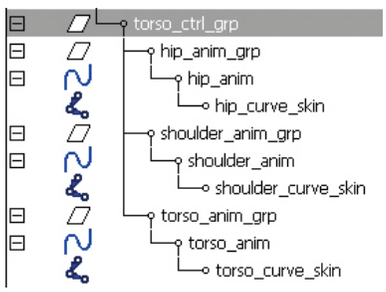


图2-78

5 使用 torso_anim 父子约束 shoulder_anim_grp，如图 2–79 所示。

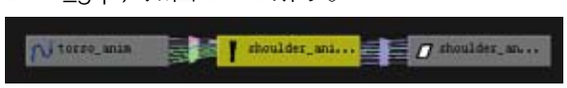


图2-79

6 使用 hip_anim 父子约束 torso_anim_grp，如图 2–80 所示。

图2-80

7 在 hip_anim 上添加浮点型属性 IKFK，最小值为 0，最大值为 1。连接 hip_anim 的属性 IKFK 到约束节点的属性。当 IKFK 为 0 时，控制即为 IK 模式；当 IKFK 为 1 时，控制即为 FK 模式。通过连接 hip_anim 的属性 IKFK 到约束节点的属性 hip_animW0 和 torso_animW0，来控制是否是 torso 的 IK 或者 FK，通过连接约束节点来控制器是否是 torso 的 IK 或者 FK，连接方式如图 2–81 所示。

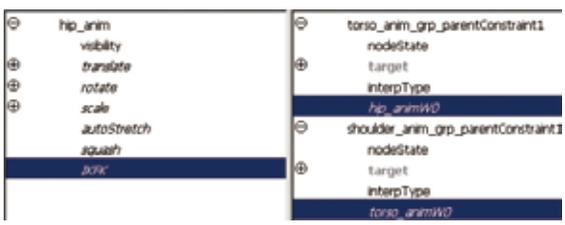


图2-81

8 设置 torso_ctrl_grp 的轴心为 hip_anim 的轴心，使用 upBody_anim 父子约束 torso_ctrl_grp，如图 2–82 所示。

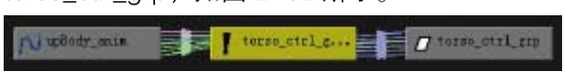


图2-82

9 将 torso_1_joint 添加进 torso_ctrl_grp，如图 2–83 所示。

图2-83

10 整体缩放 torso_grp，我们发现存在曲线（torso_splineIK_curve）的双倍位移问题，因此将隐藏曲线并关闭曲线（Inherits Transform），如图 2–84 所示。

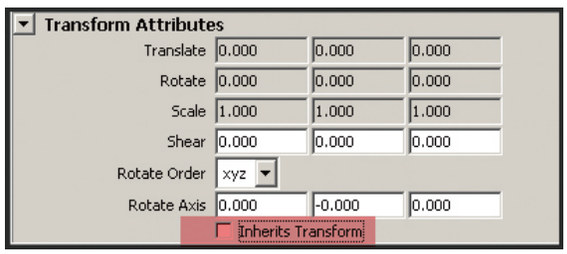


图2-84

通过上述操作基本完成了整个腰部的设置，为腰部添加了高级旋转、伸缩、挤压及 IKFK 切换等效果。整个设置的控制也比较简单，动画师可以很轻易的操作。在 IK 控制的状态下，可以实现扭臀的效果，特别是女性角色，在 FK 控制器的状态下，可以实现很轻易的弯腰动作，腰部的讲解到此为止，如图 2–85 所示。

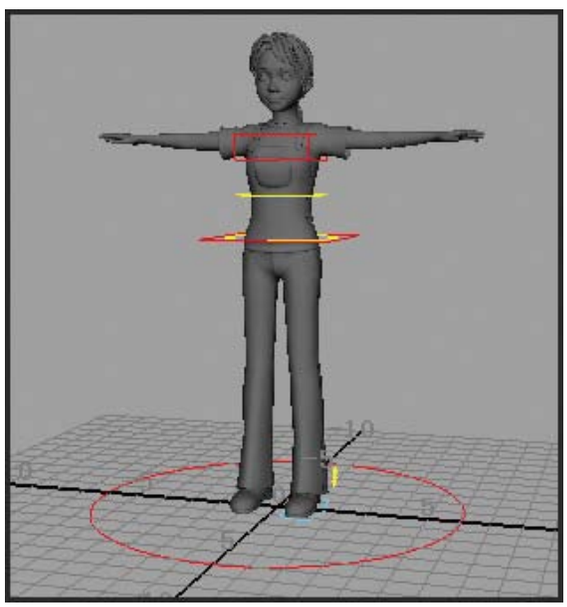


图2-85

2.8 neck & head设置

这一节，讲解 neck&head 的设置。Neck&head 的设置的重点和难点主要是 follow 效果的制作。在本小节中我们主要讲解如何创建 Follow 以及 follow 的原理。

2.8.1 zeroGrp原理

本小节将向大家讲解一个小技巧，就是如何让有位移或者旋转至的物体归“零”。

1 制作控制器的时候，往往需要调整控制器的轴向和骨骼的轴向相同，如果不对控制器 group，控制器会产生数值，如图 2–86 所示。

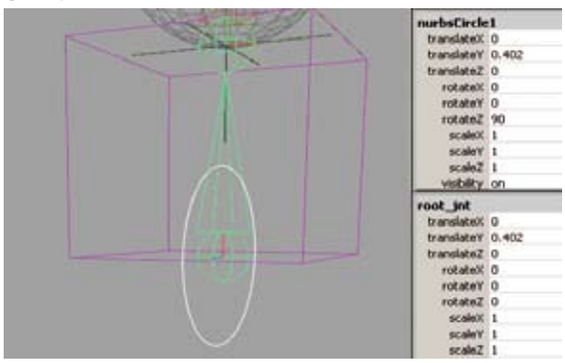


图2-86

2 nurbsCircle1 是在世界坐标系下创建的，自身坐标是依附世界坐标的。用 root_jnt 父子约束后，删除约束，曲线和骨骼轴向相同，位移相同，但是发现曲线是有数值的，显然这样不能成为动画需要的控制器。可以在控制器上创建 group，用骨骼先来约束 group，如图 2–87 所示。

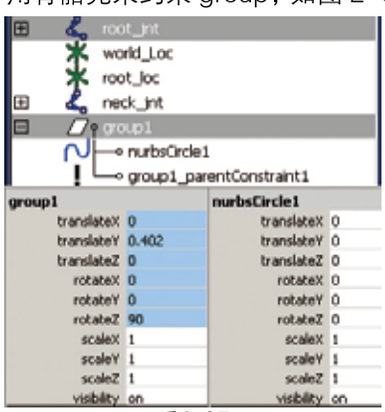


图2-87

3 这是我们需要的结果，控制所有的 transform 都是标准的，而组上面具有和骨骼相同属性的参数，控制器自身轴向的位移和骨骼是完全相同的。

4 在 Rig 工作中，这种创建控制器的工作是非常多的，如果我们每次都这样做工作量是非常大的，如果按照上面的方法需要以下 3 个步骤：

① 为控制器创建组并且命名

② 骨骼来父子约束组

③ 删除约束

从上面的示例，我们不能发现，nurbsCircle1 因为继承了 group 的属性，group 是父级，子级继承了父级的属性，所以子级仍然保持在原来的状态之中，所有的 transform 信息为“零”，如图 2–88 所示。

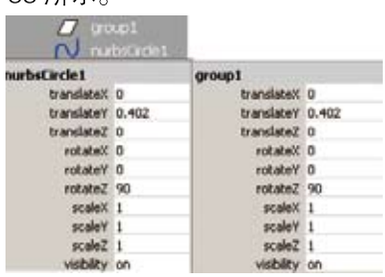


图2-88

而在某些情况下，我们想要骨骼 neck_jnt 的transform 为“零”，而不影响其他相关联的信息，怎么办呢？可以按如下方法进行操作。

1 首先创建一个空的 group，让空组（null1）成为 neck_jnt 的子物体，如图 2–89 所示。

图2-89

2 设置 null1 的 transform 为“零”，如图 2–90 所示。

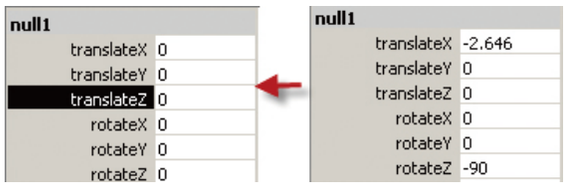


图2-90

3 对 null1 解 P，即让 null1 Parent 到世界坐标系（父级没有任何物体），如图 2–91 所示。

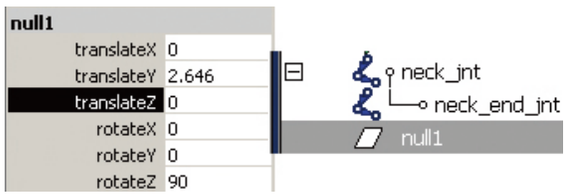


图2-91

4 把 neck_jnt 骨骼 parent 到空组 null1 下面（neck_jnt 成为 null1 的子物体），如图 2–92 所示。

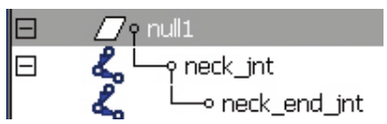


图2-92

5 制作完毕后，neck_jnt 的 transform 信息全部为“零”，完全继承了 null1 的属性，如图 2–93 所示。

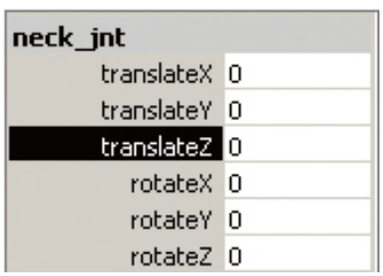


图2-93

由上所知，这样让一个骨骼的 transform 归“零”，过程也是比较繁琐的，为了让我们的制作更加的简单，这里根据如上制作过程创建脚本，逐行讲解一个脚本，它的作用是让拥有位移或者旋转值的物体归“零”。

效果如下：

```
# 导入mayaPython模块
from maya.cmds import *
# 该函数的功能是设置物体的transform为0，缩放属性为1
def lyzero(lyzeroObj):
    # 定义属性元组
    trAttr =('.tx','.ty','.tz','.rx','.ry','.rz')
    sAttr =('.sx','.sy','.sz')
    # 创建循环，并且设置属性值
    for j in range(6):
        setAttr(lyzeroObj+trAttr[j],0)
    for k in range(3):
        setAttr(lyzeroObj+sAttr[k],1)
# 使物体或者控制器，transform信息归“零”
def lyZeroTransform():
    # 获取选择的物体
    zeroObj = ls(sl=True)[0]
    # 创建空组，并进行命名
    grpName = group(em=True,name=zeroObj+'_zeroGrp')
    # parent空组到物体下面
    parent(grpName,zeroObj)
    # 设置物体的transform为0，缩放为1
    lyzero(grpName)
    # 获取物体相关联的parent物体。存在两种情况：物体本身就在世界坐标和物体在另外一个层级下面
    parentObj = listRelatives(zeroObj,p=True)
    # 这里仅仅有两种情况，可以尝试运用try&except进行异常处理
    try:
        parent(grpName,parentObj)
        parent(zeroObj,grpName)
```

```
    except:
        parent(grpName,w=True)
        parent(zeroObj,grpName)
```

以上是整个脚本，我们执行命令：

```
lyZeroTransform()
```

结果如图 2-94 所示。

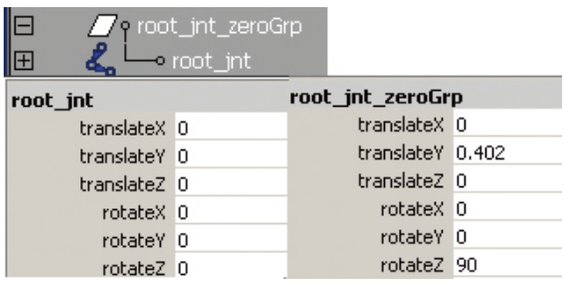


图2-94

2.8.2 follow原理

1 准备如下场景，如图 2-95 所示。

图2-95

2 创建 3 个 locator，创建一个球体，让 3 个 locator 同时约束一个球体，如图 2-96 所示。

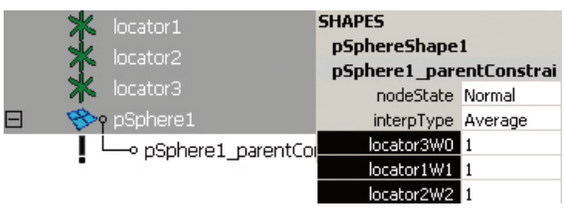


图2-96

3 分别移动 3 个 locator，由于球体受到 3 个 locator 的约束，因此所有的 locator 都影响到球，如图 2-97 所示。

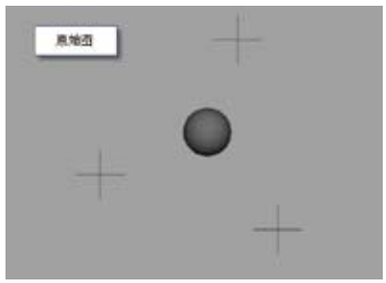


图2-97

4 运用上一节所学的脚本知识，使每个 locator 归“零”，移动查看影响，如图 2-98 所示。

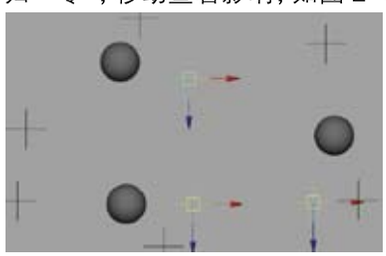


图2-98

5 如果关闭 locator2 和 locator3 对球体的影响，球体就只跟随 locator1 移动，如图 2-99 所示。

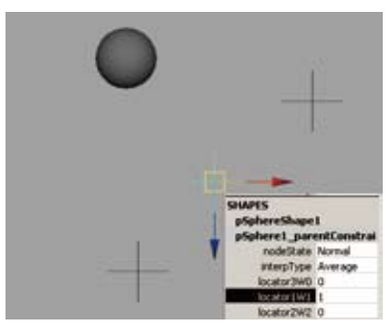


图2-99

6 同理，如果关闭任意其中两个 locator 对球体的影响，球体就只会受到一个 locator 的影响。这样我们开始理解 follow 的原理就比较简单了，让一个物体究竟 follow 哪个物体移动，只需要开启相应的约束而关闭其他的约束就可以了。

2.8.3　follow创建

前两节我们讲解了很多关于 follow 的知识，这一节我们模拟具体的示例，一步一步讲解如果创建 follow。

1 创建模拟场景后，创建 box 并将其命名为 root_geo，模拟身体部分。创建两个骨骼，并用 root_jnt 对 root_geo 蒙皮。创建 world_loc，确保 world_loc 的坐标为世界坐标，如图 2-100 所示。

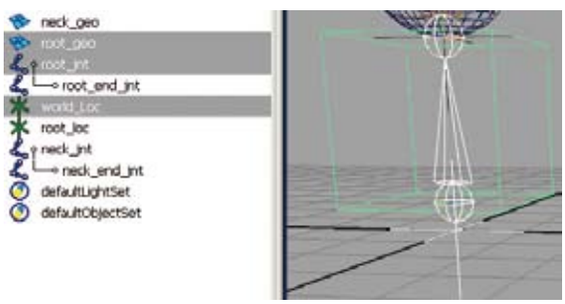


图2-100

2 创建 pSphere 并命名为 neck_geo，模拟脖子部分。创建两个骨骼，并用 neck_jnt 对 neck_geo 蒙皮。创建 root_loc，确保 root_loc 的坐标为 neck_jnt，如图 2-101 所示。

图2-101

3 接下来开始制作。选择 neck_jnt，使其属性归“零”，如图 2-102 所示。

图2-102

4 选择 world_loc、root_loc 和 neck_jnt_zeroGrp1，进行 OrientConstraint，OrientConstraint 设置如图 2-103 所示。

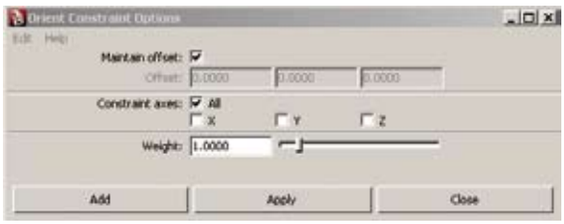


图2-103

5 选择 neck_jnt，为其添加属性 follow，如图 2-104 所示。

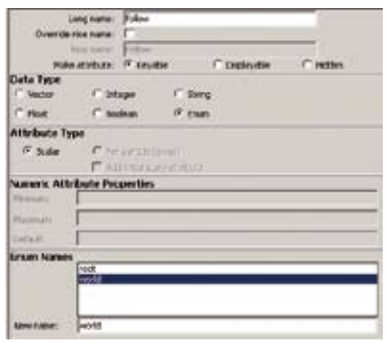


图2-104

6 打开属性连接面板，连接 neck_jnt.follow 到 neck_jnt_zeroGrp1_orientConstraint1.world_LocW0 上，如图 2-105 所示。

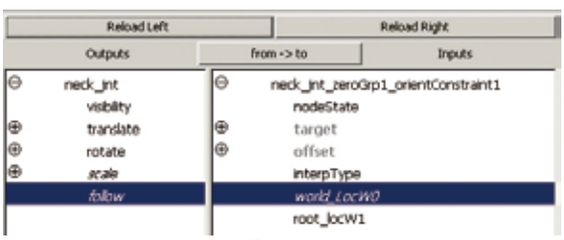


图2-105

7 创建 reverse 节点，命名为 neck_follow_reverse，并连接 neck_jnt.follow 到其 inputX，如图 2–106 所示。

图2-106

8 输出 neck_follow_reverse 的信息到 neck_jnt_zeroGrp1_orientConstraint1.root_lcoW1 上，如图 2–107 所示。

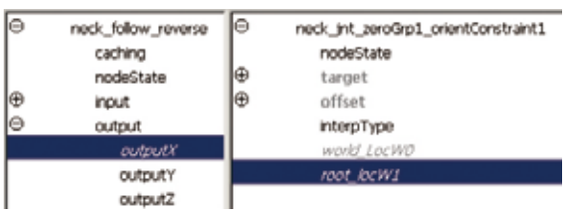


图2-107

9 分别调整 follow 的属性，因为创建了 reverse 节点，根据其原理，neck_jnt_zeroGrp1 将受到不同 locator 的影响，如图 2–108 所示。

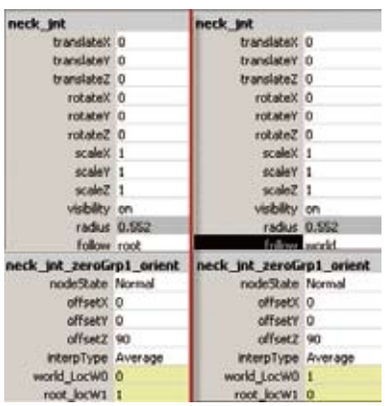


图2-108

为了更好的查看效果，我们继续下面的制作。选择 root_end_jnt 对 root_loc 进行父子约束；选择 root_loc 对 neck_jnt_zeroGrp1 进行点约束。

测试效果

（1）设置 follow 的属性为 world，旋转 root_jnt、neck_geo，始终是不跟随 root_jnt 旋转的。

（2）设置 follow 的属性为 root，旋转 root_jnt、neck_geo，始终是跟随 root_jnt 旋转的。

前后对比效果如图 2–109 所示。

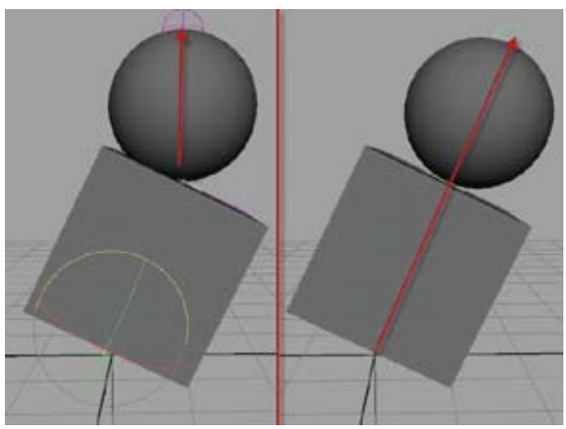


图2-109

脖子的设置需要非常精细的过程，这里我们为脖子添加一种比较复杂的设置，包括伸缩控制、FK 效果、follow 效果等，如图 2–110 所示。

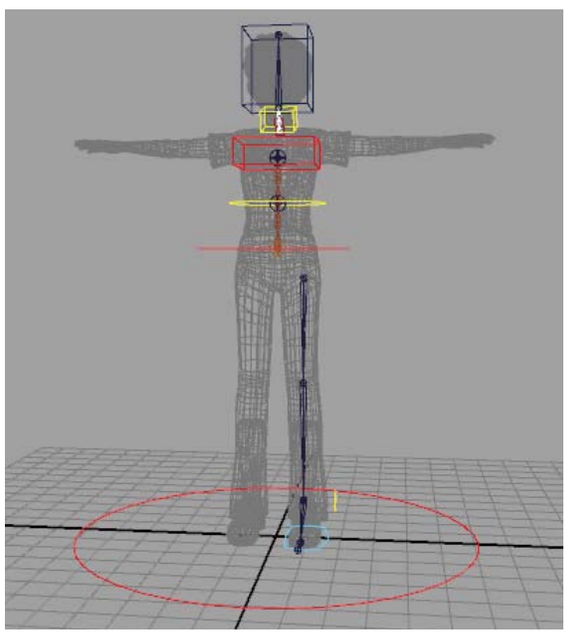


图2-110

2.8.4　创建线性IK

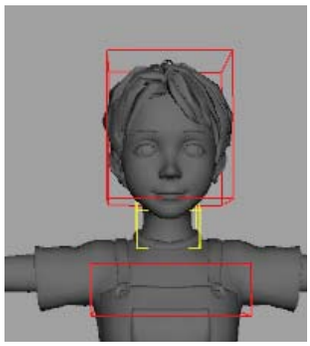


图2-111

1 创建 neck 控制器，将其命名为 neck_anim；创建头部控制器，命名为 head_anim，如图 2–111 所示。

2 依据创建 torso 线性 IK 的方法，为脖子创建线性 IK，命名如图 2–112 所示。

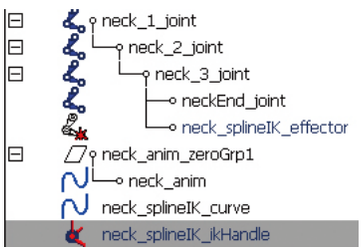


图2-112

3 为线性 IK 创建伸缩。复制曲线 neck_splineIK_curve，并且命名为 neck_splineIK_scale_curve。为每条曲线创建长度节点，分别命名为 neck_curveInfo 和 neck_scale_curveInfo，如图 2–113 所示。

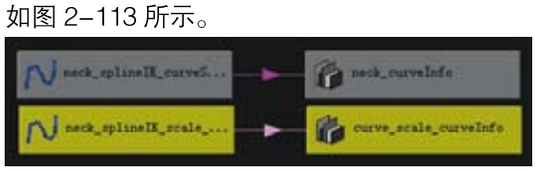


图2-113

4 创建乘除节点，将其命名为 neck_MD，连接设置如图 2–114 和图 2–115 所示。

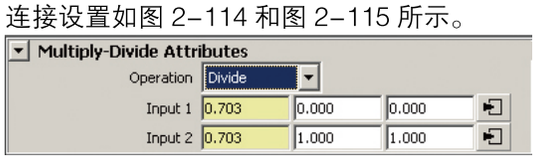


图2-114

图2-115

5 创建乘除节点，命名为 neck_joint_trans_MD，设置连接如图 2–116 和图 2–117 所示。

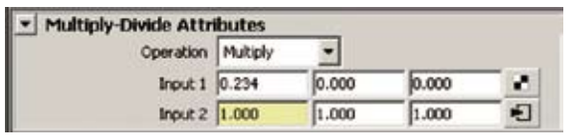


图2-116

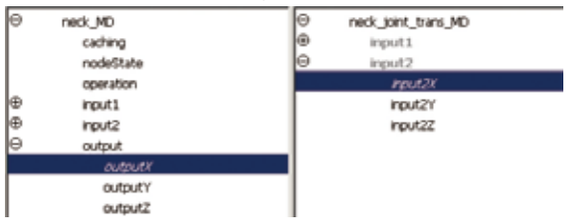


图2-117

6 输出 neck_joint_trans_MD 的属性到 neck_joint 的 translateX 数值。

```
connectAttr -f neck_joint_trans_MD.outputX
neck_2_joint.translateX;
connectAttr -f neck_joint_trans_MD.outputX
neck_3_joint.translateX;
connectAttr -f neck_joint_trans_MD.outputX
neckEnd_joint.translateX;
```

7 创建骨骼 neck_fk_skin、neck_fk_1_skin 和 neck_fk_2_skin，对线性 IK 的曲线蒙皮，精细的调节每个点的权重值，使整个运动流畅，如图 2–118、图 2–119 和图 2–120 所示。

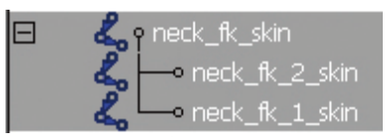


图2-118

图2-119

	neck_fk_1_skin	neck_fk_2_skin	neck_fk_skin	Total
Hold	off	off	on	
neck_splineIK_				
cv[0]	0.000	0.000	1.000	1.000
cv[1]	0.250	0.250	0.500	1.000
cv[2]	0.500	0.500	0.000	1.000
cv[3]	0.000	1.000	0.000	1.000

图2-120

8 为线性 IK 创建高级旋转，设置如图 2–121 所示。

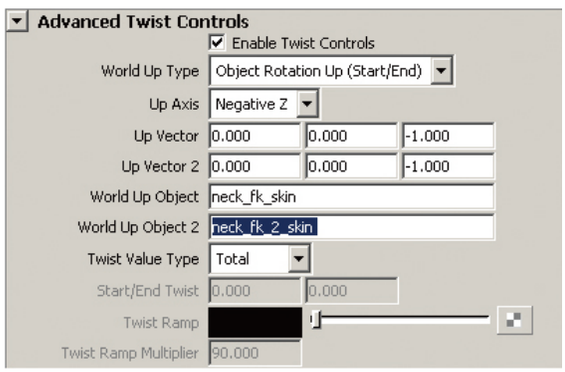


图2-121

9 整理 group，使整个设置更加工整，如图 2–122 和图 2–123 所示。

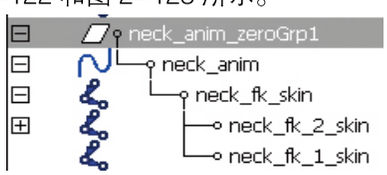


图2-122

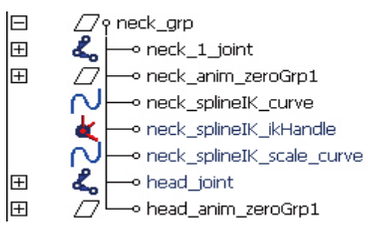


图2-123

10 关闭曲线的 inherits transform，避免双倍位移，如图 2–124 所示。

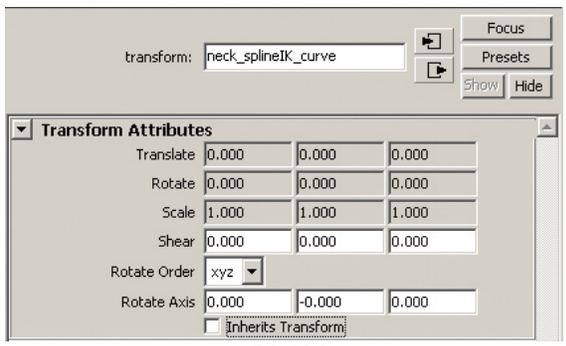


图2-124

2.8.5 follow设置

1 为 neck_1_joint 和 neck_anim_zeroGrp1 创建 group，命名为 neck_ctrl_grp，轴心与其相同，如图 2–125 所示。

图2-125

2 创建 locator，命名为 neck_local_loc，轴心与 shoulder_anim 相同，使用 shoulder_anim 父子约束约束 neck_local_loc。使用 neck_local_loc 点约束 neck_ctrl_grp，如图 2–126 所示。

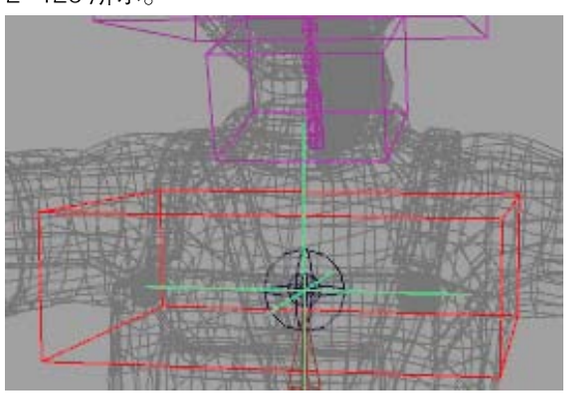


图2-126

3 创建 locator，命名为 neck_world_loc，如图 2–127 所示。

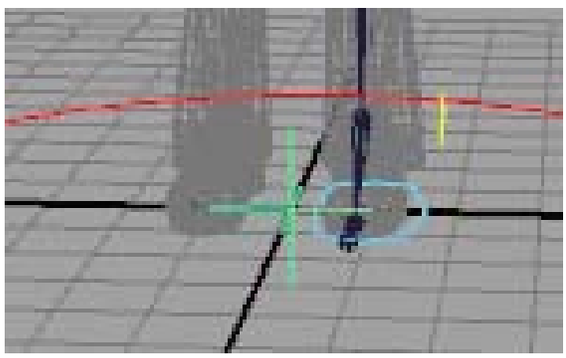


图2-127

4 选择 neck_world_loc 和 neck_local_loc，为 neck_ctrl_grp 添加旋转约束，如图 2–128 所示。

图2-128

5 为 neck_anim 添加 enum 属性 follow，如图 2–129 所示。

图2-129

6 连接 follow 属性到约束节点，如图 2–130 所示。

图2-130

7 创建 reverse 节点，命名为 neck_follow_reverse，连接如图 2–131 和图 2–132 所示。

图2-131

图2-132

脖子的设置已经完毕，大家注意一些细节的东西：

第一，是 follow 设置。这点不再多说，我们已经花了很多篇幅为大家其原理和制作了。

第二，脖子的线性 IK 制作。如图 2–133 所示参加蒙皮的骨骼是这样 group 的，旋转头部控制器时，线性 IK 的骨骼是跟着一起旋转和扭曲的。权重只能实现简单的拉伸和弯曲，而使用线性 IK 的制作效果会很好，整个过渡会非常的自然，如图 2–134 所示。

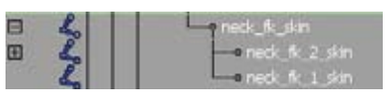


图2-133

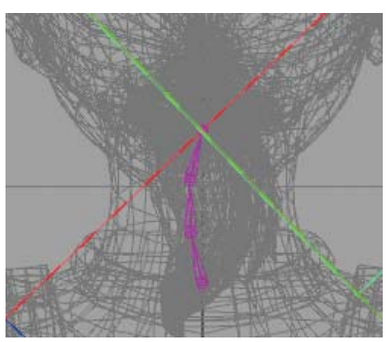


图2-134