TD Matt

Maya character rigging for games, scripting, Unreal Engine.

Wednesday, 12 January 2011

Spine control rig

Happy New Year! Sorry it's taken a while to get the first post of 2011 out of the door. I started writing this ages ago but work, family and post holiday blues have conspired against me. Anyway...

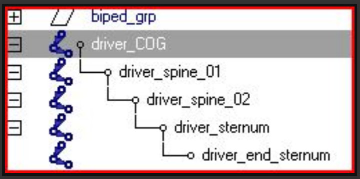
In this post I'm going to cover how to set up a spine control system. This does include squash & stretch functionality, but in the interests of getting this out I'm going to cover how to add this to it in the next post.

(Update) You can find this here

The core of it consists of a NURBS surface with ik joints controlled via groups attached to the surface using **pointOnSurfaceInfo** nodes. The surface is deformed by clusters which in turn are driven by controllers. If you're not familiar with uv (texture) coordinates it's worth reading up on this first, I'm not going to go into it here as there is plenty of information kicking around on the subject.

Create driver joints

First off, **duplicate** the spine joints out and prefix them with the string *driver_*. If there is no end joint for the sternum, create one as we'll need it for an ik handle later.



Controls

We need some controllers, four in all - *COG_ctrl, hips_ctrl, torso_ctrl* and *gut_ctrl*. Make sure you set your rotation orders appropriately. In the case of the *hips_ctrl* and *COG_ctrl,* make absolutely sure that it matches the rotation order of the joint *driver_COG.*
Place *COG_ctrl* and *hips_ctrl* so their pivot points are at the position of *driver_COG*, *torso_ctrl* at *driver_sternum* and *gut_ctrl* mid way between the two.

About Me

Matt

Senior Technical Artist for Ninja Theory Ltd.

View my complete profile

Links

- Keith Lango
- Chris Goodall - animator @ NT
- Jason Schleifer
- Animation Nourishment
- Rigging 101 - a good starting point
- Ninja Theory
- JB - Character artist @ NT
- tech-artists.org

Blog Archive

▼ 2011 (7)
► May (2)
► February (2)
▼ January (3)
Adding squash & stretch to the spine control rig
Hey, I thought I'd do a quick post just to say fe...
Spine control rig
► 2010 (18)

Search This Blog

Search

Total Pageviews

239980

Subscribe To

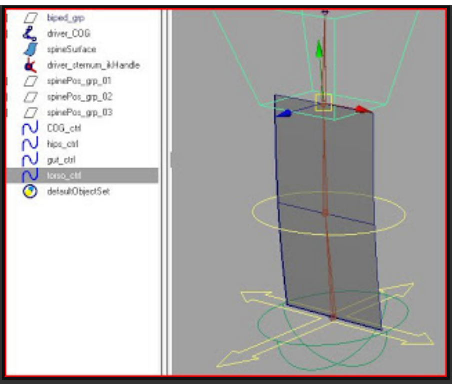
Posts

Comments

Followers

Abonnés (94) Suiv.

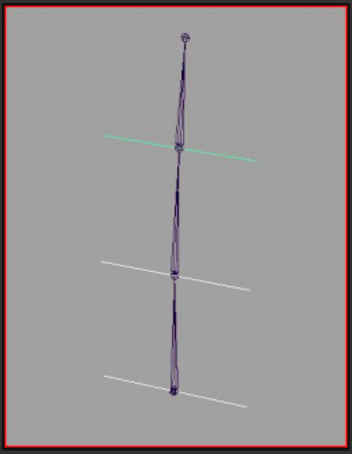
S'abonner



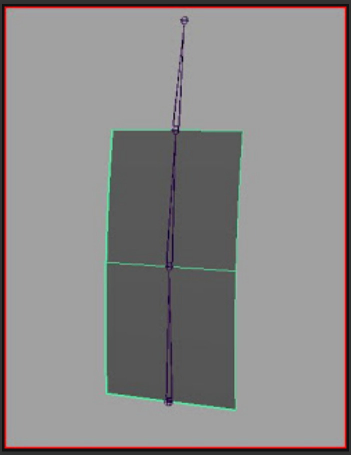
Group each controller and freeze translate, rotate and scale attributes so that your controls are at zero in this position. Also make sure the rotation orders of the groups match the controller.

Control surface

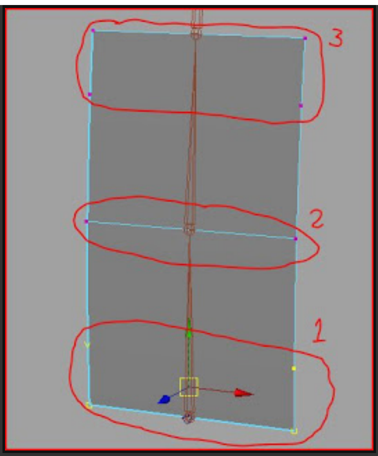
Create a CV curve (linear curve degree is fine), centre its pivot and snap it to the base of the spine. **Duplicate** it twice more, snapping one to *driver_spine_02* and the other to *driver_sternum*.



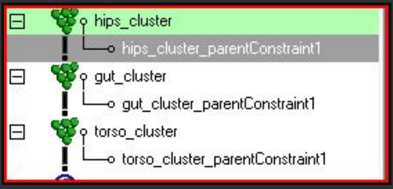
Starting at the base, select each curve in order and create a lofted NURBS surface. This is the underlying structure that we will use to control the joints. Rename it to *spineSurface* and delete history on the surface - unnecessary history stacks on nodes will impact rig performance. You can delete the curves.



We need to create three clusters to deform the surface. This shows which CVs should be influenced by each cluster:



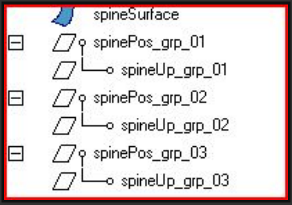
From the bottom up, call them *hips_cluster, gut_cluster* and *torso_cluster.* **Parent constrain** each cluster to the equivalent controller.



hips_cluster
hips_cluster_parentConstraint1
gut_cluster
gut_cluster_parentConstraint1
torso_cluster
torso_cluster_parentConstraint1

Attaching groups to the surface

We need to create the group structure that follows the surface as it deforms. These groups will eventually drive ikHandles on the joints.
Create a group called *spinePos_grp_01* with another group as a child of this called *spineUp_grp_01*. **Duplicate** these twice more so you have three pairs of groups.



spineSurface
spinePos_grp_01
spineUp_grp_01
spinePos_grp_02
spineUp_grp_02
spinePos_grp_03
spineUp_grp_03

To connect these to the the surface we will use a **pointOnSurfaceInfo** node. From Maya documentation:

Compute information associated with a point on a nurbs surface. The point is specified with the input surface (inputSurface) and the parameter value (parameterU and parameterV), either in the surface domain or in the 0-1 domain (if turnOnPercentage is set to true.) The information available is: position, surface normal (normalized to length 1 in normalizedNormal), surface tangents and their normalized counterparts.

So basically, we associate the node with our NURBS surface and give it a U and V value to position it on the surface. We can query the node to give us the position in world-space coordinates of that point on surface, as well as the U and V tangents of the surface at that point. These then drive world space positions of our groups.

The easiest way to create and connect this up is using a small amount of mel script.

```
// First, create the pointOnSurfaceInfo node:
createNode -n "spine_01_PoSI" pointOnSurfaceInfo;

// Connect the NURBS surface to it:
connectAttr spineSurface.worldSpace spine_01_PoSI.inputSurface;

// Connect the position attribute to the translate attribute of our first group:
connectAttr spine_01_PoSI.position spinePos_grp_01.translate;

// Connect the U tangent attribute to the child of our first group:
connectAttr spine_01_PoSI.tangentU spineUp_grp_01.translate;
```

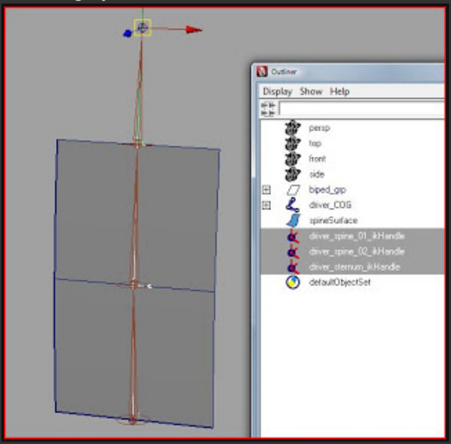
Now, if you select *spinePosGrp_01* you should see it is connected to the surface at the bottom left, i.e a 0,0 UV value on the surface. The group *spineUpGrp_01* should be some way off to the right, and is basically a vector from this point that represents the tangent of the surface in U. To place it where we need, i.e at the base of the spine, set *spine_01_PoSI.parameterU* to **0.5**;

We need to do this for the remaining two pairs of groups, each with its own pointOnSurfaceInfo node. For the second group set *spine_02_PoSI.parameterU* to **1** and for the third set *spine_01_PoSI.parameterU* to **2**.
Here is a quick capture to demonstrate things so far. I've placed some temporary locators to visualise the position of the groups so you can see what should be happening. The yellow locators are being driven by the **position** attribute of the **pointOnSurfaceInfo** node, the red locators by the **tangentU** attribute. As the surface is deformed the locator positions move around

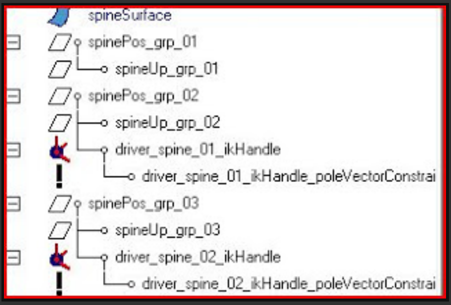
giving us a point and a vector in world space, we will use these positions to drive ikHandles for the spine joints.

ikHandles driven by the surface-attached groups

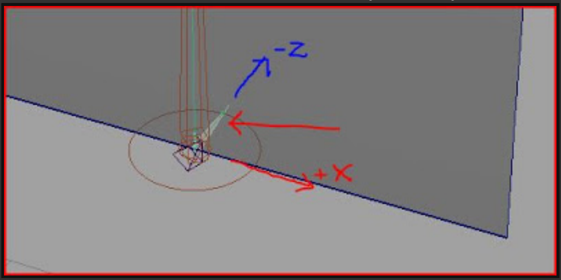
The joints themselves are controlled by ikHandles. This enables us to use the ikBlend attribute to switch between IK and FK control methods.
Create an **ikRP handle** from *driver_spine_01* to *driver_spine_02*, *driver_spine_02* to *driver_sternum* and *driver_sternum* to *end_driver_sternum*. So, you should have three ik handles in total, each for a single joint.



To drive the ikHandle positions, parent *driver_spine_01_ikHandle* to *spinePosGrp_02*, and *driver_spine_02_ikHandle* to *spinePosGrp_03*.



Now we use the ..UpGrps to drive the pole vectors. First off, you'll need to take note of the current pole vector of the ikHandle. Often the default pole vector Maya gives you isn't pointing in the direction we want. In this example the default vector is pointing down negative Z, and by setting up a **poleVectorConstraint** to *spineUpGrp_01* which is in posiveX we will effectively be placing a 90 degree rotation on it that we don't want. This is dead easy to fix as you will see in just a sec.



Create a **poleVectorConstraint** from *spineUpGrp_02* to *driver_spine_01_ikHandle*. You should see the pole vector snap to positive X, giving a rotation of -90 on rY. To fix this just set *driver_spine_01_ikHandle.twist* to **90** and you'll see the pole vector snap back to its default position. To double check everything is as it should be, make sure the rotation values for *driver_spine_01* are at 0,0,0 after creating the **poleVectorConstraint**.

Parent *driver_spine_02_ikHandle* to *spinePosGrp_03* and **poleVectorConstrain** *driver_spine_02_ikHandle* to *spineUpGrp_03*. Double check your pole vectors as above, making sure that once the constraint is set up you have zero rotation values on *driver_spine_02*.

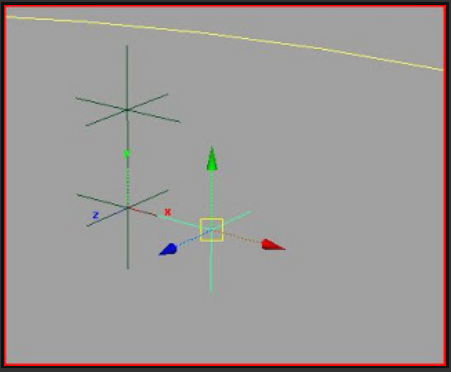
Parent *driver_sternum_ikHandle* to *torso_ctrl.*

Setting up inter-control constraints

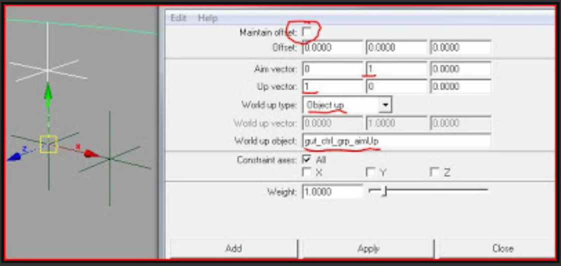
Parent constrain *torso_ctrl_grp* and *hips_ctrl_grp* to *COG_ctrl.*
Parent constrain the joint *driver_COG* to *hips_ctrl.*

So, just the constraint for *gut_ctrl_grp* to set up... The quick way is to parent constrain it to both *hips_ctrl* and *torso_ctrl* but I have found this can be extremely unstable causing *gut_ctrl_grp* to flip. The following method allows us to create a stable parent constraint between two nodes. It consists of three locators that drive the group position and orientation via a **point** and **aim constraint.** These three locators are parent constrained to both *hips_ctrl* and *torso_ctrl.*

Display local rotation axis for *gut_ctrl_grp* to help us set up **aim constraint** correctly.
Create three locators called *gut_ctrl_grp_point, gut_ctrl_grp_aimUp* and *gut_ctrl_grp_aimAt.*
Parent snap each locator to *gut_ctrl_grp* (explanation of custom tool to parent snap). Using the rotation axis of the group, align *gut_ctrl_grp_aimUp* so that the X axis of the group points towards it. Do the same for *gut_ctrl_grp_aimAt* in the Y axis. You can see what it should look like below, with the group rotation axis visible and the locators aligned to it ready for setting up the **aim constraint**:



Point constrain *gut_ctrl_grp* to *gut_ctrl_grp_point.* Create an **aim constraint** from *gut_ctrl_grp_aimAt* to *gut_ctrl_grp* with these settings (assuming you've set up your group and locators in the same way):



Maintain offset: ☐
Offset: 0.0000 0.0000 0.0000
Aim vector: 0 1 0.0000
Up vector: 1 0 0.0000
World up type: Object up
World up vector: 0.0000 1.0000 0.0000
World up object: gut_ctrl_grp_aimUp
Constraint axes: ☑ All
☐ X ☐ Y ☐ Z
Weight: 1.0000

If you've got your locators positioned correctly and constraint settings entered correctly the group should not change orientation at all when the constraint is created. If it does, check your locator positions and constrain options and try again.

Lastly, **parent constraint** each locator to both *torso_ctrl* and *hips_ctrl*. The gut controller should now follow an intermediate point and orientation between hips and torso but be much more stable that a straight parent constraint between controllers.

At the top of the post is a capture of this spine setup working on a fully rigged character. I'll be covering how to add squash & stretch and also an fk/ik switch in the next post.

Posted by Matt at 15:16

Labels: control, ribbon spine, rig, spine

11 comments:

Josh 11 February 2011 at 14:13

Brilliant stuff, Matt. Tested it out and really like it. I'm gonna play with using a variation of this for twisty limb segments. Had a question for you - is there a benefit to using clusters to control the surface cv's rather than a separate set of control joints they could be skinned to? Does that