# Constraints & Dynamic Parenting in Maya

By Eric Keller (www.bloopatone.com)

Dynamic Parenting refers to a situation where the inheritance of an object's transform channels can be turned on and off, either by keyframes or other means, during the course of an animation. Commonly this is used to allow an animated character to pick up and put down objects. It is also very useful when controlling the binding and interaction of two molecular structures in a scientific animation. In Maya, the easiest way to achieve dynamic parenting is through the use of constraints.

This lesson will go through several variations of dynamic parenting situations as a means to inspire you to develop your own creative solutions to the problem. As always, there are a million ways to achieve something in Maya, and only 500,000 of those ways are a bad idea.

Point Constraint: One or more other objects control any or all of one object's translation channels.

Orient Constraint: One or more other objects control any or all of one object's rotation channels.

Parent Constraint: One or more other objects control any or all of one object's translation and rotation (but not scale) channels.

Note that the Maintain Offset option will allow you to maintain the controlled object's current position at the time these constraints are applied.

Other constraints include Geometry, Normal, Tangent, Pole Vector, Aim, and Scale.

Example files:

The models were imported by Dr. Gael McGill based on rcsb.org file 2bex.pdb (Placental ribonuclease inhibitor in complex with human eosinophil derived neurotoxin)

dynamicParent1.mb, dynamicParent2.mb:

The two structures have been imported, separated, grouped and shaded. If the inhibitor is keyframed binding to the ribo structure we're fine, until we decide we want to animate the ribo structure. If we parent the inhibitor to the ribo structure and then animate ribo we get a very strange movement.

The preferred method is to use constraints. A good workflow in this situation would be this:

1. Create two locators named “bound” and “unbound”

2. Bound should be at the pivot point of the inhibitor while the inhibitor is in its bound position; this ensures that the inhibitor ends up in the right place when it is animated. A quick trick to this is to point constrain the locator to the inhibitor so that it snaps to the pivot point of the inhibitor, and then delete the constraint. (delete `pointConstraint -offset 0 0 0 -weight 1 constrainingObject constrainedObject`;)

3. Bound can then be parented to the ribo group.

4. Then the inhibitor is constrained to the unbound locator (select the constraining object first, then shift-select the object to be constrained). You can use a point and an orientation constraint or a parent constraint. I prefer the former. Make sure that “maintain offset” is unchecked in the options.

5. The inhibitor is then constrained in the same way, to the bound locator.

6. The inhibitor should end up equidistant from the two locators.

7. The inhibitor can then be animated binding to ribo by setting keyframes on the constraint’s weight channel.

8. This is demonstrated in dynamicParent2.mb

9. Keyframes can be set on the locators and the weight values to create the look of the structures binding when they get with the desired distance.
Note: the rotation on the bound object is correct because of the combination of the point and orient constraints.

dynamicParent3.mb:

Adding a fractal noise texture to the rotation of the structures can create a more interesting animation.

1. Starting with the same set up as dynamicParent2 except without rotational constraints on any of the objects.

2. Create a locator and name it “riboShaker”. Create a locator and name it “inhibitorShaker”.

3. inhibitorShaker is point-constrained to the inhibitor.

4. riboShaker is point-constrained to ribo.

5. Animated fractal textures are added to each rotational channel of the shaker constraints.

6. The place2Dtexture nodes can be rotated to mix up the movement a bit.

7. If this is done a lot, a MEL script could make set-up less tedious (see one possible example at the end of the notes shakerMake3.mel).

8. Ribo is orient-Constrained to riboShaker.

9. Inhibitor in orient-constrained to both riboShaker and inhibitorShaker. The weights on the orient constraint can be animated, or a connection can be set-up between the point constraint weight keyframes and the orient constraint weight keyframes.

dynamicParent4.mb:

More interesting motion can be created if the translational locators (bound and unbound) are placed on motion paths. These paths can then be converted to soft-body curves or dynamic curves (using the hair system in Maya Unlimited).

dynmaicParent5.mb

The locators can be constrained to single particles using the worldCentroid attribute of the particle. One particle can be made the goal of the other so that eventually they meet. The constraint weight can then be controlled by a distanceBetween/condition node set up so that the structures bind correctly when the particles are within a certain distance.

Baking the simulation into keyframes is usually a good idea once the animation is looking good. Keyframes can be adjusted easily to correct for mistakes like interpenetration.

shakerMake3.mel

```
//put selected objects into array
string $myObject[] = `ls - sl`;

//start loop based on size of selected object array
for ($k=0;$k<size($myObject);$k++){
	select $myObject[$k];
	//loop 3 times to create fractal textures for each rotational channel
	for($i=0;$i<3;$i++){
		//create random number varaiable to randomize the rotation of placement nodes
		float $rand=rand(360);
		//create fractal texture
		shadingNode -asTexture fractal -n ($myObject[$k]+"shaker"+$i);
		//create placement node
		shadingNode -asUtility place2dTexture -n ($myObject[$k]+"shakerUV"+$i);
		//connect fractal to placement node
		connectAttr ($myObject[$k]+"shakerUV"+$i+".outUV") ($myObject[$k]+"shaker"+$i+".uv");
		//connect filter size - this happens whenever you create a fractal texture node
		connectAttr ($myObject[$k]+"shakerUV"+$i+".outUvFilterSize")
($myObject[$k]+"shaker"+$i+".uvFilterSize");

		//randomize the rotation on the placement node
		setAttr ($myObject[$k]+"shakerUV"+$i+".rotateFrame") $rand;
		//make fractal animated
		setAttr ($myObject[$k]+"shaker"+$i+".animated") 1;
		//set current frame at 1
		currentTime 1;
		//set time on animated fractal
		setAttr ($myObject[$k]+"shaker"+$i+".time") 0;
		//create keyframe on fractal time
		setKeyframe ($myObject[$k]+"shaker"+$i+".time");
		//move time to frame 25
		currentTime 25;
		//set fractal time to 100
		setAttr ($myObject[$k]+"shaker"+$i+".time") 100;
		//create keyframe on fractal time
		setKeyframe ($myObject[$k]+"shaker"+$i+".time");
		//set post infinity to linear
		setAttr ($myObject[$k]+"shaker"+$i+"_time.postInfinity") 1;
		//set amplitude to 60
		setAttr ($myObject[$k]+"shaker"+$i+".amplitude") 60;
		//create if statement to attach each of the three textures to each rotational channel on the
current object
		if ($i==0){
			connectAttr -f ($myObject[$k]+"shaker"+$i+".outAlpha") ($myObject[$k] +
".rotateX");
		}
		else if ($i==1){
			connectAttr -f ($myObject[$k]+"shaker"+$i+".outAlpha") ($myObject[$k] +
".rotateY");
		}
		else {
			connectAttr -f ($myObject[$k]+"shaker"+$i+".outAlpha") ($myObject[$k] +
".rotateZ");
		}
	}
}
```