# Xfrog 5

for MAYA

Manual 2

Xfrog inc

# Xfrog 5

## Individual access and customization at single instance level

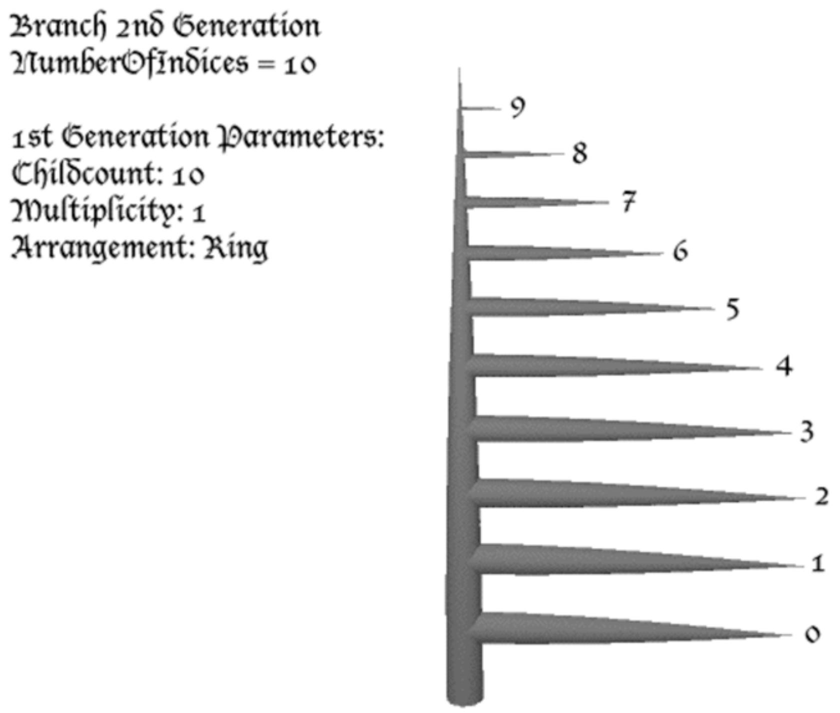
1. Access to single instance elements by offset
2. Remove single instance elements from whole objects
3. Individually access parameters of single instance element
4. Higher flexibility in combination of elements
5. Access locations of single instance elements
6. Better usability

## 1. Access to single instance elements by instance-number

- Each multiplying object generates a number of links, these are:
  - Branch-child-links
    **(number of parent links) * childcount * multiplicity**
    *multiplicity = 2 for arrangement = pair-mirror, otherwise as set*
  - Branch-shape-links
    **(number of parent links) * (linear spline point-count)**
  - Branch-head-links
    **(number of parent links)**
  - Hydra-child-links
    **(number of parent links) * childcount**
  - Phyllotaxis-child-links
    **(number of parent links) * childcount**
- Within the generated number of links,
  each link keeps ist place relative to its parent link
- Each link gets its own index called offset,
  the offset is calculated as follows
  **offset = (parent offset) * childcount [ * multiplicity] + (local iteration)**

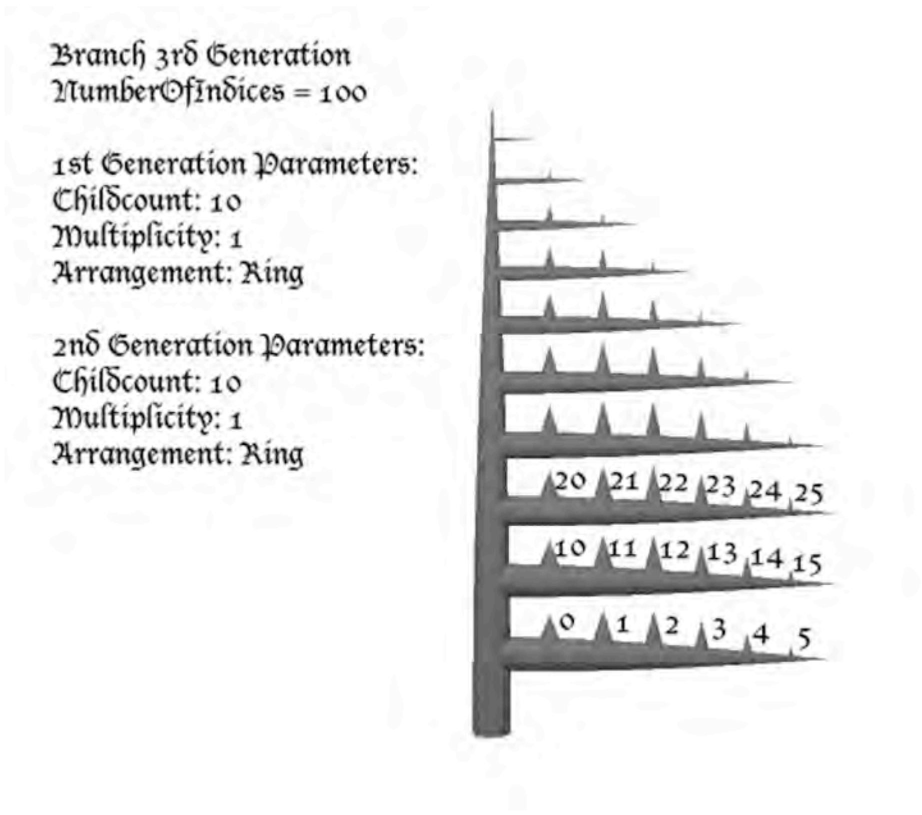
*The following pages show examples for offset numbering.*

## 1. Access to single instance elements by instance-number



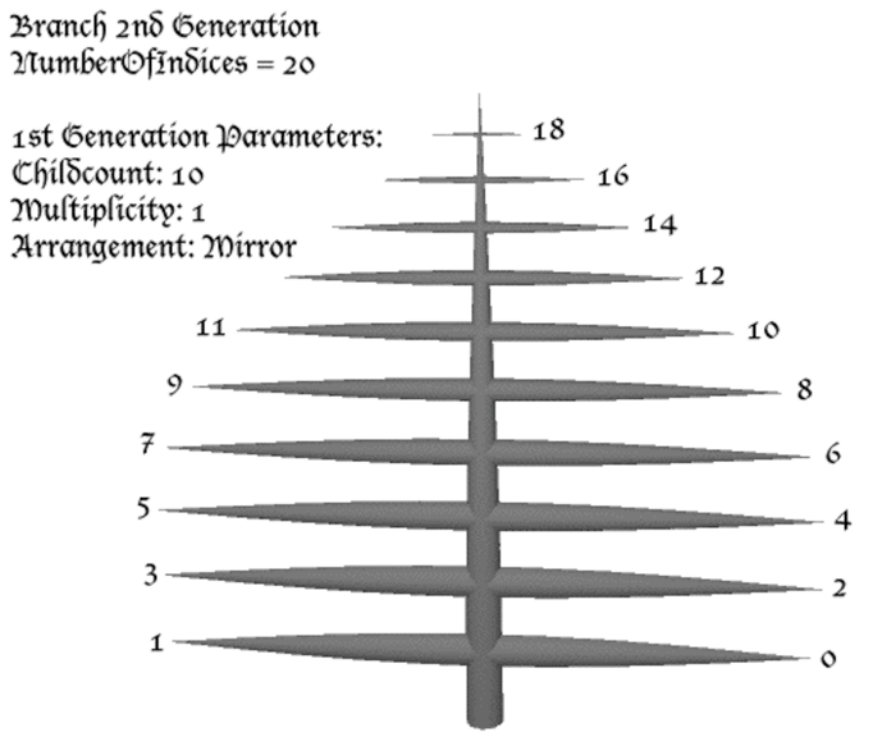
*Branch as child of standard Branch*

1. Access to single instance elements by instance-number



*Standard Branch in higher generation*

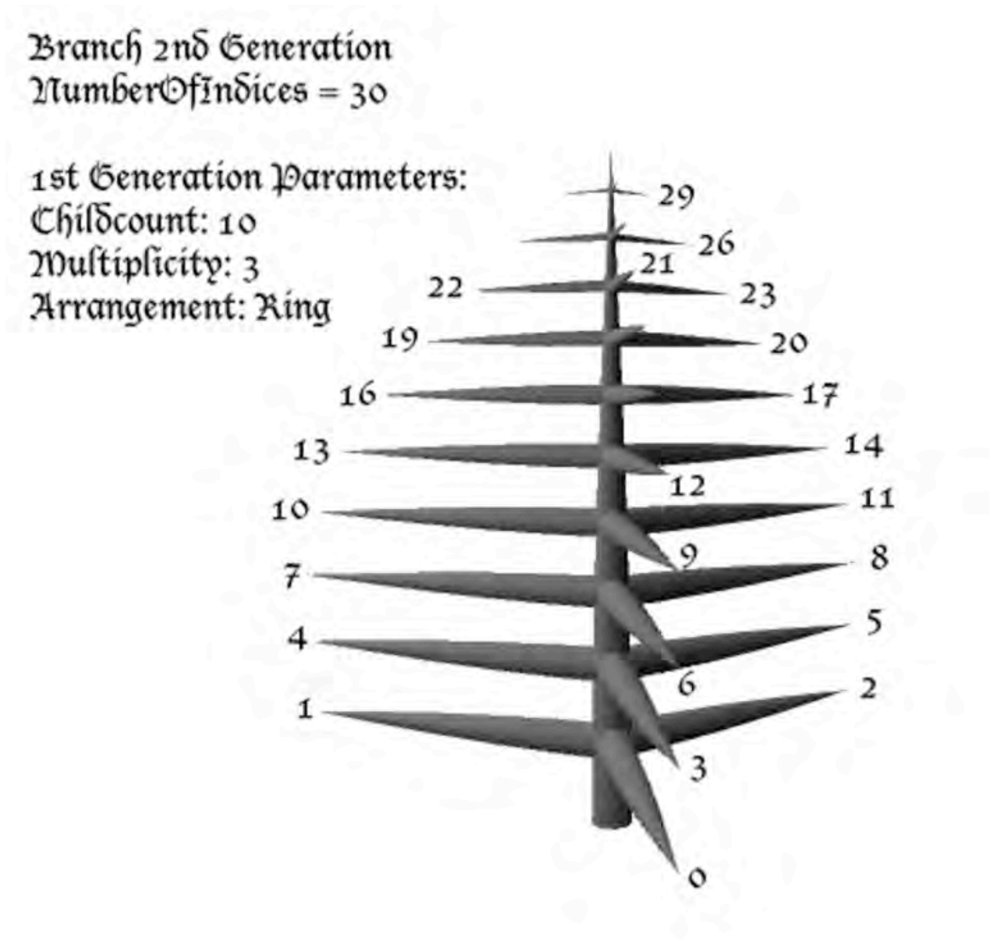
## 1. Access to single instance elements by instance-number



*Standard Branch as child of Branch with arrangement Pair-Mirror*

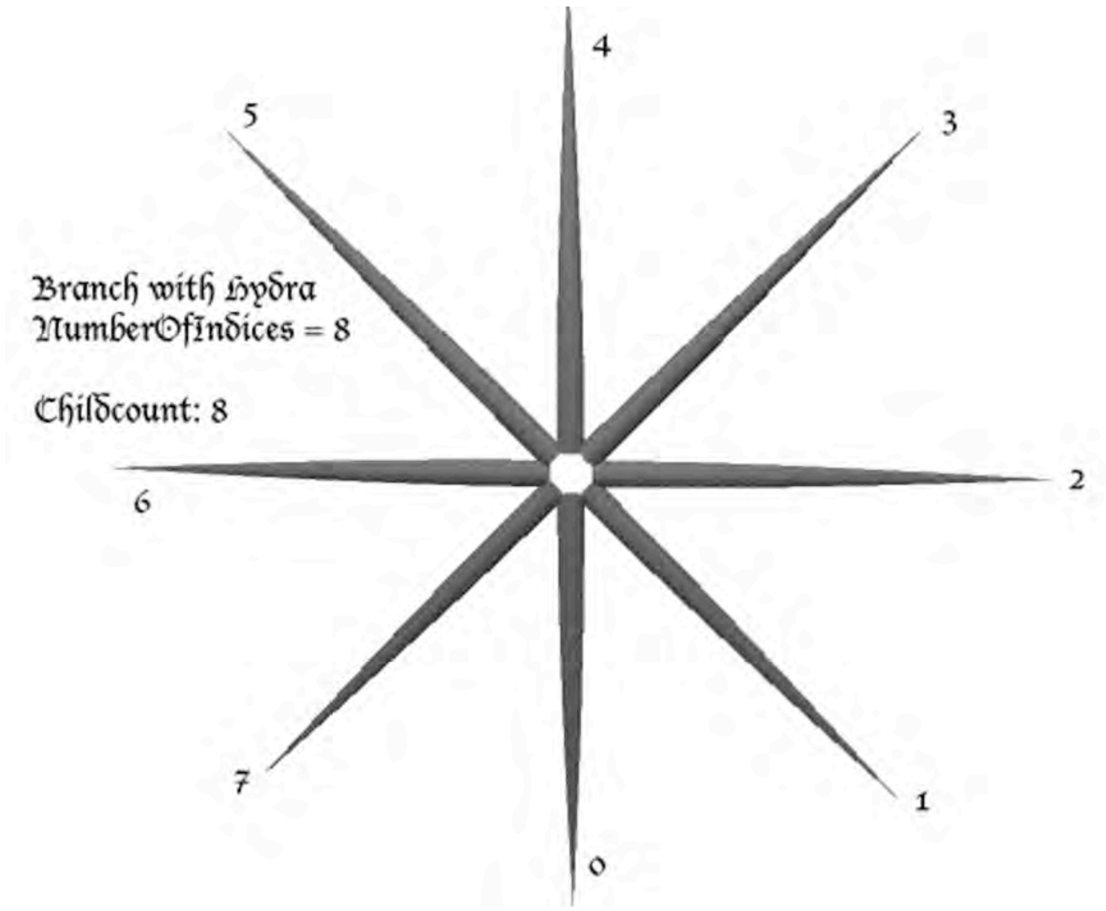
*Set multiplicity=1, because of Pair-Mirror multiplicity = 2*

## 1. Access to single instance elements by instance-number



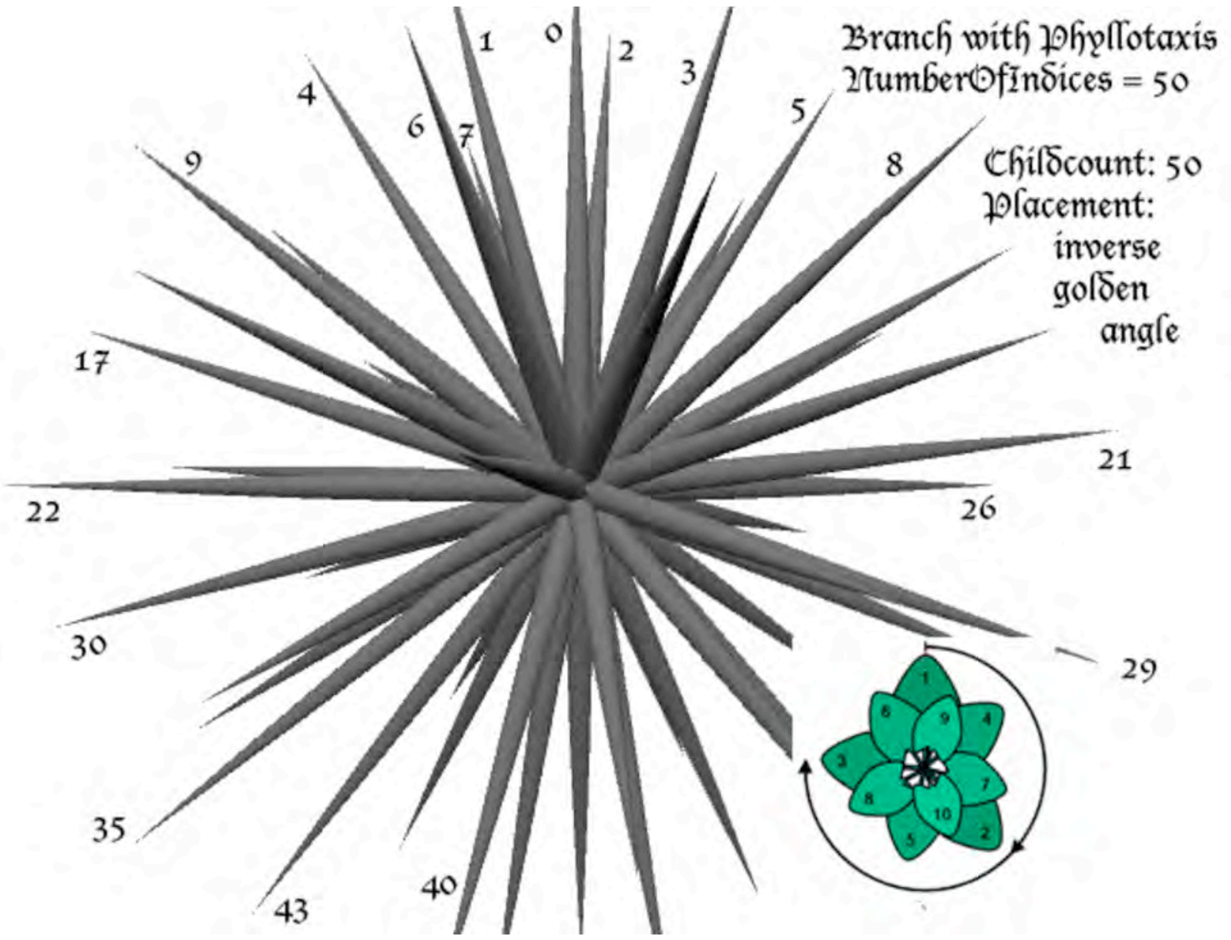
*Standard Branch as child of Branch with multiplicity = 3*

## 1. Access to single instance elements by instance-number



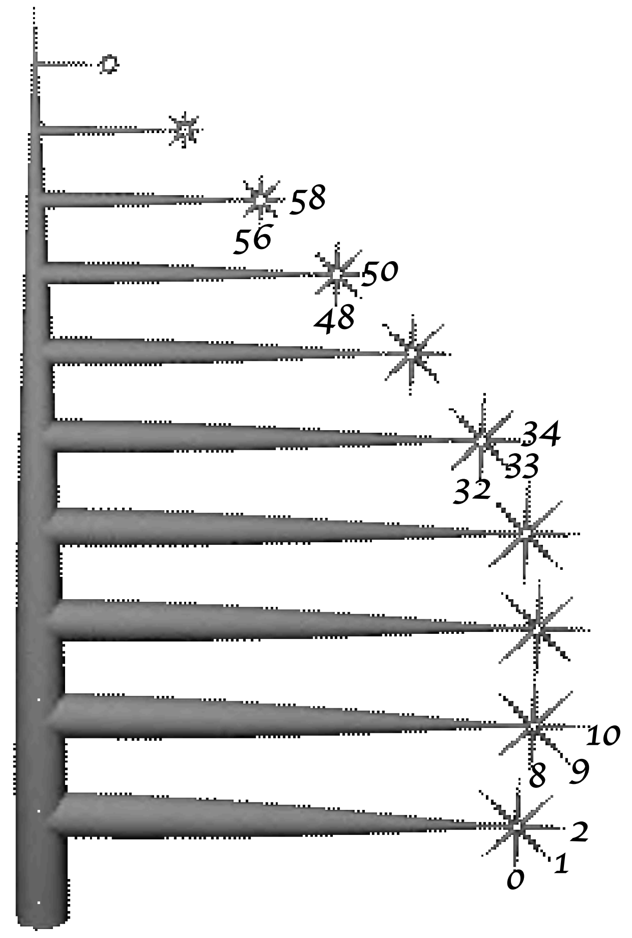
*Standard Branch as child of Hydra*

## 1. Access to single instance elements by instance-number



*Standard Branch as child of Phyllotaxis*

## 1. Access to single instance elements by instance-number



*Higher level combination Branch (10) - Branch (10) - Hydra (8) - Branch*

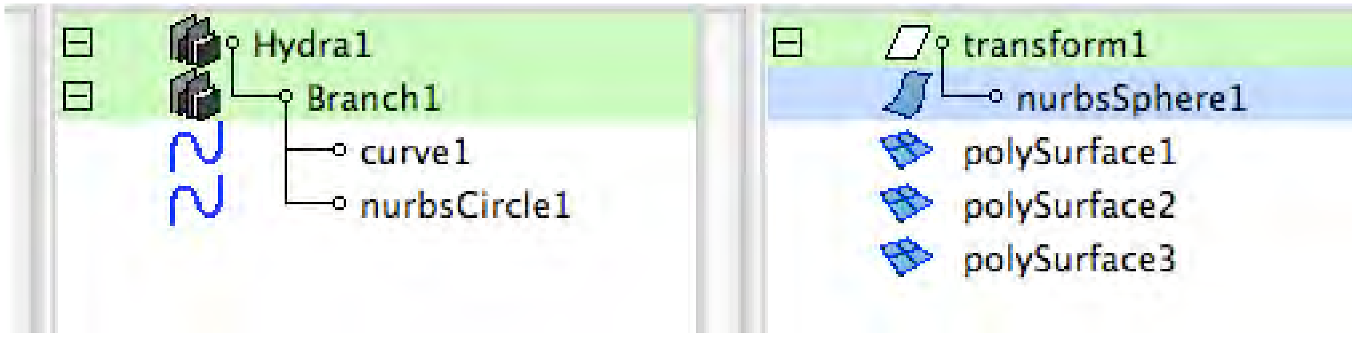
## 1. Access to single instance elements by instance-number

Using this offset-number all instance elements in the Xfrog-hierarchy can be accessed to remove or edit parameters for. To access more than one instance element at once, the following numbering is used:

| | |
|---|---|
| Single element: | 1 |
| Multiple elements: | 1,2,3,6,7 |
| Sequence: | 1*2*9 |
| Multiple and sequence: | 1,2,3,10*2*4,20*2*4 |
| | |
| How sequence works: | a*b*c |
| | = a,(a+1*b),(a+2*b),...,(a+c*b) |
| Example: | 1*2*9 |
| | = 1,3,5,7,9,11,13,15,17,19 |

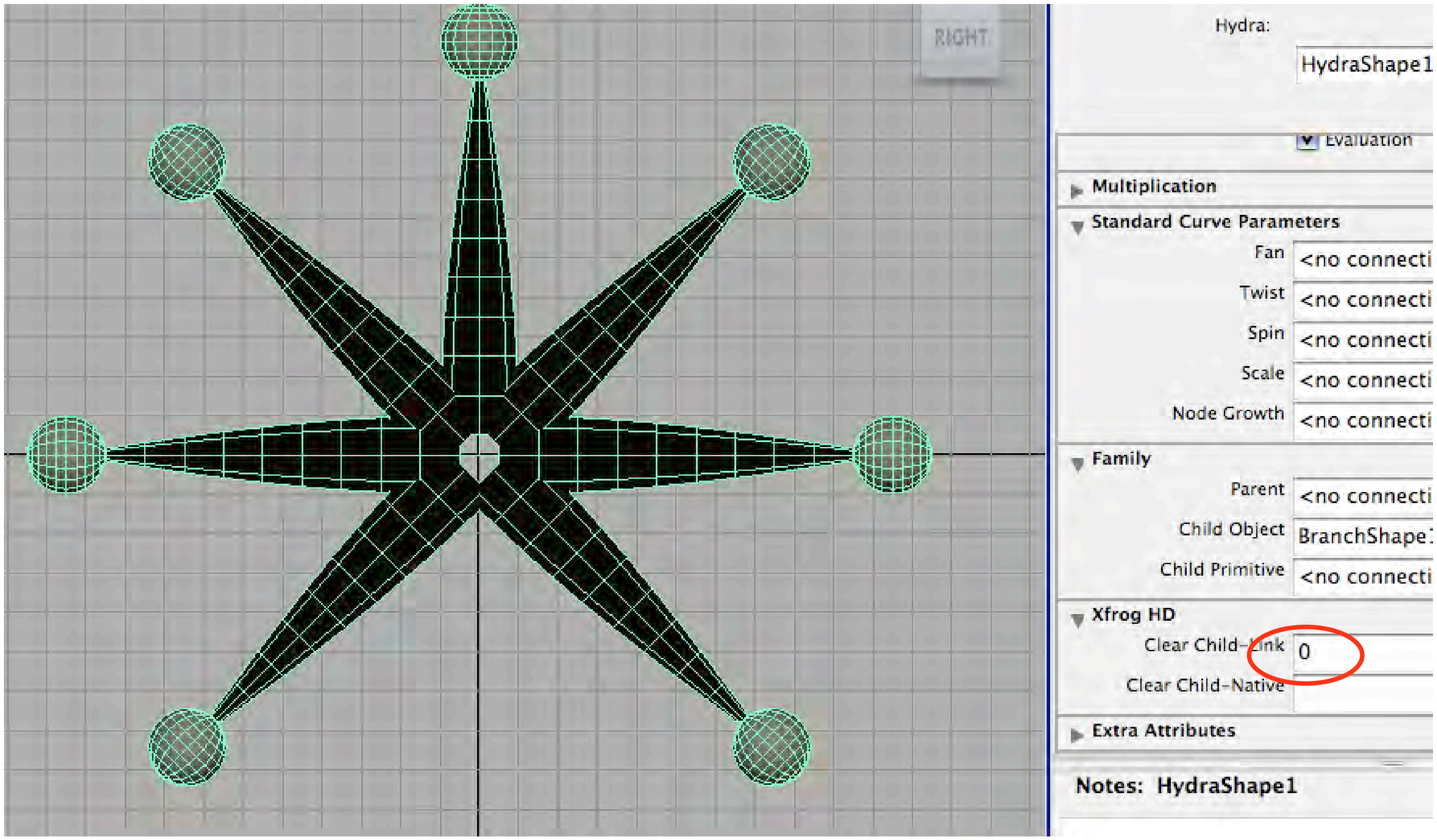
*The following pages show how to use the introduced offset-numbering.*

## 2. Remove single instance elements from whole objects



*Generate the following structure: Hydra - Branch - Primitive sphere as head*

## 2. Remove single instance elements from whole objects



Hydra: HydraShape1

Evaluation

Multiplication

Standard Curve Parameters

Fan <no connecti

Twist <no connecti

Spin <no connecti

Scale <no connecti

Node Growth <no connecti

Family

Parent <no connecti

Child Object BranchShape

Child Primitive <no connecti

Xfrog HD

Clear Child-Link 0

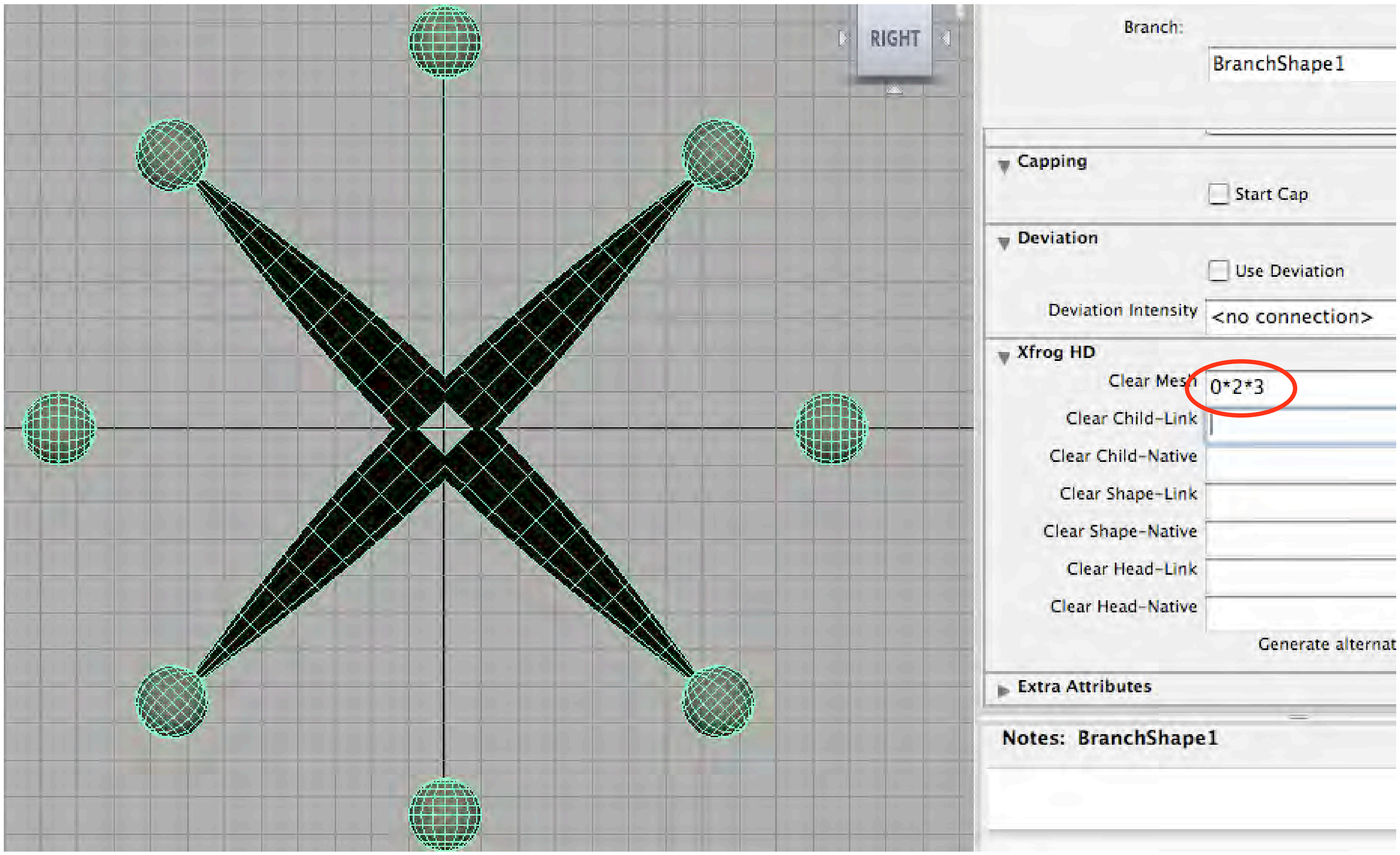
Clear Child-Native

Extra Attributes

Notes: HydraShape1

*Clear child-links from Hydra using the introduced offset*

## 2. Remove single instance elements from whole objects



Branch: BranchShape1

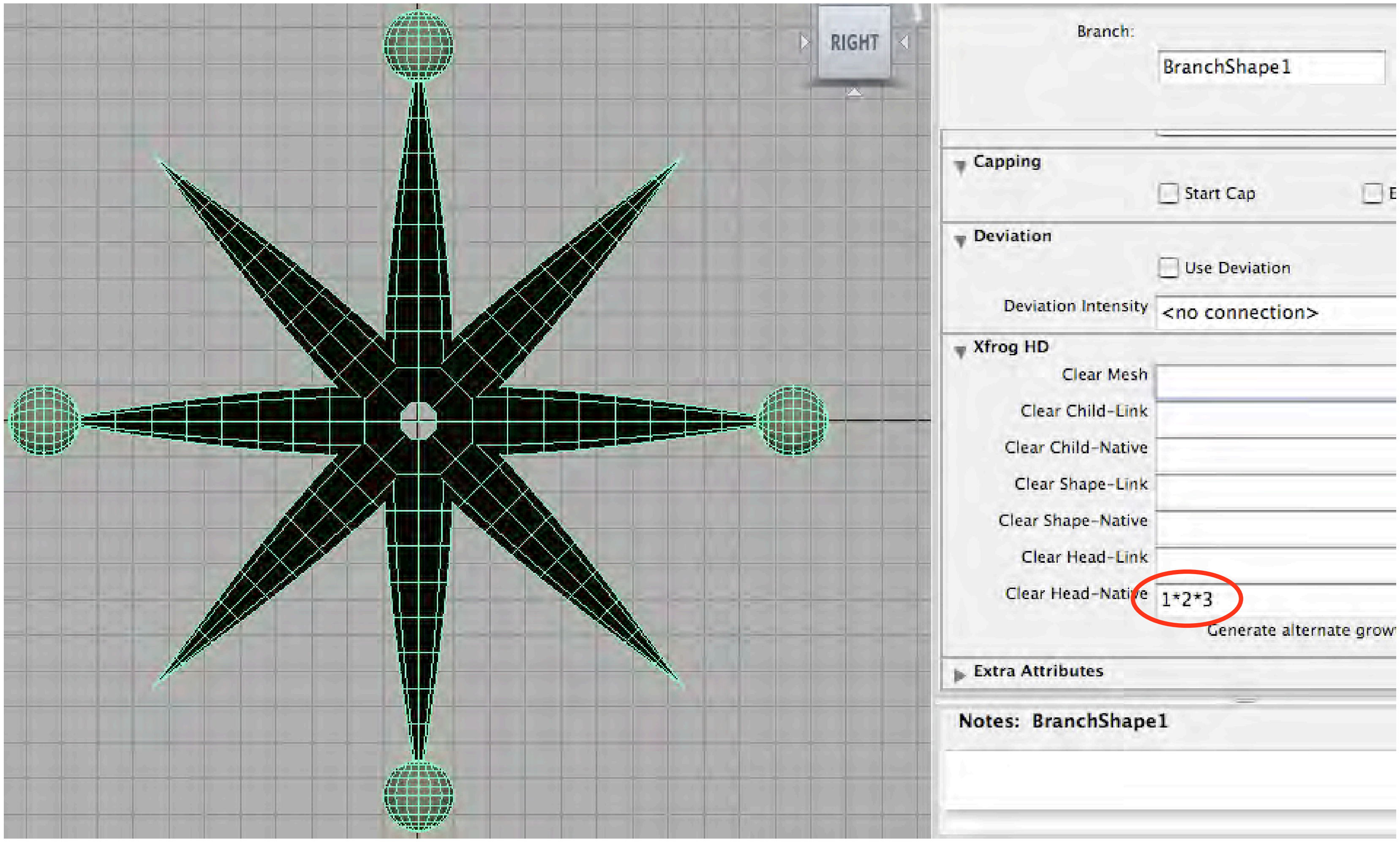
- [ ] Start Cap
- [ ] Use Deviation

| | |
|---|---|
| Deviation Intensity | <no connection> |
| Clear Mesh | 0*2*3 |
| Clear Child-Link | |
| Clear Child-Native | |
| Clear Shape-Link | |
| Clear Shape-Native | |
| Clear Head-Link | |
| Clear Head-Native | |

Notes: BranchShape1

*Clear mesh from Branch this time (only mesh is cleared, the head-links remain)*

## 2. Remove single instance elements from whole objects



Branch: BranchShape1

Capping
Start Cap

Deviation
Use Deviation
Deviation Intensity <no connection>

Xfrog HD
Clear Mesh
Clear Child-Link
Clear Child-Native
Clear Shape-Link
Clear Shape-Native
Clear Head-Link
Clear Head-Native 1*2*3
Generate alternate grow

Extra Attributes

Notes: BranchShape1

*This time clear primitives from Branch‘s head (the head-link remains)*

*Clearing the head-link would have cleared the links and the primitives too...*

## 2. Remove single instance elements from whole objects

Removal works for:

- Branch‘s child-, shape-, head-links and -primitives
- Branch‘s mesh
- Hydra‘s child-links and -primitives
- Phyllotaxis‘ child-links and -primitives
- Variation‘s child-links

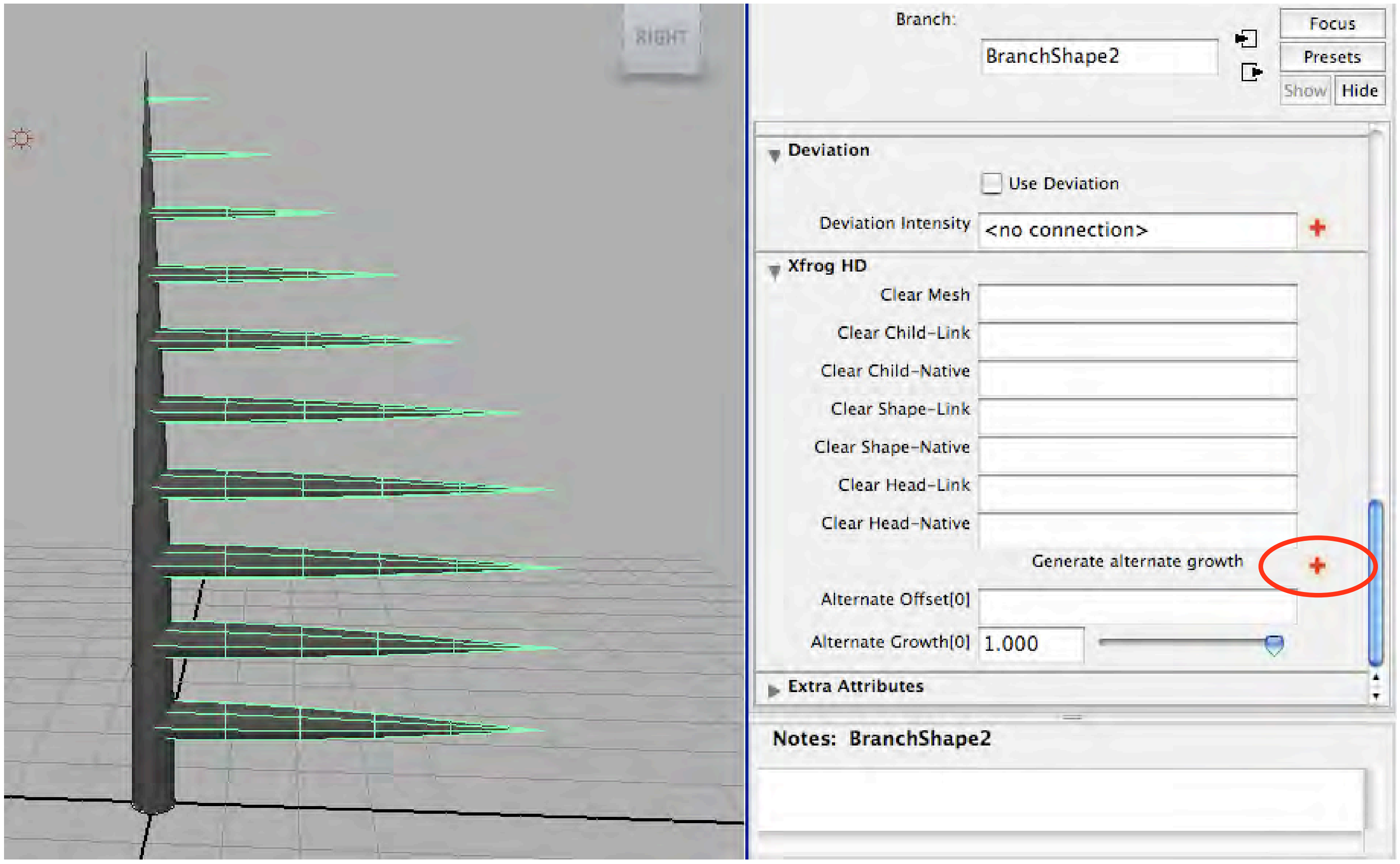
This way it‘s possible to:

- Remove selected branches from trees.
- Remove selected elements below Phyllotaxis or Hydra.
- Much more...

Ok, we can remove single instance elements. Now it would be cool to influence single instance elements. *See next pages...*

## 3. Individually access parameters of single instance element

A) Access simple parameters



Branch: BranchShape2

Use Deviation

Deviation Intensity <no connection>

Clear Mesh

Clear Child-Link

Clear Child-Native

Clear Shape-Link

Clear Shape-Native

Clear Head-Link

Clear Head-Native

Generate alternate growth

Alternate Offset[0]

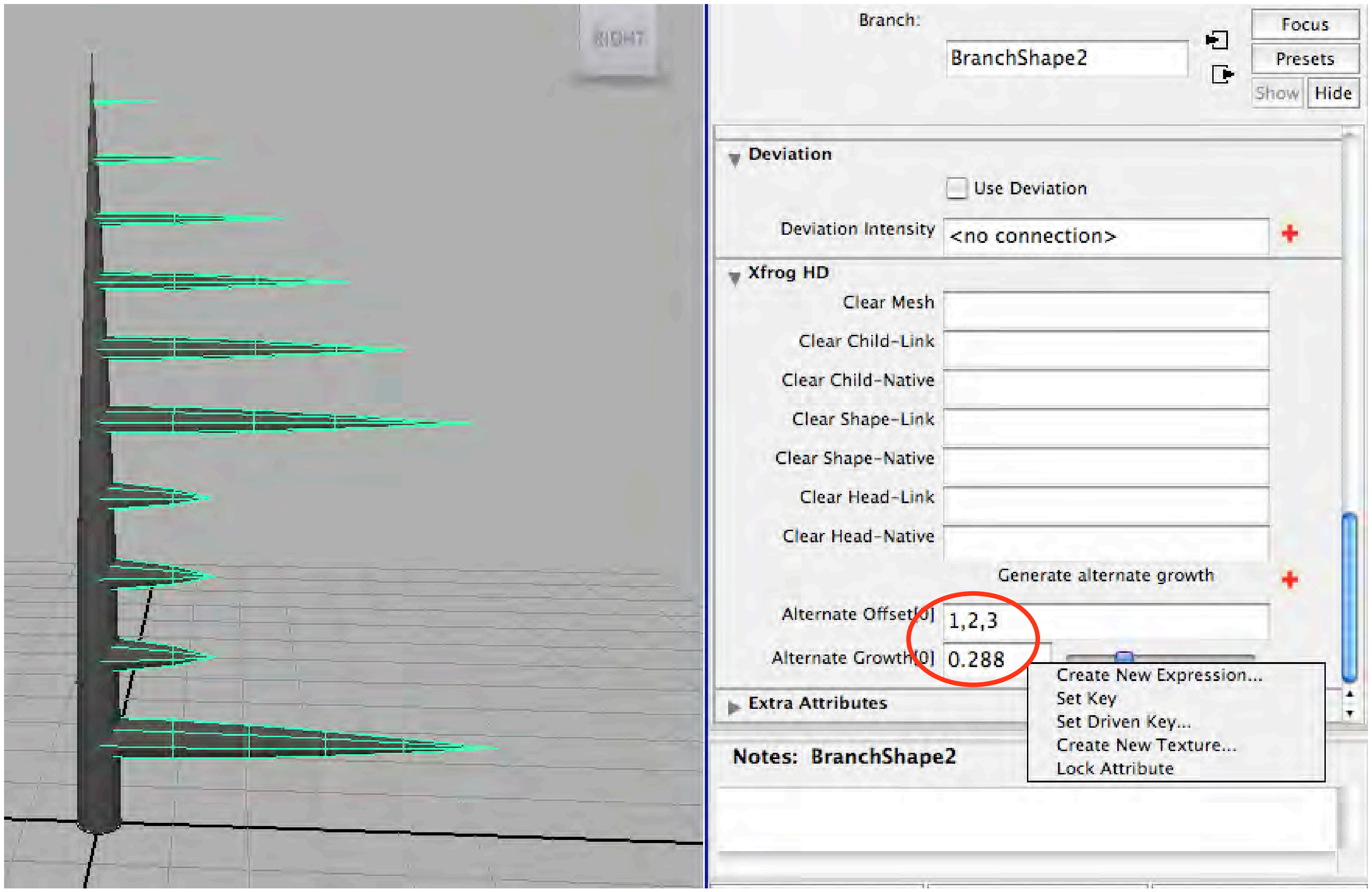
Alternate Growth[0] 1.000

Notes: BranchShape2

*Add alternate value for the simple parameter of a Branch as child of another Branch*

## 3. Individually access parameters of single instance element

A) Access simple parameters



Branch: BranchShape2

Deviation

Use Deviation

Deviation Intensity <no connection>

Xfrog HD

Clear Mesh

Clear Child-Link

Clear Child-Native

Clear Shape-Link

Clear Shape-Native

Clear Head-Link

Clear Head-Native

Generate alternate growth

Alternate Offset[0] 1,2,3

Alternate Growth[0] 0.288

Extra Attributes

Notes: BranchShape2

*Set offsets you want to have alternate growth for and set the alternate growth value*

*By keyframing the alternate value you can also animate the alternate growth*

## 3. Individually access parameters of single instance element

A) Access simple parameters

Simple parameters that can have alternate values:

Branch‘s growth

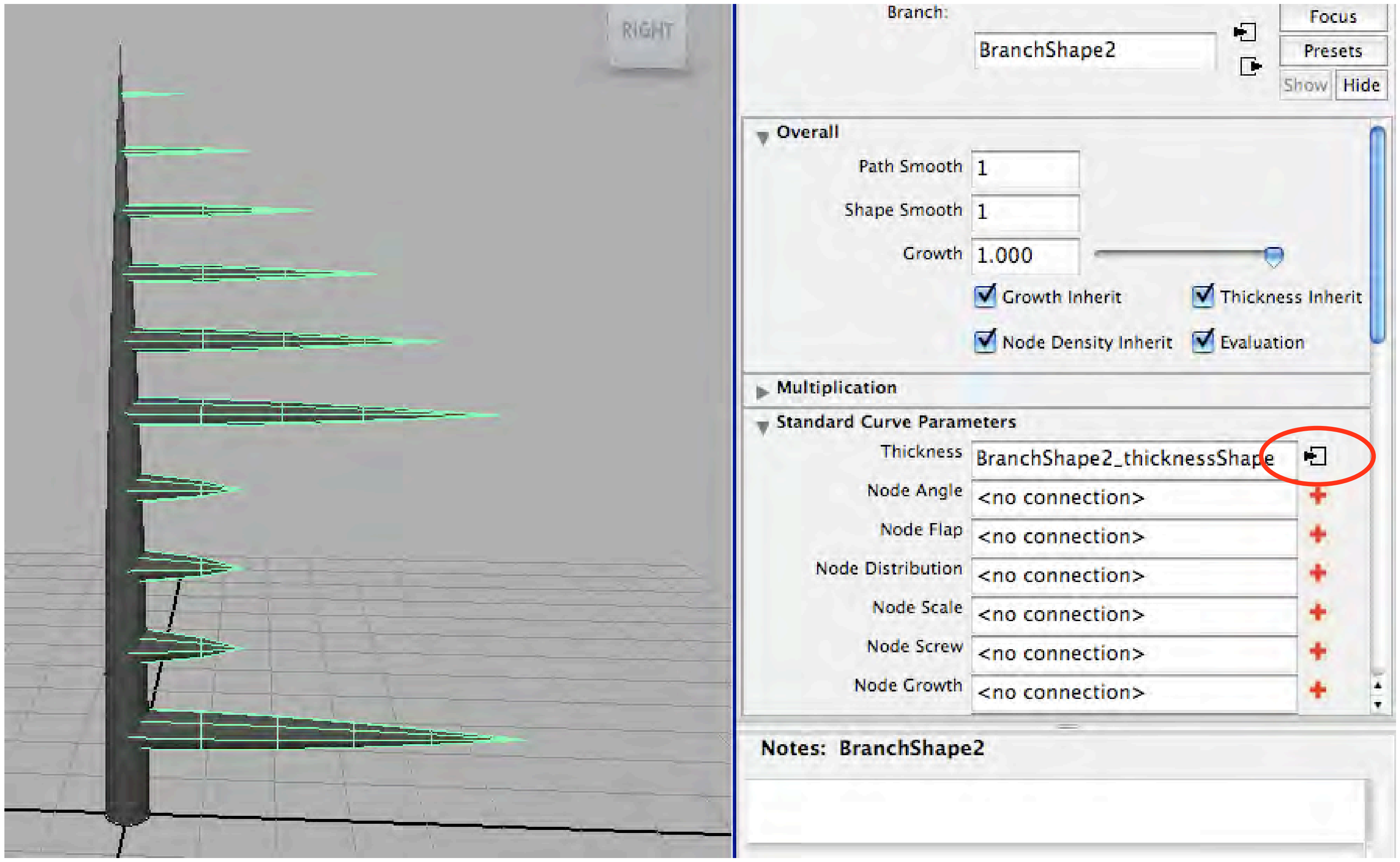
Variation‘s type and seed

By pressing the add-button more than once you can generate multiple different alternate values for one object. Each value can be assigned to any chosen offset and set or animated separately from the original value.

Cool so far. Now to the more complex parameters...

## 3. Individually access parameters of single instance element

B) Access complex parameters (curve-param)



Branch: BranchShape2

| | |
|---|---|
| Path Smooth | 1 |
| Shape Smooth | 1 |
| Growth | 1.000 |

[✓] Growth Inherit [✓] Thickness Inherit
[✓] Node Density Inherit [✓] Evaluation

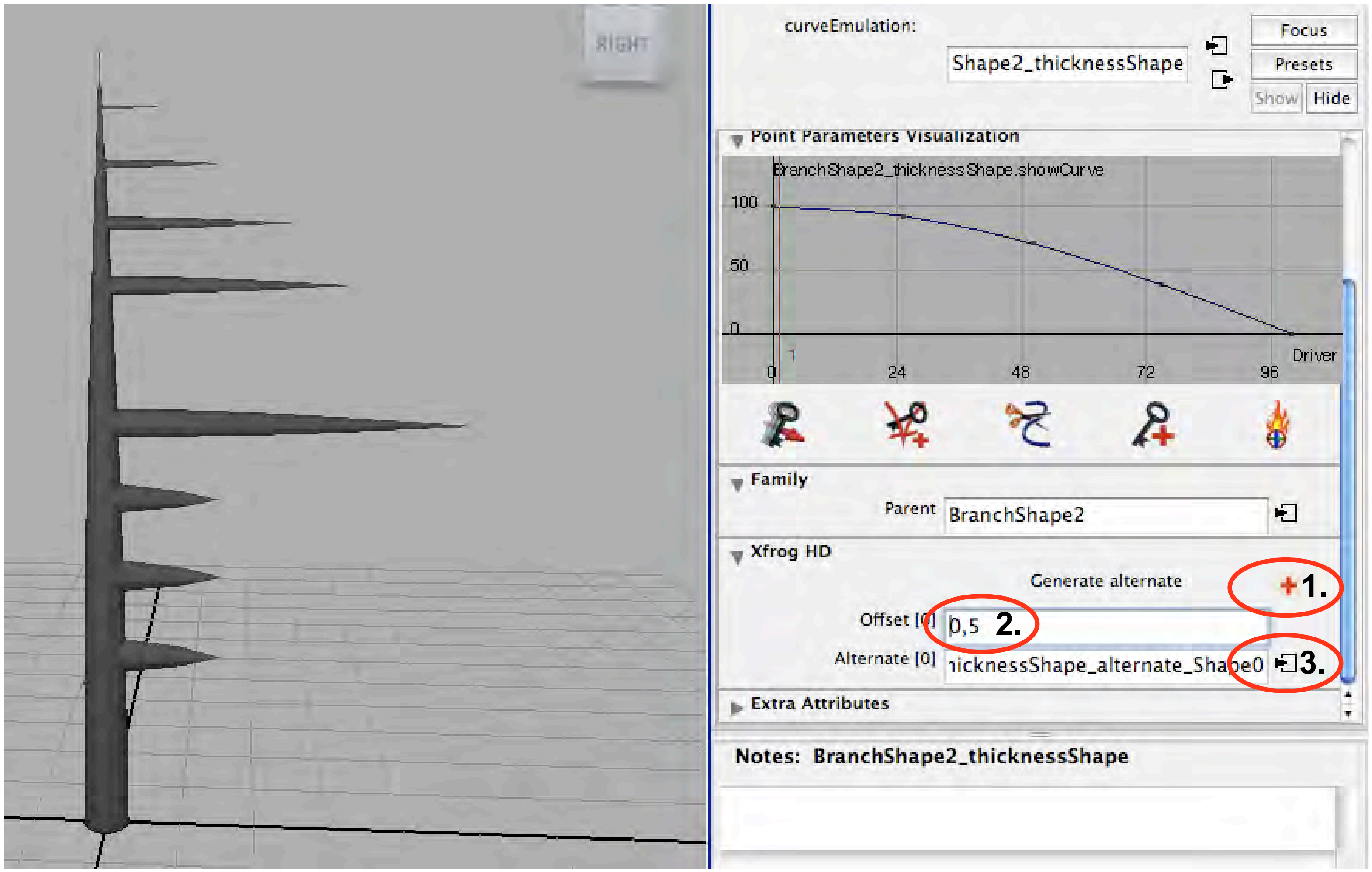
| | |
|---|---|
| Thickness | BranchShape2_thicknessShape |
| Node Angle | <no connection> |
| Node Flap | <no connection> |
| Node Distribution | <no connection> |
| Node Scale | <no connection> |
| Node Screw | <no connection> |
| Node Growth | <no connection> |

Notes: BranchShape2

*To add alternate curve-param for the known Branch, switch to the param to access*

## 3. Individually access parameters of single instance element

### B) Access complex parameters (curve-param)



curveEmulation: Shape2_thicknessShape

Parent BranchShape2

Offset [0] 0,5

Alternate [0] ıicknessShape_alternate_Shape0

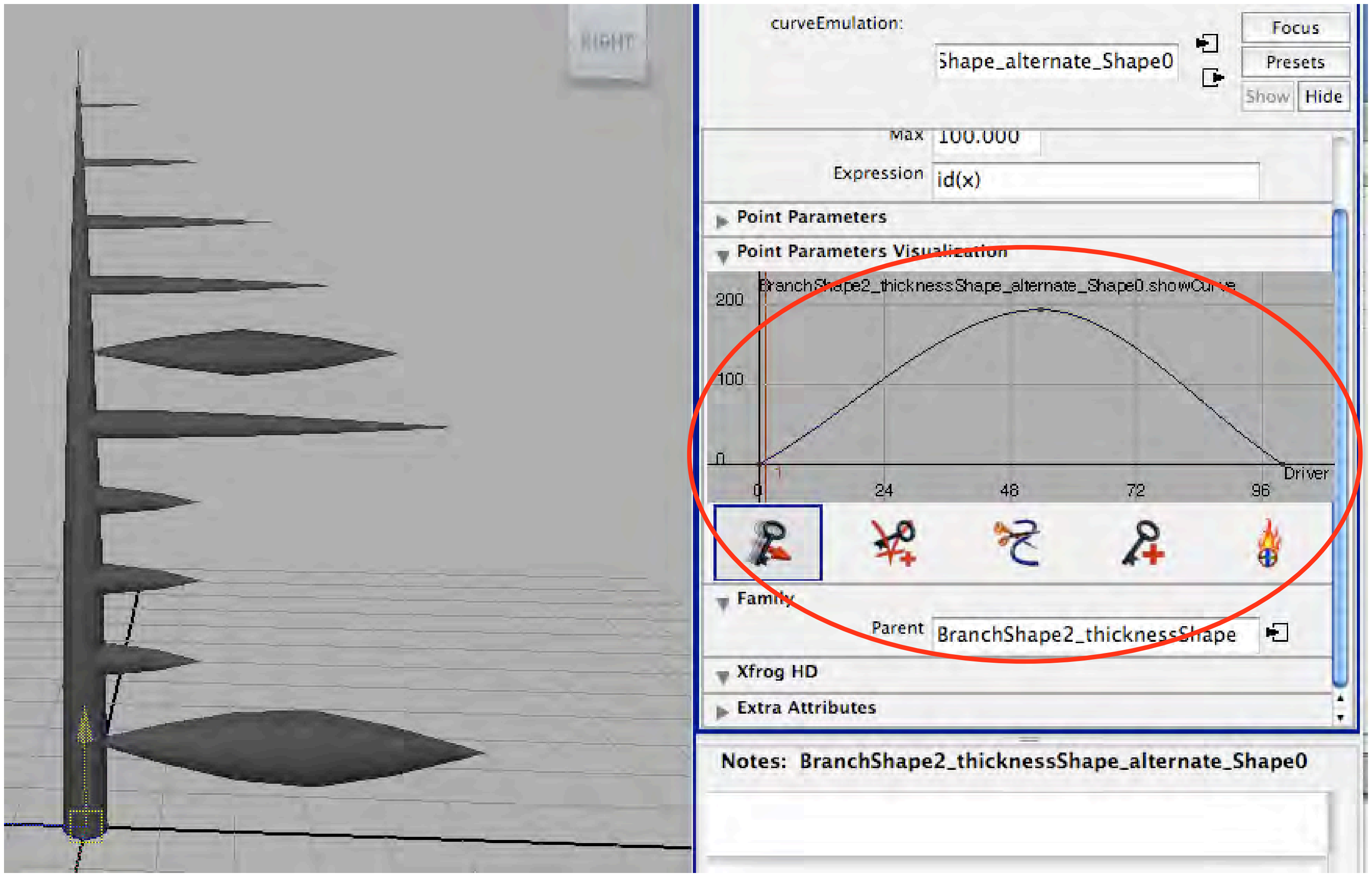
Notes: BranchShape2_thicknessShape

*At the curve-param generate an alternate param(1) and set offsets for the alternate(2)*

*As soon as you set the offset, the alternate param becomes valid for these elements. Now switch to the generated alternate param (3) to set it.*

## 3. Individually access parameters of single instance element

B) Access complex parameters (curve-param)


RIGHT
curveEmulation:
Shape_alternate_Shape0
Focus
Presets
Show
Hide
Max
100.000
Expression
id(x)
Point Parameters
Point Parameters Visualization
BranchShape2_thicknessShape_alternate_Shape0.showCurve
200
100
0
Driver
0
24
48
72
96
Family
Parent
BranchShape2_thicknessShape
Xfrog HD
Extra Attributes
Notes: BranchShape2_thicknessShape_alternate_Shape0

*At the alternate curve-param edit the curve-points to the required values.*

## 3. Individually access parameters of single instance element

### B) Access complex parameters (curve-param)

All curve-parameters of any Xfrog object can have alternate values.
By pressing the add-button more than once you can generate multiple different alternate values for one curve-param. Each value can be assigned to any chosen offset and set or animated separately from the original value.

This way you can do the following:

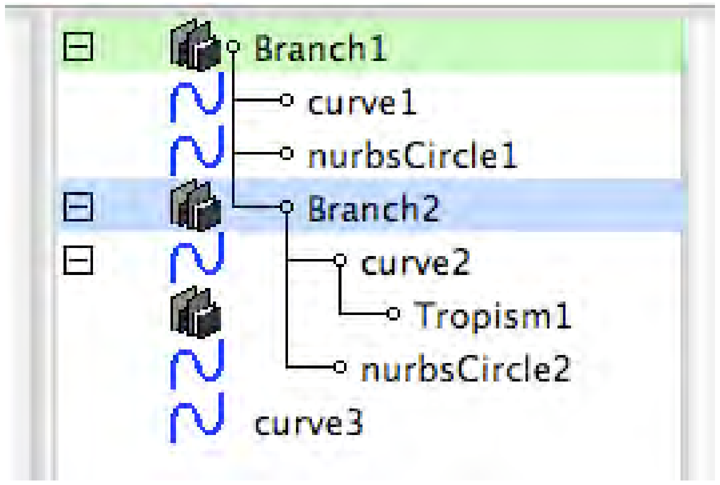
- Edit curve-params for CurveNurbs being path- or shape-spline of a Branch to change shape of single Branch element.
- Edit curve-params for Phyllotaxis or Hydra to change shape of single Phyllo-ball etc. in the full set off multiplied Phyllos, Hydras.
- Much more...

Now we have nearly all parameters of Xfrog objects editable on a single instance level. There are just 2 types of parameters remaining:

- Child-count-parameters: *Setting alternate values here would make the offset-numbering extremely complex. Thatswhy it‘s not possible to set alternates here. The new possibilities of Variation will help here.*
- Path- or shape-spline-parameters using simple splines: *See next pages*

## 3. Individually access parameters of single instance element

C) Access complex parameters (path-, shape-spline)

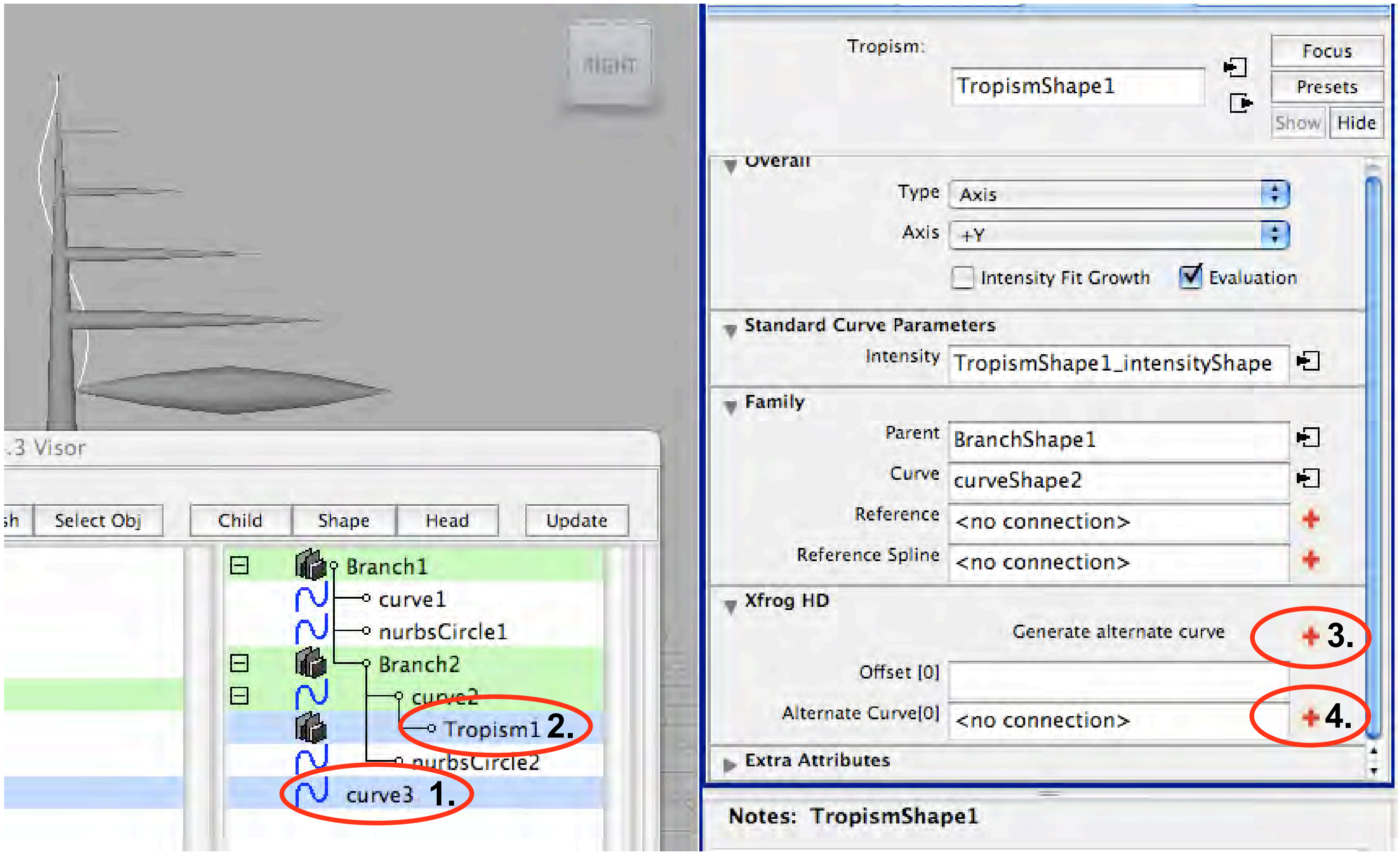


*We use the known Branch. Add a tropism to the path-spline of the 2nd level Branch. Generate the spline being the alternate path-spline for the 2nd level Branch.*

*Setting alterate path- or shape-splines leads over a Tropism. If you don‘t need the Tropism-functionality, you can set the Tropism-Intensity to Zero.*

## 3. Individually access parameters of single instance element

C) Access complex parameters (path-, shape-spline)



Tropism: TropismShape1

Type: Axis

Axis: +Y

Intensity Fit Growth ☐ Evaluation ✓

Intensity: TropismShape1_intensityShape

Parent: BranchShape1

Curve: curveShape2

Reference: <no connection>

Reference Spline: <no connection>

Offset [0]:

Alternate Curve[0]: <no connection>

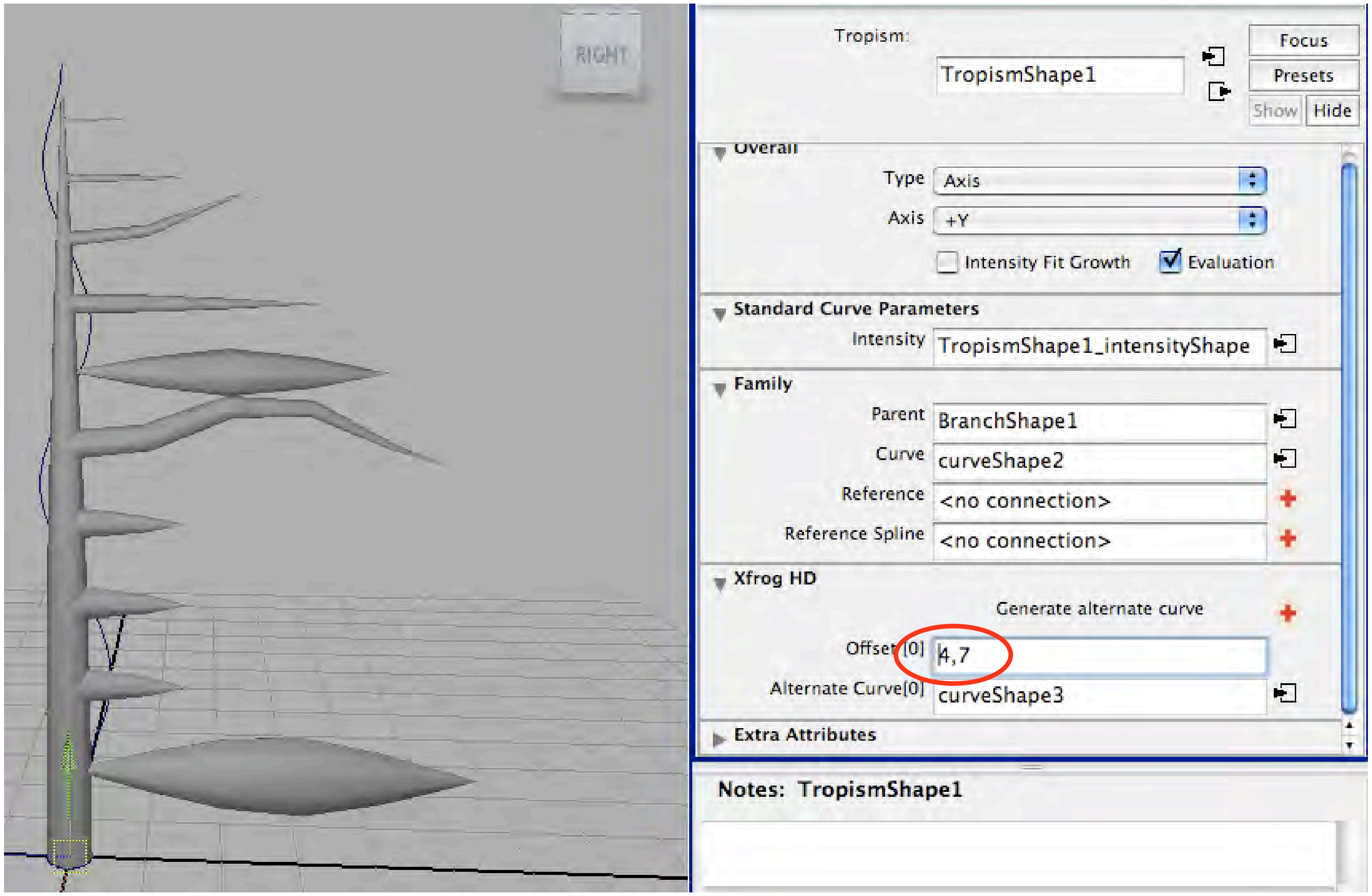
Notes: TropismShape1

*First select the alternate path-spline(1), then add-select the generated Tropism(2). At the Tropism generate an alternate curve(3). Now add the alternate spline(4).*

*The alternate spline is grouped under the Tropism object.*

## 3. Individually access parameters of single instance element

C) Access complex parameters (path-, shape-spline)



Tropism: TropismShape1

Overall

Type: Axis

Axis: +Y

[ ] Intensity Fit Growth   [✓] Evaluation

Standard Curve Parameters

Intensity: TropismShape1_intensityShape

Family

Parent: BranchShape1

Curve: curveShape2

Reference: <no connection>

Reference Spline: <no connection>

Xfrog HD

Generate alternate curve

Offset[0]: 4,7

Alternate Curve[0]: curveShape3

Extra Attributes

Notes: TropismShape1

*Now you can set the offsets for the alternate path-spline.*

## 3. Individually access parameters of single instance element

C) Access complex parameters (path-, shape-spline)

This way you can edit the following:

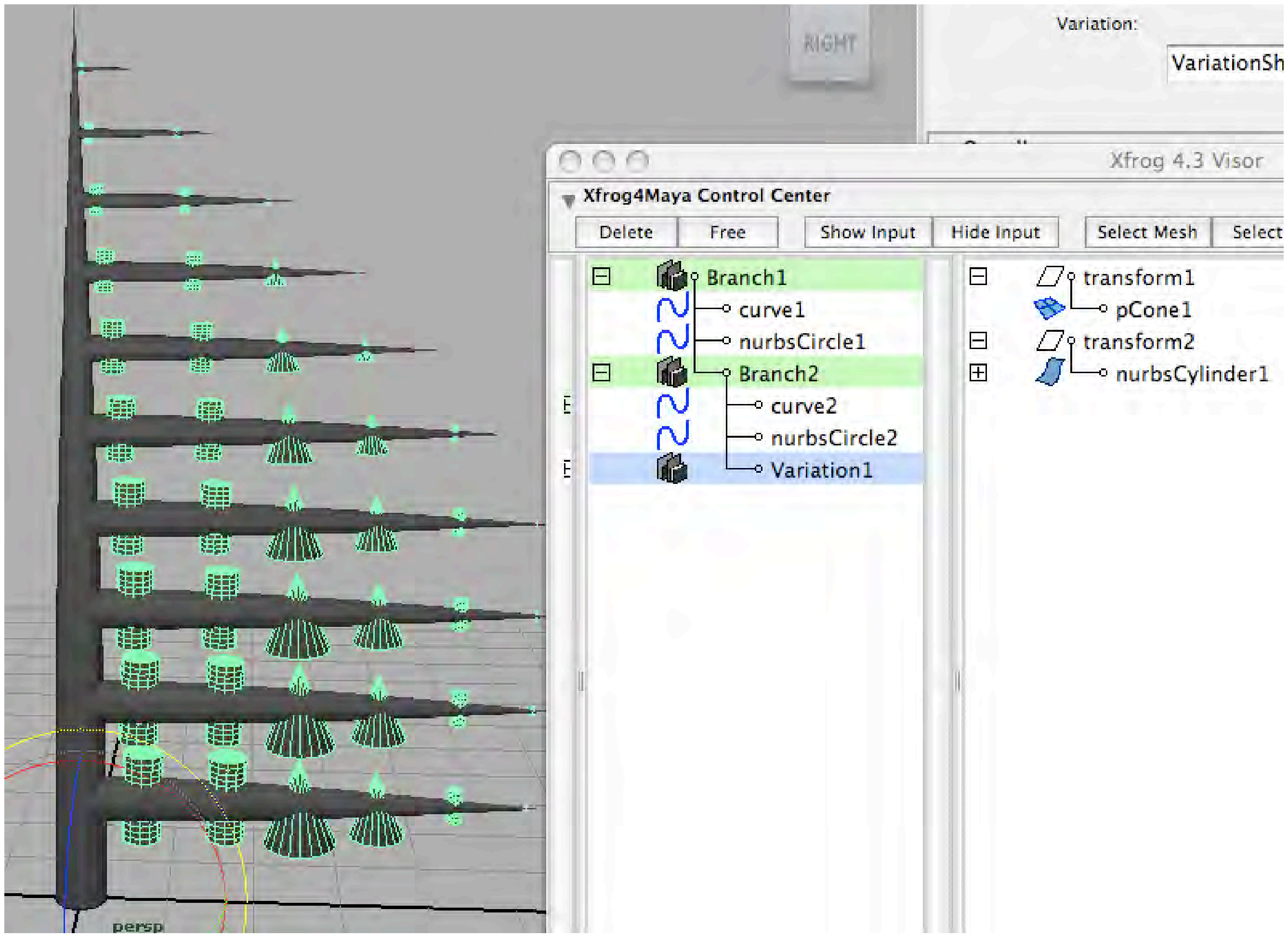
Branch‘s path- and shape-spline

Phyllotaxis‘ path-spline

Now we have a full set of features to edit all Xfrog objects on a single instance level.

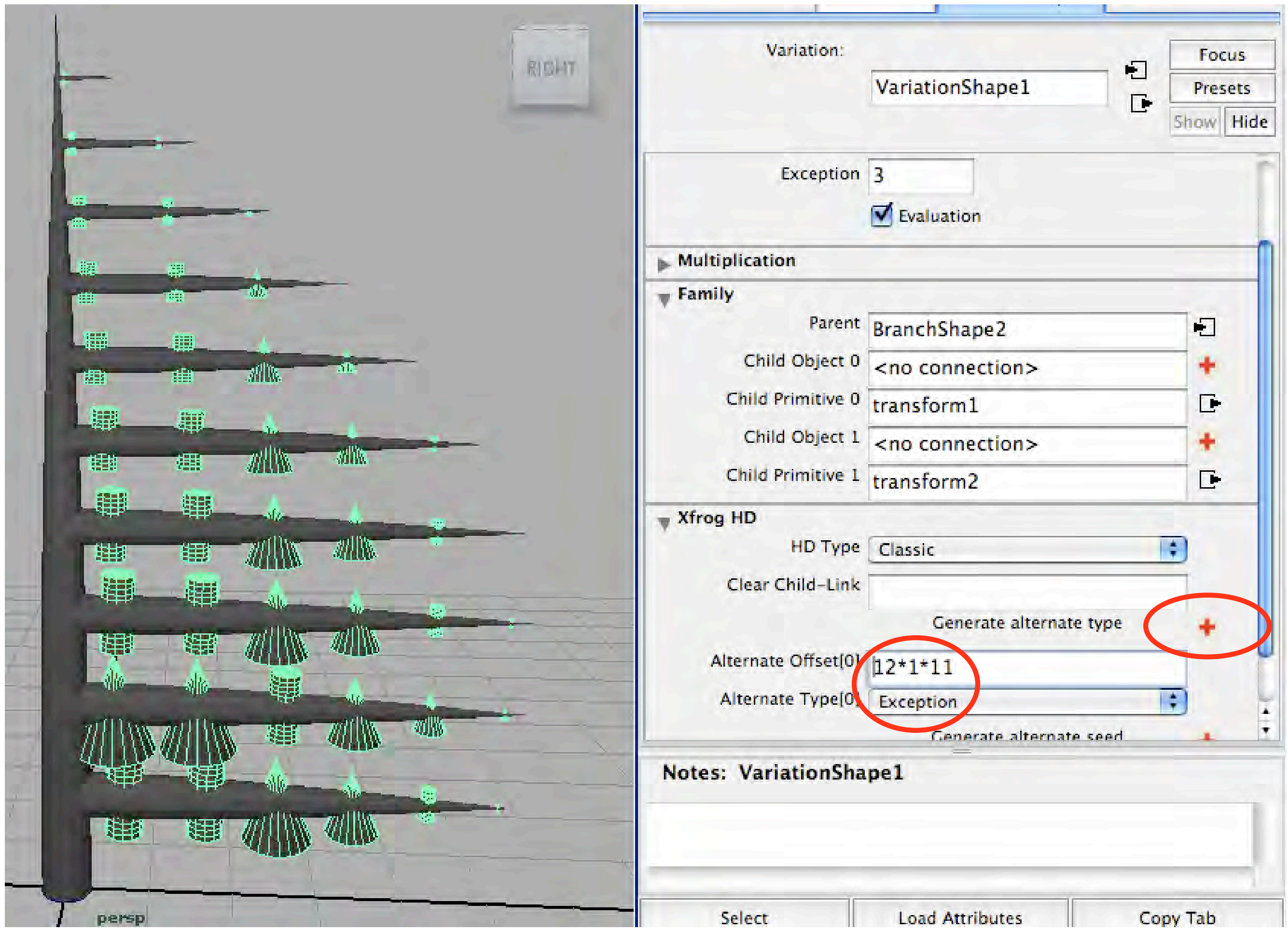
New Variation features complete the possibilities for instance-level editing and higher flexibility in combination of objects using a Variation. *See next...*

## 4. Higher flexibility in combination of elements



*Generate a Variation under the known Branch and some primitives in variation.*

## 4. Higher flexibility in combination of elements



Variation: VariationShape1

Exception 3

Evaluation ✓

Parent BranchShape2

Child Object 0 <no connection>

Child Primitive 0 transform1

Child Object 1 <no connection>

Child Primitive 1 transform2

HD Type Classic

Clear Child-Link

Alternate Offset[0] 12*1*11

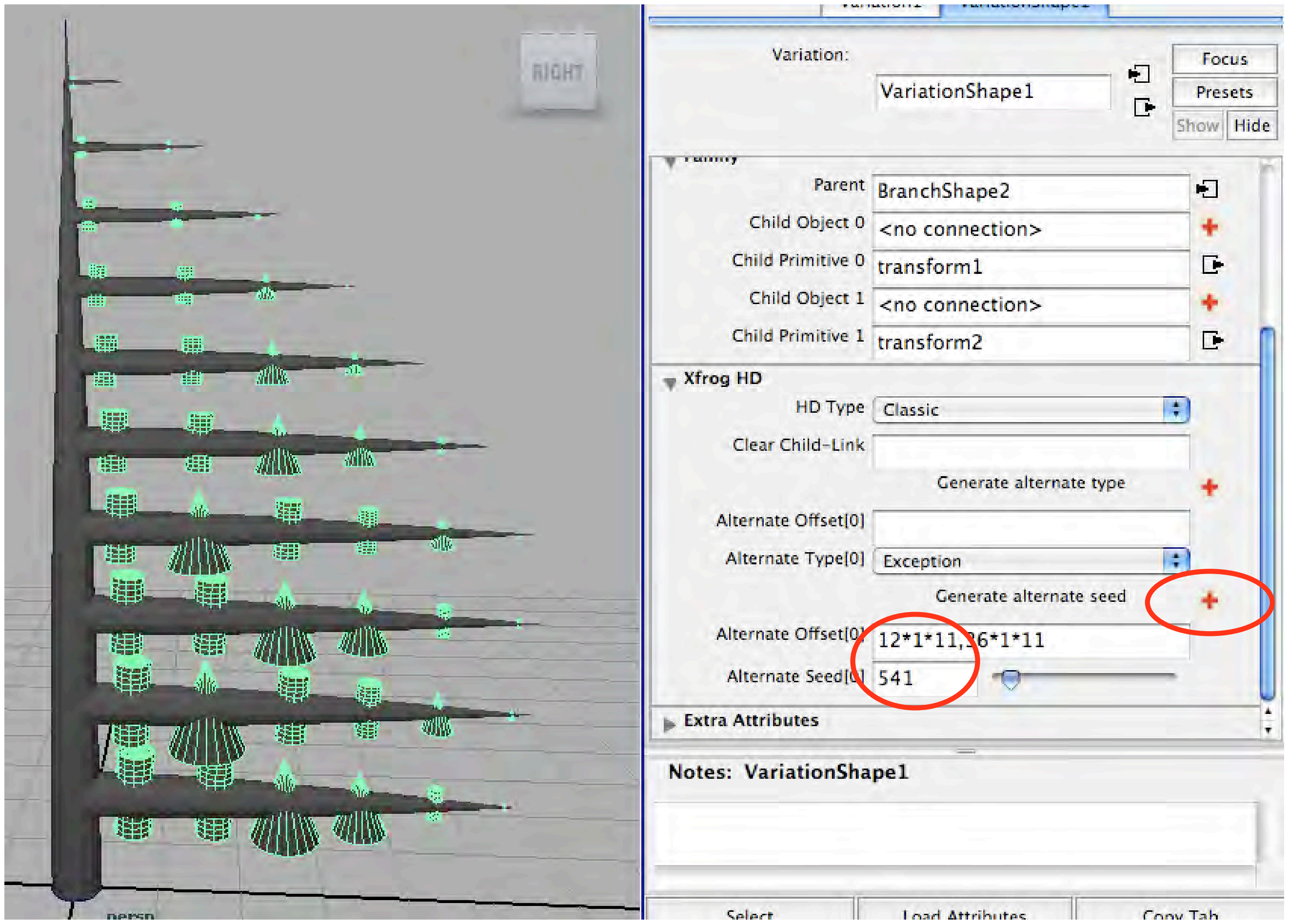
Alternate Type[0] Exception

Notes: VariationShape1

*First we have alternates for the simple parameter Variation-type.*

*Here the children of the 2nd Branch-instance (offsets 12-23) get the Variation-type Exception (with Exception = 3). All other have type Random.*

## 4. Higher flexibility in combination of elements



Variation: VariationShape1

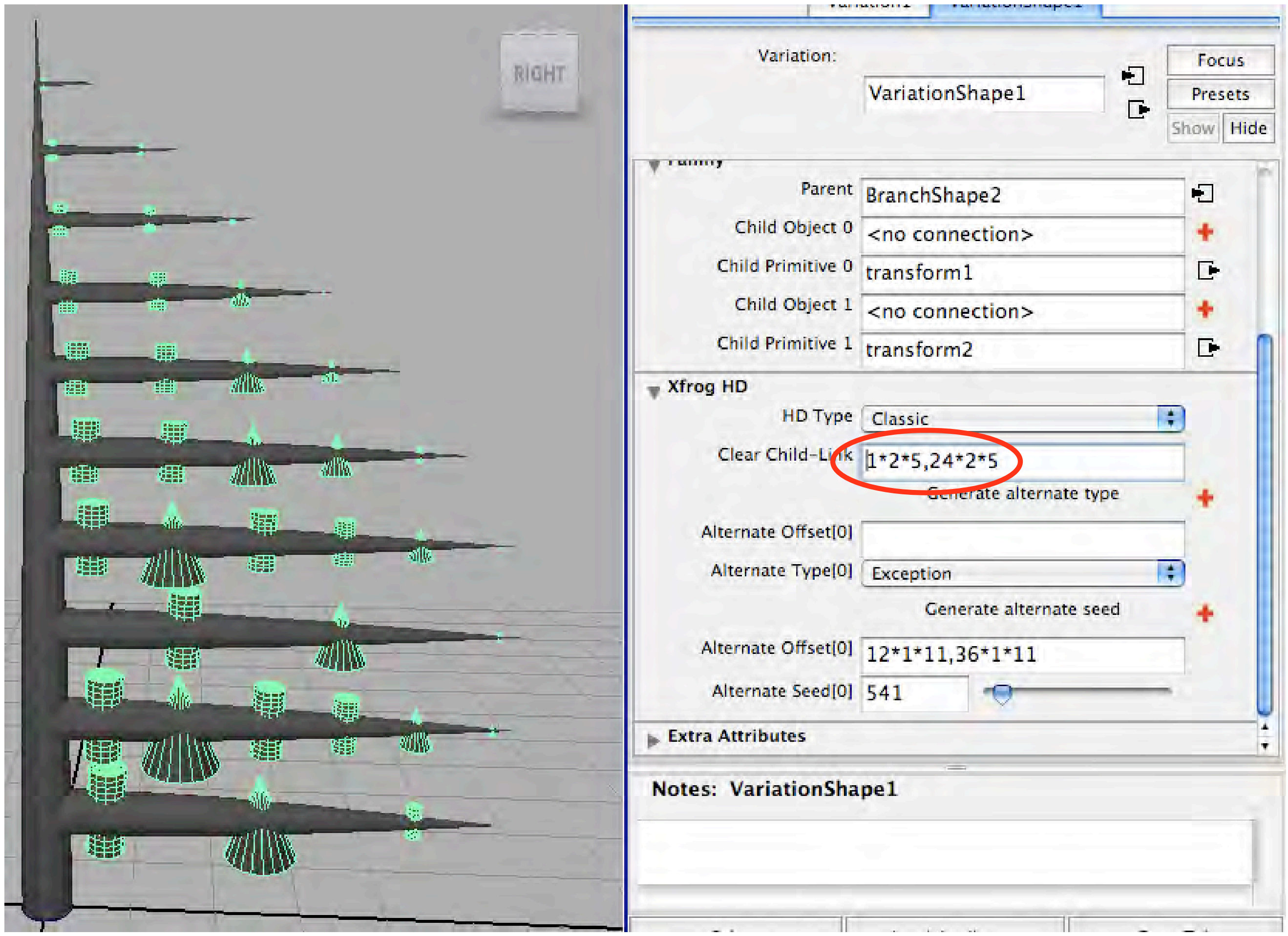
| | |
|---|---|
| Parent | BranchShape2 |
| Child Object 0 | <no connection> |
| Child Primitive 0 | transform1 |
| Child Object 1 | <no connection> |
| Child Primitive 1 | transform2 |
| HD Type | Classic |
| Clear Child-Link | |
| Alternate Offset[0] | |
| Alternate Type[0] | Exception |
| Alternate Offset[0] | 12*1*11,36*1*11 |
| Alternate Seed[0] | 541 |

Notes: VariationShape1

*Now we have alternates for the simple parameter Variation-seed.*

*Here the 2nd and 4th instance-children have an alternate seed. Using more alternates you can reach random distribution over all branches.*

## 4. Higher flexibility in combination of elements



Variation: VariationShape1

| | |
|---|---|
| Parent | BranchShape2 |
| Child Object 0 | <no connection> |
| Child Primitive 0 | transform1 |
| Child Object 1 | <no connection> |
| Child Primitive 1 | transform2 |

Xfrog HD

| | |
|---|---|
| HD Type | Classic |
| Clear Child-Link | 1*2*5,24*2*5 |
| | Generate alternate type |
| Alternate Offset[0] | |
| Alternate Type[0] | Exception |
| | Generate alternate seed |
| Alternate Offset[0] | 12*1*11,36*1*11 |
| Alternate Seed[0] | 541 |

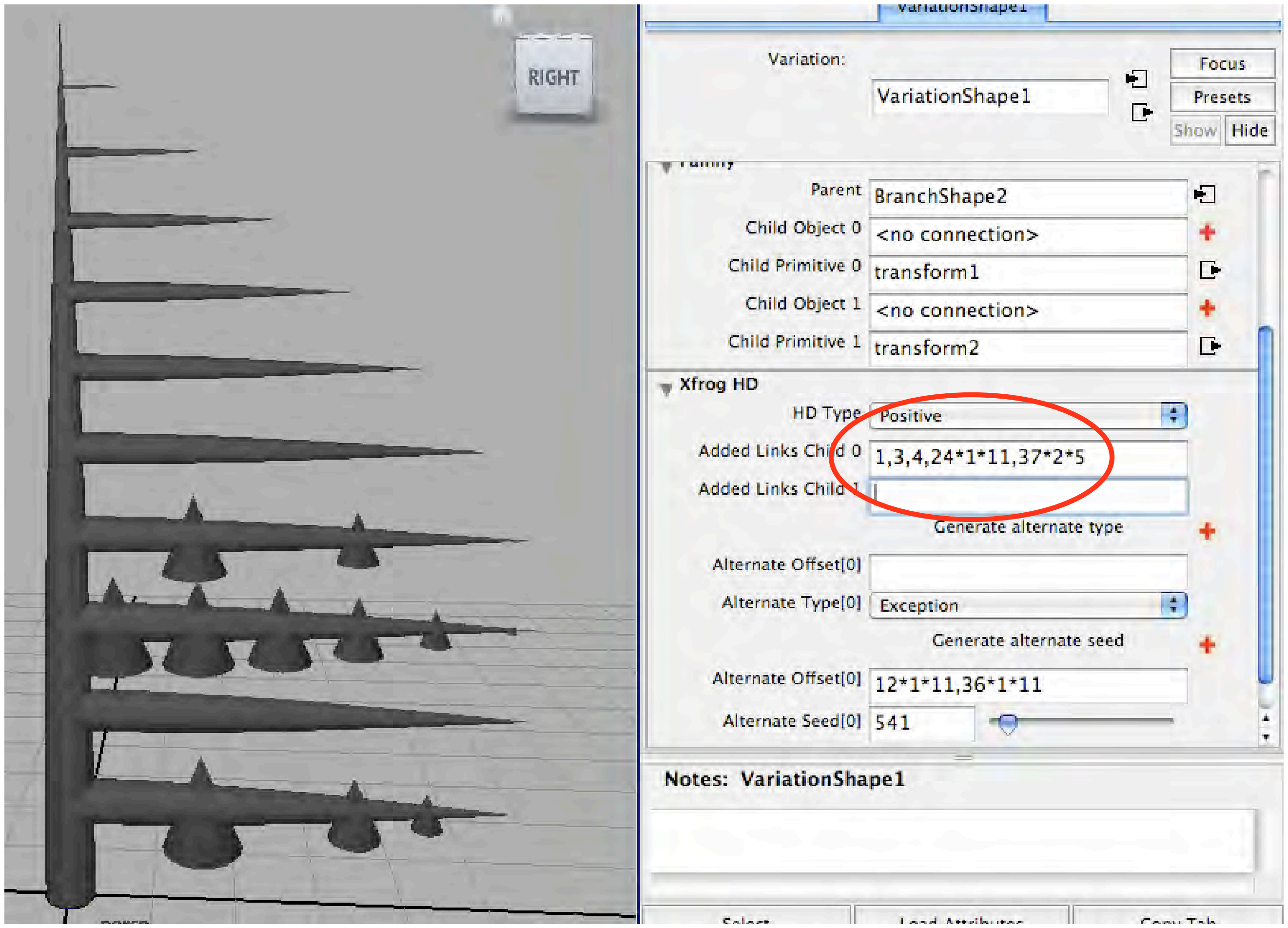
Extra Attributes

Notes: VariationShape1

*In addition we can remove single instance elements using the clear-functionality.*

*This way we can make random distributions that do not repeat for all parent Branches - these can be random over the full object.*

## 4. Higher flexibility in combination of elements



Variation: VariationShape1

Parent: BranchShape2
Child Object 0: <no connection>
Child Primitive 0: transform1
Child Object 1: <no connection>
Child Primitive 1: transform2

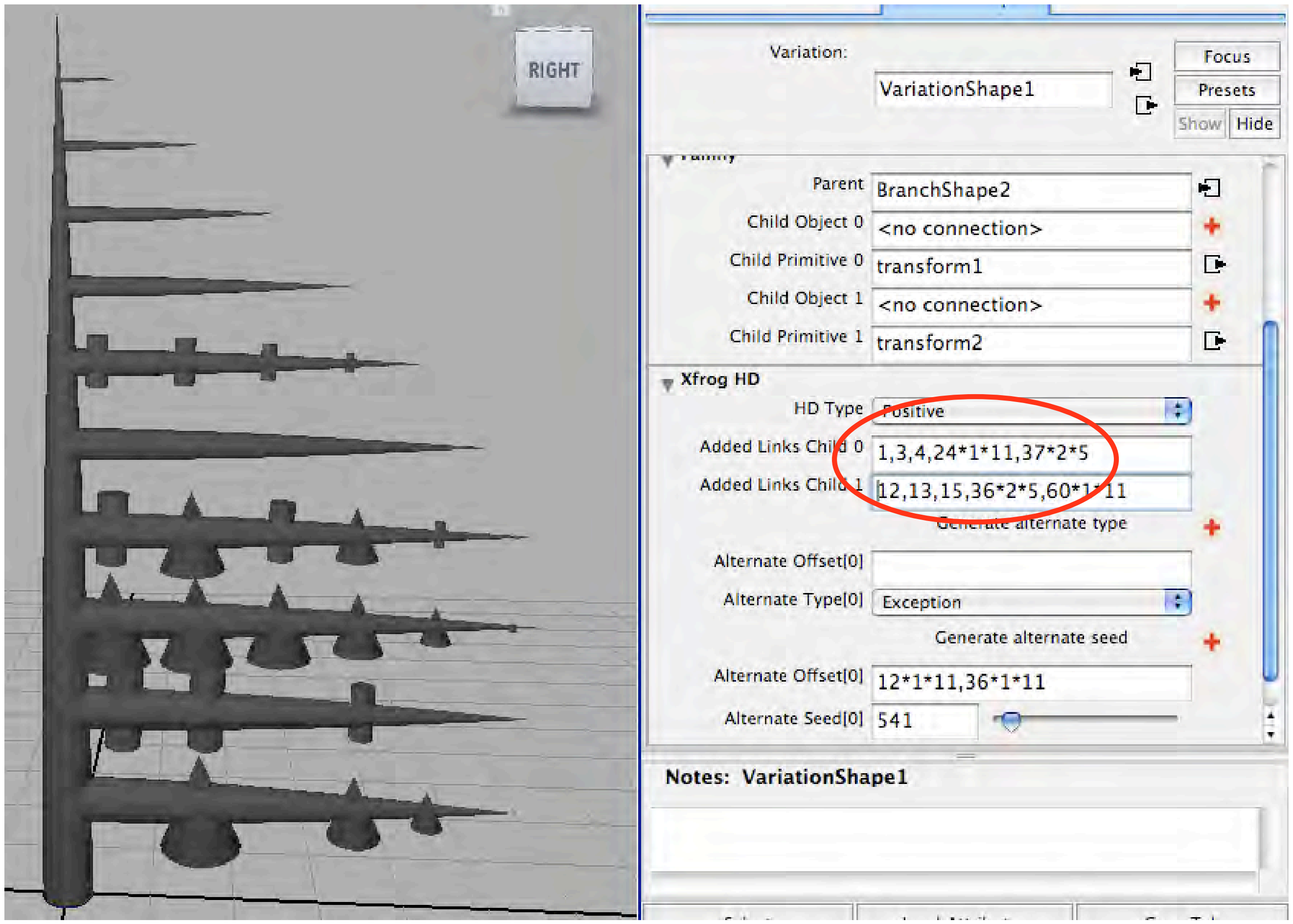
Xfrog HD
HD Type: Positive
Added Links Child 0: 1,3,4,24*1*11,37*2*5
Added Links Child 1:
Generate alternate type
Alternate Offset[0]:
Alternate Type[0]: Exception
Generate alternate seed
Alternate Offset[0]: 12*1*11,36*1*11
Alternate Seed[0]: 541

Notes: VariationShape1

*Now we have a look at the new 5-types Positive and Negative.*

*This way you can exactly choose, which elements are visible over the full object.*

## 4. Higher flexibility in combination of elements



Variation: VariationShape1

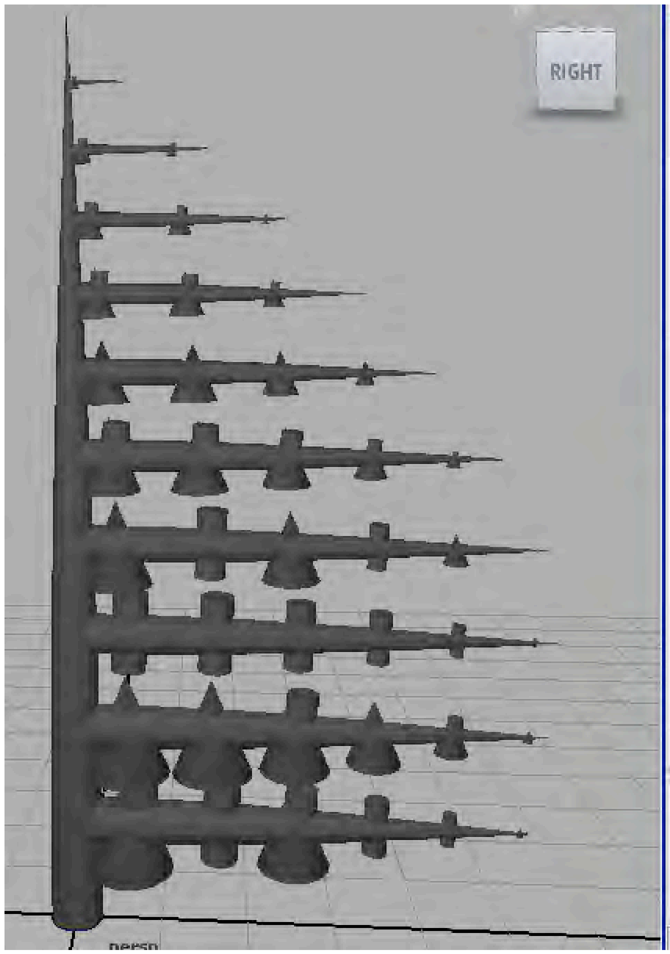
Parent: BranchShape2
Child Object 0: <no connection>
Child Primitive 0: transform1
Child Object 1: <no connection>
Child Primitive 1: transform2

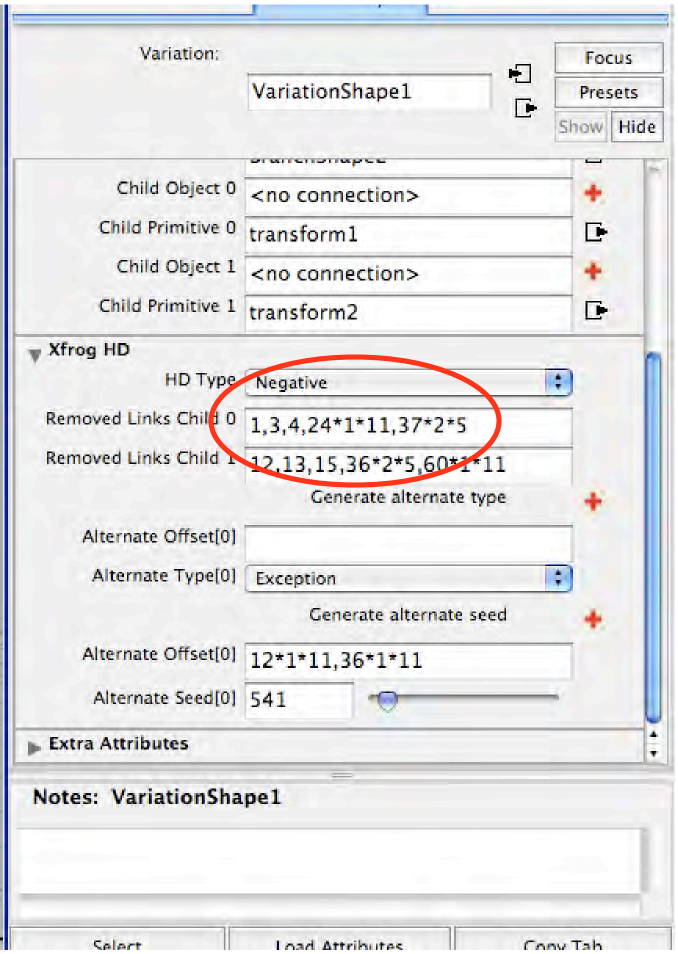
Xfrog HD
HD Type: Positive
Added Links Child 0: 1,3,4,24*1*11,37*2*5
Added Links Child 1: 12,13,15,36*2*5,60*1*11
Generate alternate type
Alternate Offset[0]:
Alternate Type[0]: Exception
Generate alternate seed
Alternate Offset[0]: 12*1*11,36*1*11
Alternate Seed[0]: 541

Notes: VariationShape1

*Positive means, all set offsets become visible.*

## 4. Higher flexibility in combination of elements




Variation: VariationShape1

| | |
|---|---|
| Child Object 0 | <no connection> |
| Child Primitive 0 | transform1 |
| Child Object 1 | <no connection> |
| Child Primitive 1 | transform2 |

Xfrog HD

| | |
|---|---|
| HD Type | Negative |
| Removed Links Child 0 | 1,3,4,24*1*11,37*2*5 |
| Removed Links Child 1 | 12,13,15,36*2*5,60*1*11 |

Generate alternate type

| | |
|---|---|
| Alternate Offset[0] | |
| Alternate Type[0] | Exception |

Generate alternate seed

| | |
|---|---|
| Alternate Offset[0] | 12*1*11,36*1*11 |
| Alternate Seed[0] | 541 |

Extra Attributes

Notes: VariationShape1

*Negative means, all offsets are visible. Only the set offsets are removed from display.*

## 4. Higher flexibility in combination of elements

You can do the new types of variation for as many children as you like:

Set the child-count the the number of children as you require.

Set the 5-type to positive or negative.

You get as many offset-fields as you have children.

For each of the children you can now set the offsets for display or hiding.

This way you can generate single sawn branches:

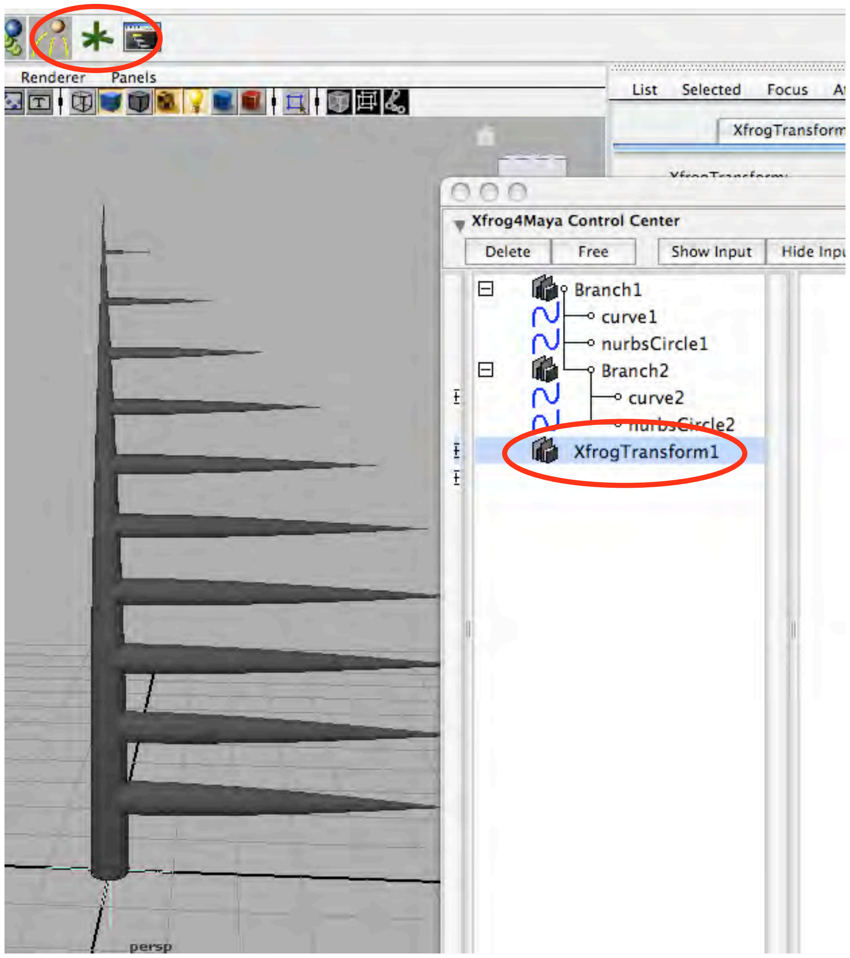
Remove the Branch instance element to be sawn by the clear-link-field.

Use a Variation to add a single Branch instance of another Branch to exactly this instance offset.

Use pruning against invisible box to saw just this single Branch instance.

And you can imagine so much more, after each instance is accessible separately. I think this gives you the flexibility many of your customers always requested. Editing Xfrog-hierarchies in high definition - Xfrog 5.

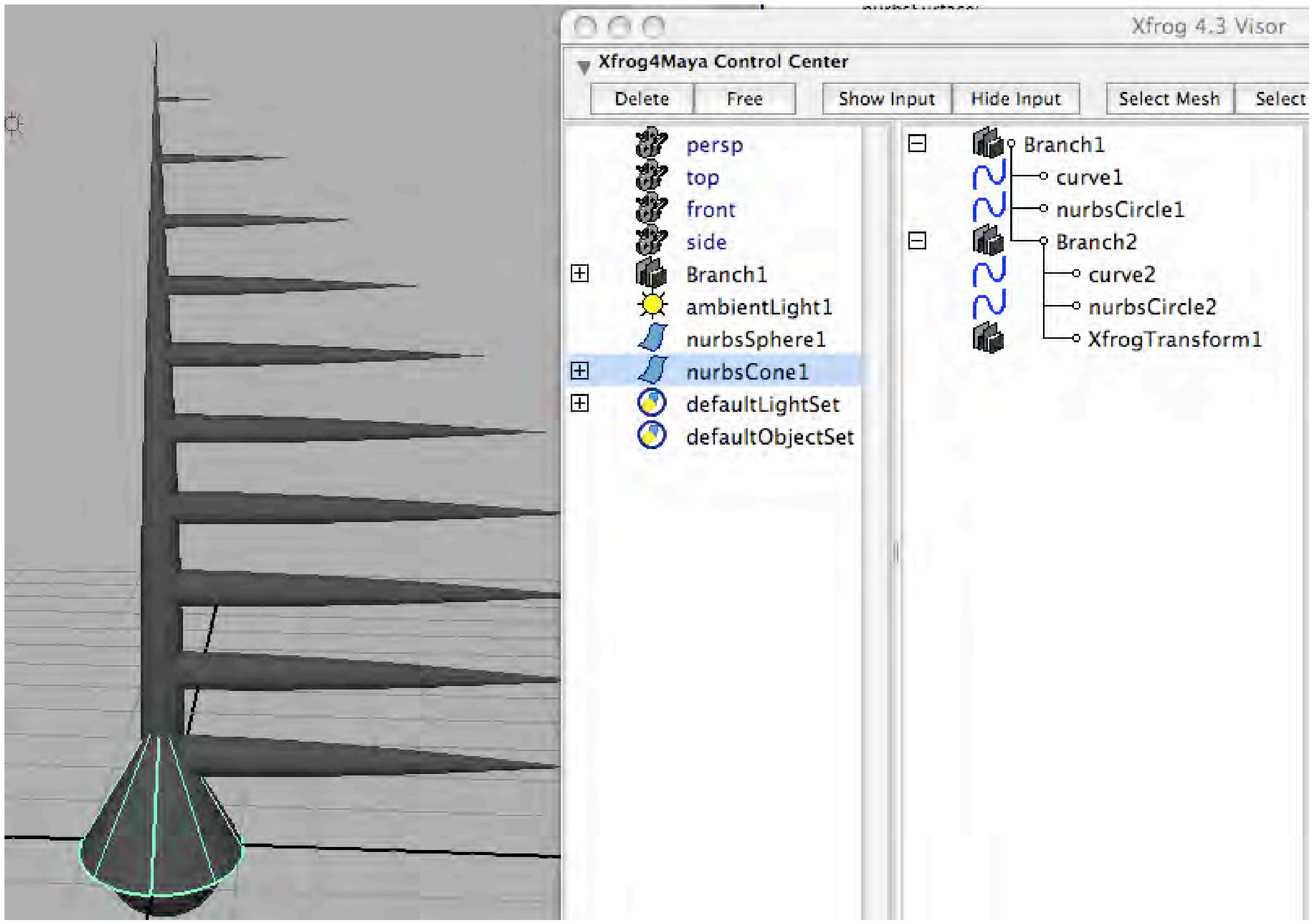
## 5. Access locations of single instance elements



*To access link-locations create a XfrogTransform object.*

*Accessing positions of single links (p.e. leaf-positions or branch-head-positions) was a feature often requested.*

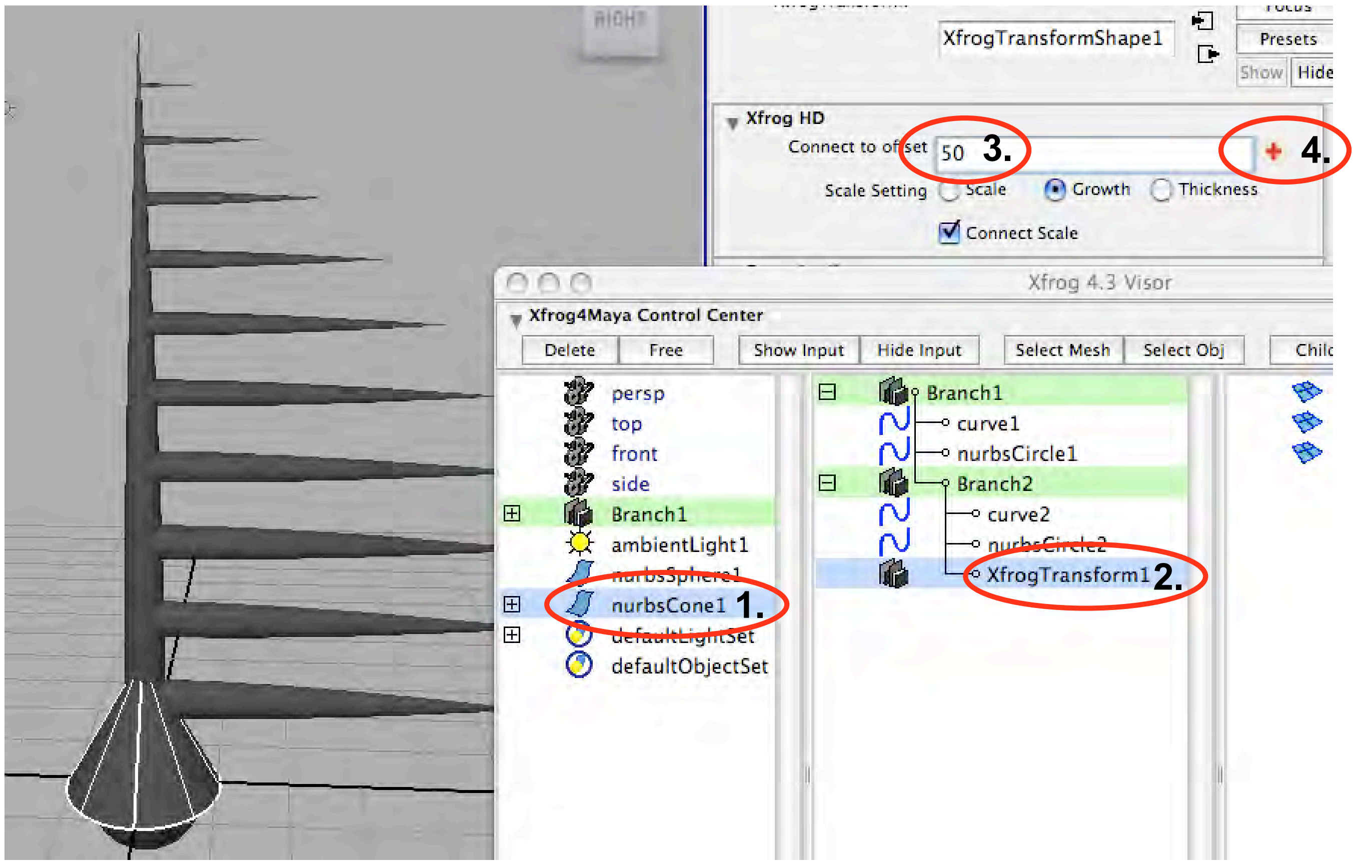
## 5. Access locations of single instance elements



*Add this XfrogTransform object as child/shape/head to your Xfrog object.*

*You can use the same buttons and funtions used to add existing Xfrog objects as child/shape/head to other Xfrog objects.*

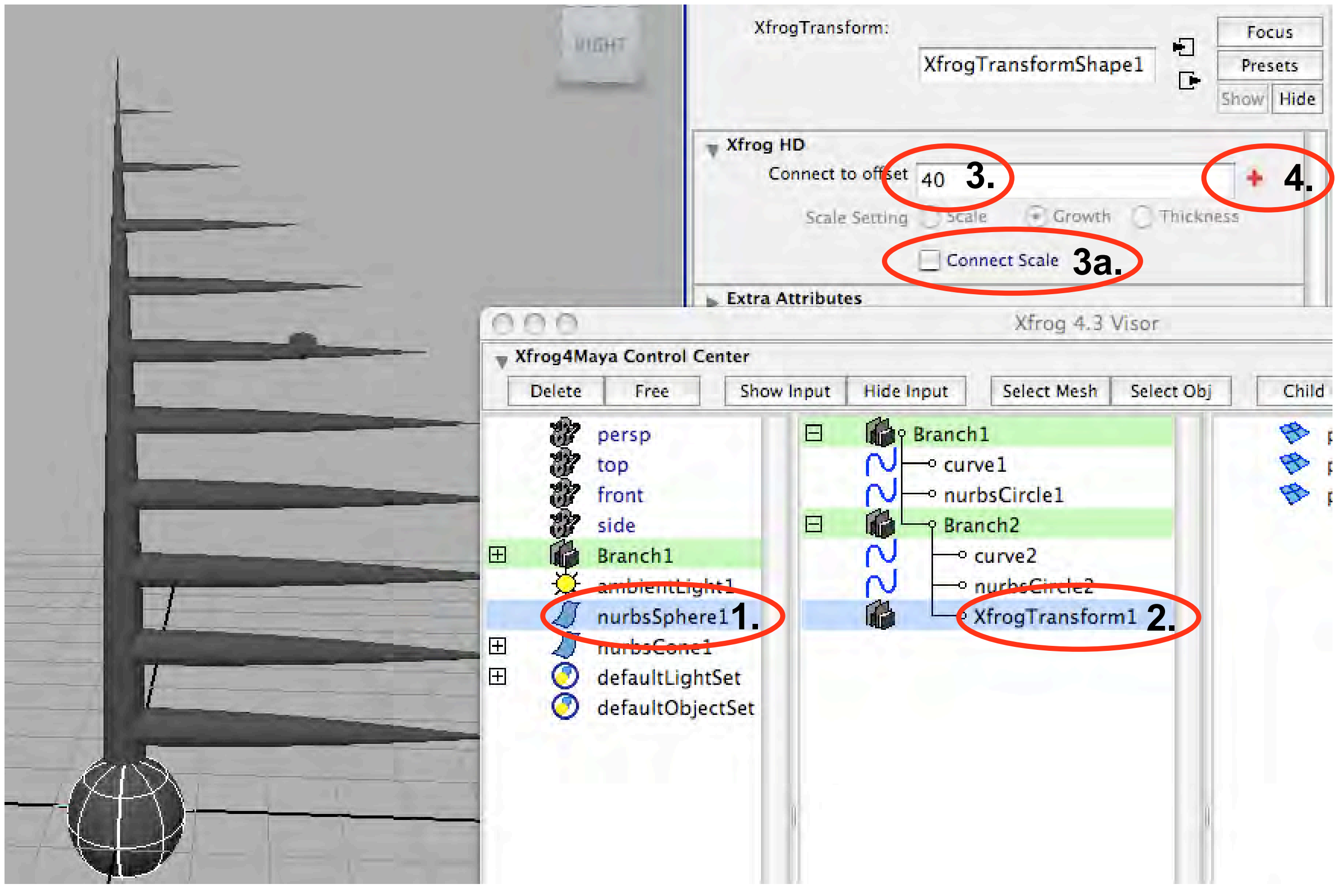
## 5. Access locations of single instance elements



*Now you can select any Transform-node(1). Then add-select the XfrogTransform(2). Set the Transform-node to be controlled(4) by the entered link-offset(3).*

*If you want to track head-positions, add the XfrogTransform as head. If you want to track child-positions, add the XfrogTransform as child.*

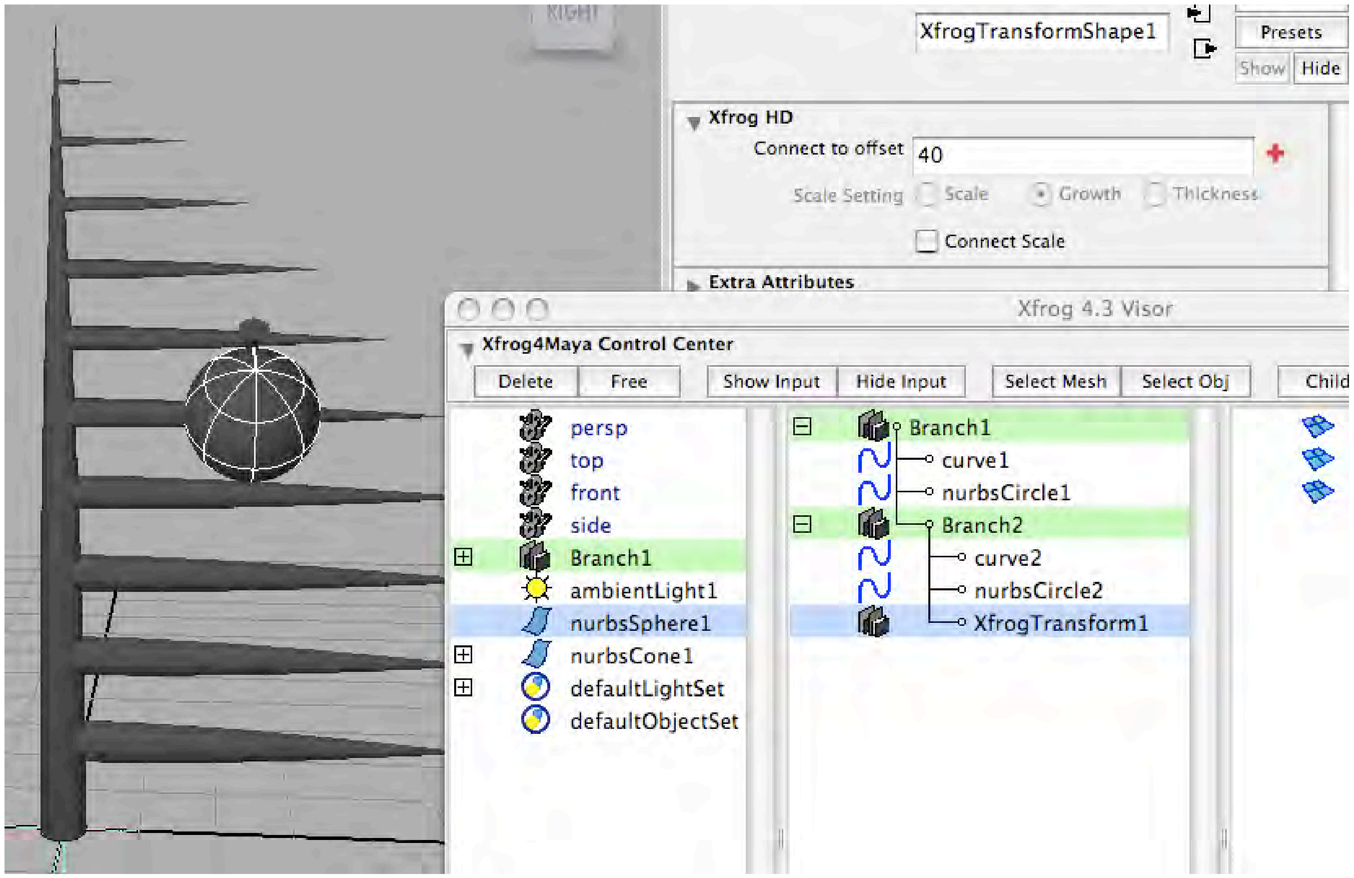
## 5. Access locations of single instance elements



*Decide how want to have the Transform node influenced(3a) by the link-position.*

*Simple move and rotate the Transform to the link-position (ConnectScale = off).*
*Move, rotate and scale the Transform to the link-position (ConnectScale = on).*
*Scale the Transform like a primitive attached to the link-position (ScaleSetting = Scale).*
*Scale the Transform like the growth set to a Xfrog object at the link-position (ScaleSetting = Growth).*
*Scale the Transform like the thickness set to a Xfrog object at the link-position (ScaleSetting = Thickness).*

## 5. Access locations of single instance elements


XfrogTransformShape1

Xfrog HD

Connect to offset 40

Scale Setting: Scale [✓ Growth] Thickness

Connect Scale

Extra Attributes

Xfrog 4.3 Visor

Xfrog4Maya Control Center

- persp
- top
- front
- side
- Branch1
- ambientLight1
- nurbsSphere1
- nurbsCone1
- defaultLightSet
- defaultObjectSet

```
Branch1
 ├─ curve1
 ├─ nurbsCircle1
 └─ Branch2
     ├─ curve2
     ├─ nurbsCircle2
     └─ XfrogTransform1
```

*Added Transform nodes are not integrated in the Xfrog hierarchy. It keep to be completely separate objects. They are just controlled by the link-location.*

*Have a look at the connection-graph to see how the XfrogTransform delivers location.*
*You can use the location-information of the Xfrog links in many different ways besides having primitives controlled by them.*

## 5. Access locations of single instance elements

This function was often requested to attach whatever to some branches heads or to track location of the branches leafs.

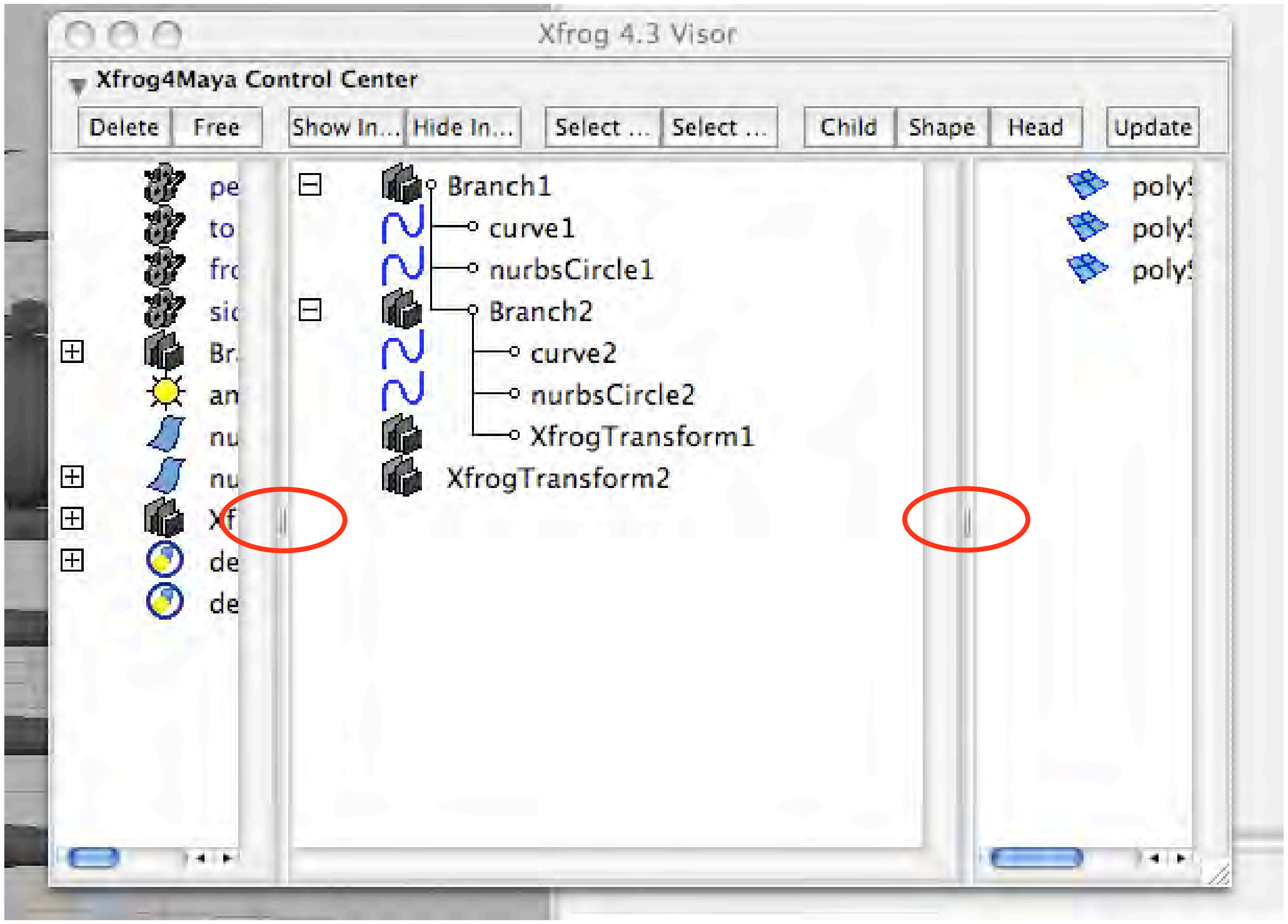
Each transform node can be used for this. The nodes are not integrated into the Xfrog hierarchy. The position just gets controlled by the link‘s position.

This way you can generate falling leafs in autumn:

Don‘t add leafs as child to the last branch.
Add a XfrogTransform as child to the last branch.
Generate the leafs by script and add them to the XfrogTransform by script (You know how - after looking at the connection-graph).
As required disconnect the leaf to fall down by script and use gravity.

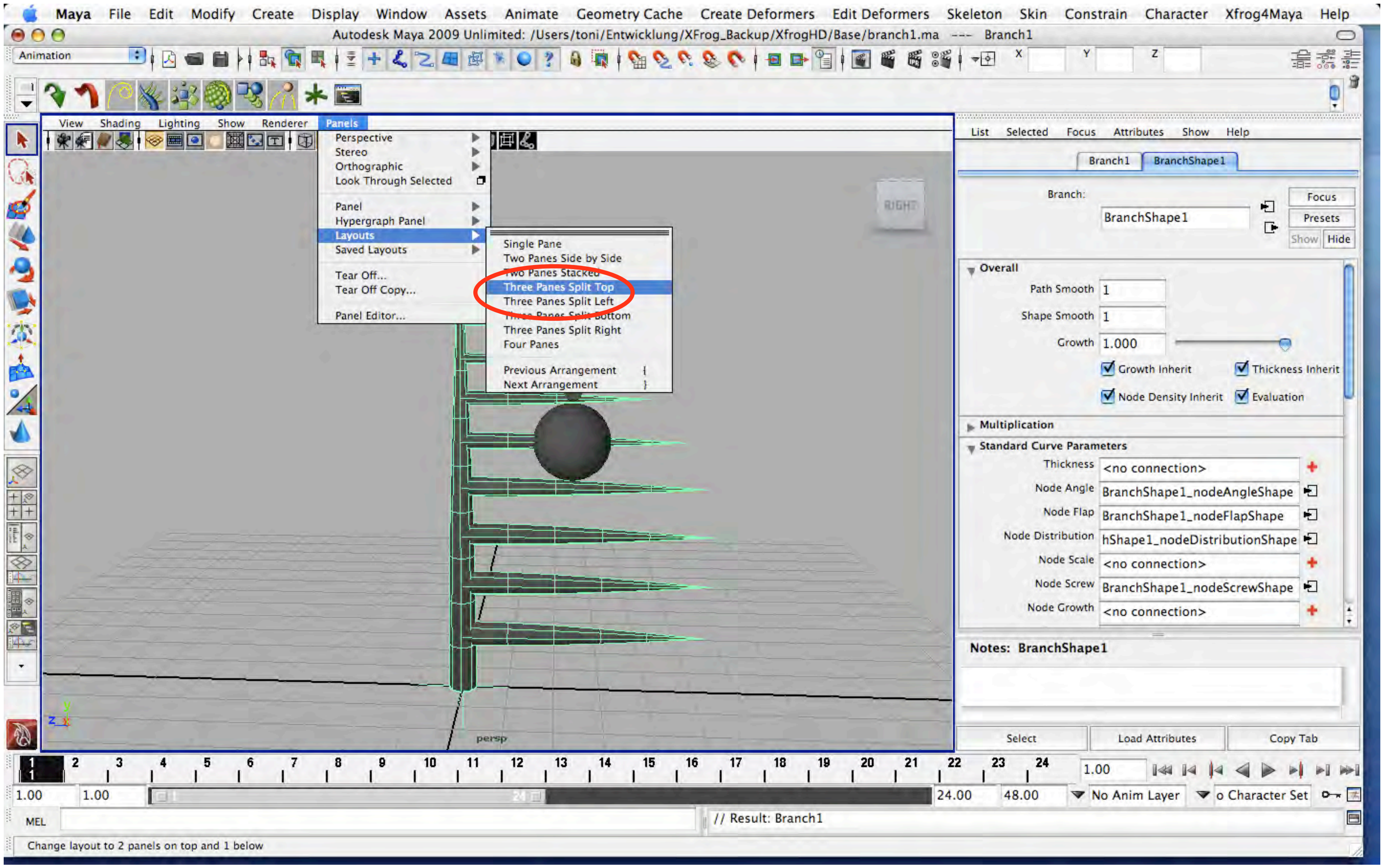
And you can imagine so much more using this feature...

## 6. Better usability



*Finally: Not new, but often requested - the visor-columns are movable now.*

## 6. Better usability



Autodesk Maya 2009 Unlimited: /Users/toni/Entwicklung/XFrog_Backup/XfrogHD/Base/branch1.ma --- Branch1

Branch: BranchShape1

Overall

Path Smooth 1

Shape Smooth 1

Growth 1.000

✓ Growth Inherit ✓ Thickness Inherit

✓ Node Density Inherit ✓ Evaluation

Multiplication

Standard Curve Parameters

| | |
|---|---|
| Thickness | <no connection> |
| Node Angle | BranchShape1_nodeAngleShape |
| Node Flap | BranchShape1_nodeFlapShape |
| Node Distribution | hShape1_nodeDistributionShape |
| Node Scale | <no connection> |
| Node Screw | BranchShape1_nodeScrewShape |
| Node Growth | <no connection> |

Notes: BranchShape1

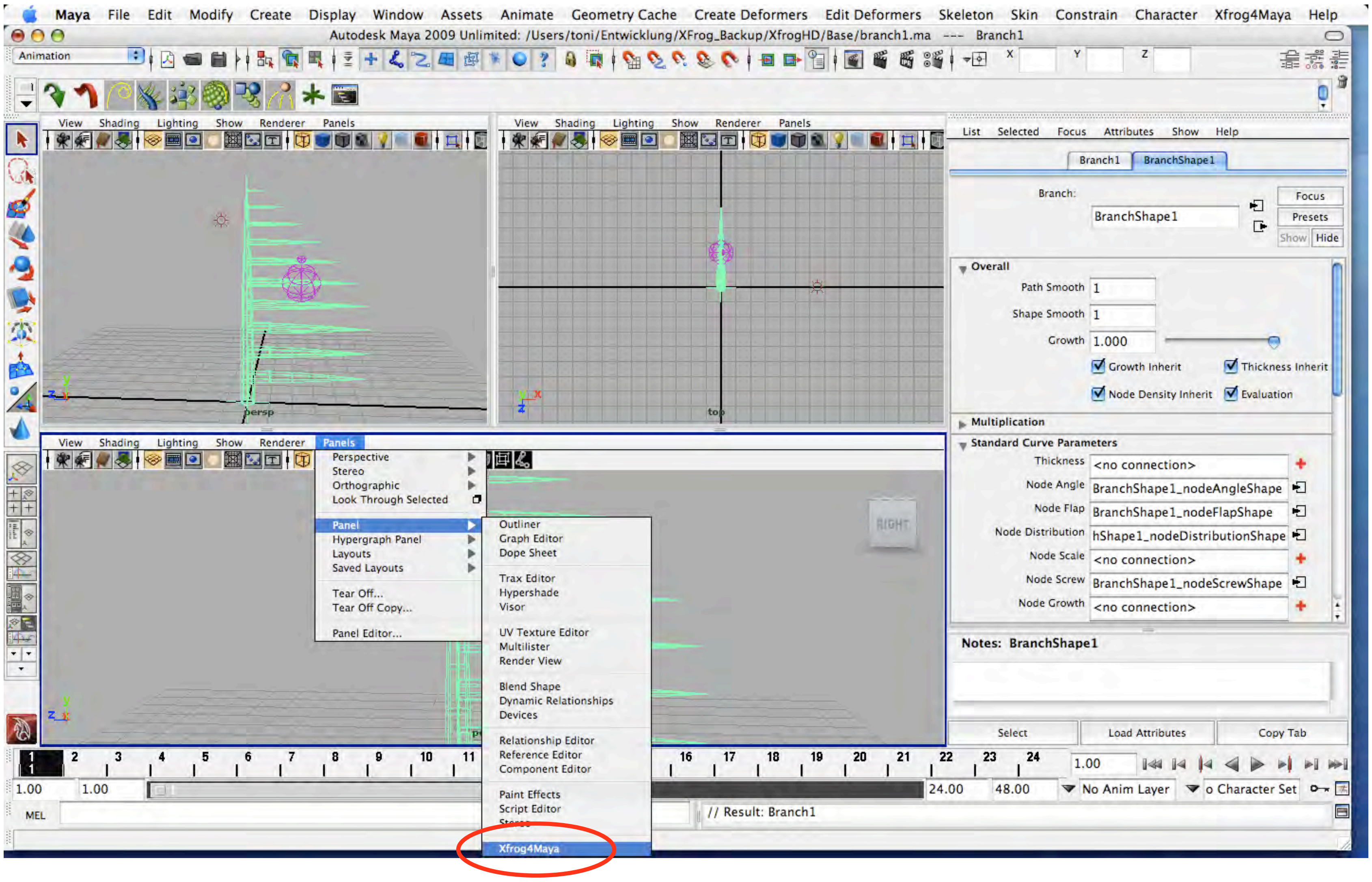
1.00 1.00 24.00 48.00

// Result: Branch1

Change layout to 2 panels on top and 1 below

*Finally: But new now - the visor can be integrated and docked as panel inside the UI.*

## 6. Better usability



Autodesk Maya 2009 Unlimited: /Users/toni/Entwicklung/XFrog_Backup/XfrogHD/Base/branch1.ma --- Branch1

Branch: BranchShape1

| | |
|---|---|
| Path Smooth | 1 |
| Shape Smooth | 1 |
| Growth | 1.000 |

✓ Growth Inherit ✓ Thickness Inherit
✓ Node Density Inherit ✓ Evaluation

| | |
|---|---|
| Thickness | <no connection> |
| Node Angle | BranchShape1_nodeAngleShape |
| Node Flap | BranchShape1_nodeFlapShape |
| Node Distribution | hShape1_nodeDistributionShape |
| Node Scale | <no connection> |
| Node Screw | BranchShape1_nodeScrewShape |
| Node Growth | <no connection> |

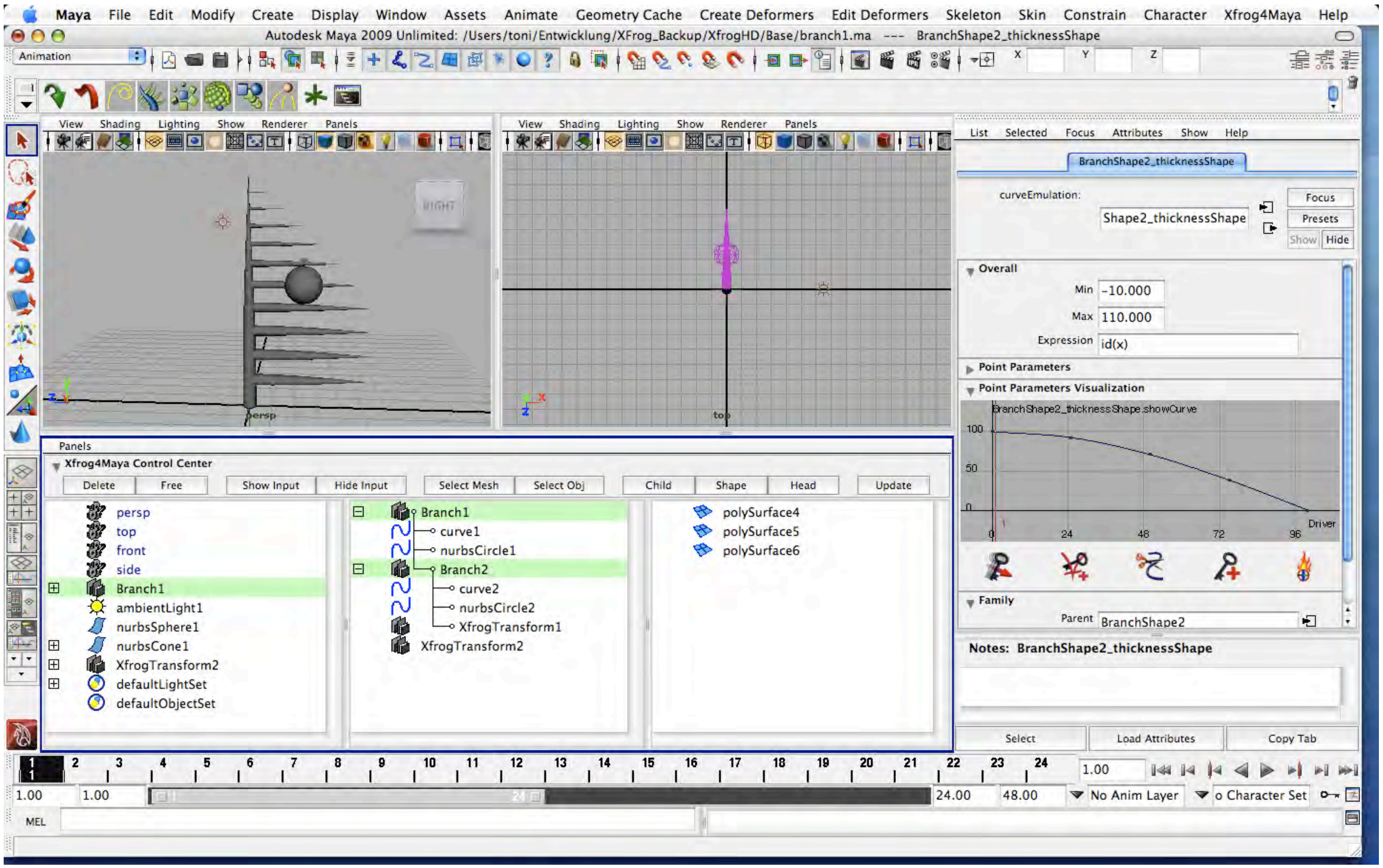
Notes: BranchShape1

1.00 1.00 24.00 48.00 1.00

// Result: Branch1

*Finally: But new now - the visor can be integrated and docked as panel inside the UI.*

## 6. Better usability


Maya File Edit Modify Create Display Window Assets Animate Geometry Cache Create Deformers Edit Deformers Skeleton Skin Constrain Character Xfrog4Maya Help
Animation
View Shading Lighting Show Renderer Panels
persp
top
List Selected Focus Attributes Show Help
BranchShape2_thicknessShape
Focus
Presets
Show Hide
Overall
Point Parameters
Point Parameters Visualization
BranchShape2_thicknessShape.showCurve
100
50
0
Driver
0
24
48
72
96
Family
Panels
Xfrog4Maya Control Center
Delete Free Show Input Hide Input Select Mesh Select Obj Child Shape Head Update
Select Load Attributes Copy Tab
No Anim Layer
o Character Set
MEL

Autodesk Maya 2009 Unlimited: /Users/toni/Entwicklung/XFrog_Backup/XfrogHD/Base/branch1.ma --- BranchShape2_thicknessShape

curveEmulation: Shape2_thicknessShape

Min -10.000
Max 110.000
Expression id(x)
Parent BranchShape2

Notes: BranchShape2_thicknessShape

```
persp                Branch1                polySurface4
top                    curve1               polySurface5
front                  nurbsCircle1         polySurface6
side                   Branch2
Branch1                  curve2
ambientLight1            nurbsCircle2
nurbsSphere1             XfrogTransform1
nurbsCone1           XfrogTransform2
XfrogTransform2
defaultLightSet
defaultObjectSet
```

1.00 1.00 24.00 48.00

*Yippieeeeehhhh....*

# Xfrog 5

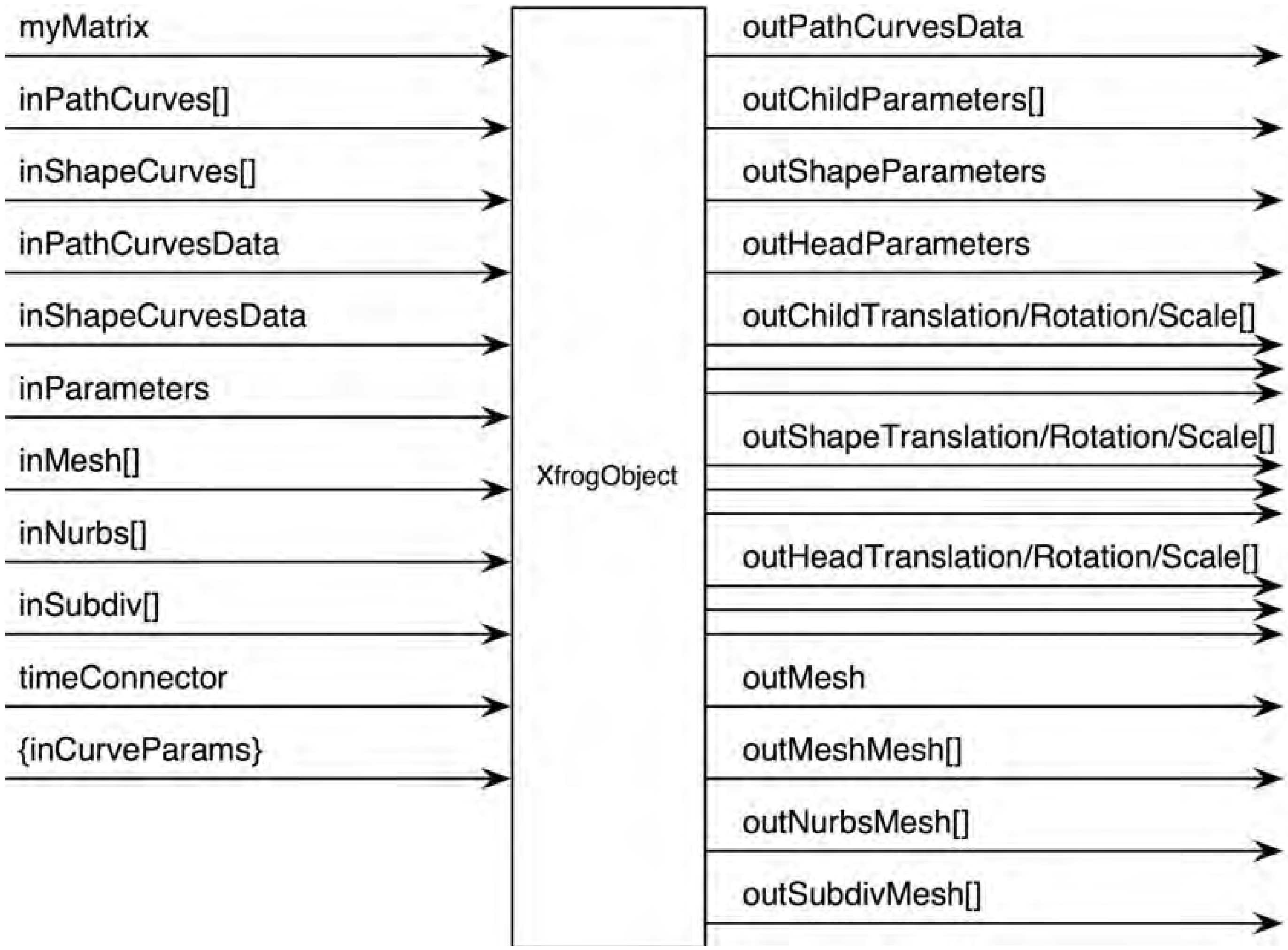
**Individual access and customization at single instance level**

Wow, a whole bunch of new functionality.
I think, this is what Xfrog always lacked.

Xfrog was always genially simple in making complex structures.
But multiplication ended at the object-level.
Having influence to the instance-level makes Xfrog the best solution for nearly all purposes.

Here we go!!!

## XfrogObject – In-/Out-Plug Overview



| Used Plugs / Object | CurveNurbs | Branch | Phyllo/Hydra | Variation | Tropism |
|---|---|---|---|---|---|
| myMatrix | x | x | x | - | - |
| inPathCurves[] | - | only [0] | only [0] | - | only [0] |
| inShapeCurves[] | - | only [0] | - | - | - |
| inPathCurvesData | - | x | x | - | x |
| inShapeCurvesData | - | x | - | - | - |
| inParameters | x | x | x | x | x |
| inMesh[] | - | x | x | x | - |
| inNurbs[] | - | x | x | x | - |
| inSubdiv[] | - | x | x | x | - |
| timeConnector | x | x | x | - | x |
| {inCurveParams} | x | x | x | - | x |
| outPathCurvesData | x | - | - | - | x |
| outChildParameters[] | - | only [0] | only [0] | x | - |
| outShapeParameters | - | x | - | - | - |
| outHeadParameters | - | x | - | - | - |
| outChildT/R/S[] | - | x | x | x | - |
| outShapeT/R/S[] | - | x | - | - | - |
| outHeadT/R/S[] | - | x | - | - | - |
| outMesh | - | x | - | - | - |
| outMeshMesh[] | - | x | x | x | - |
| outNurbsMesh[] | - | x | x | x | - |
| outSubdivMesh[] | - | x | x | x | - |

All listed plugs exist in all XfrogObjects. Usage of plugs not used for given objects may influence performance, but should not have any effect to the look of the object.

**myMatrix:**

This plug is needed to connect the XfrogObject to its assigned transform-nodes matrix-plug. This connection is needed to ensure updates to the object in case it is moved/rotated/scaled, because in most cases the objects form is influenced by its position in relation to XfrogObjects of higher levels.

**inPathCurves[]:**

Currently only the first element of this array is used to connect native Maya-spline-curves to the XfrogObjects using this plug. For Branch/Phyllotaxis/Hydra this curve is the parameter for the path the objects form follows. The Tropism gets its input-curve to be influenced via this plug. If the object receives input via the inPathCurvesData-plug, input from this plug is ignored. In most cases for correct functionality the native spline connected to this plug should be Maya-childed to the object this plug belongs to. Nevertheless there are cases imaginable where this may not be needed.

**inShapeCurves[]:**

Only the first element is currently used. Only the Branch-Object uses this plug to receive information from native Maya-spline-curves about the shape forming the Branch while following the given path. If the native spline must be influenced by a Tropism, the curve must be connected to the Tropisms inPathCurves[0]-plug and its outPathCurvesData-plug to this objects inShapeCurvesData-plug. If the object receives input via the inShapeCurvesData-plug, input from this plug is ignored. In most cases for correct functionality the native spline connected to this plug should be Maya-childed to the object this plug belongs to. Nevertheless there are cases imaginable where this may not be needed.

**inPathCurvesData:**

XfrogObjects using this plug (Branch, Phyllotaxis, Hydra, Tropism) receive input from splines with extended functionality (CurveNurbs, CurveNurbs or native Spline influenced by Tropism). The plug is used to define the path the objects form will follow.

**inShapeCurvesData:**

XfrogObjects using this plug (Branch) receive input from splines with extended functionality (CurveNurbs, CurveNurbs or native Spline influenced by Tropism). The plug is used to define the shape creating the objects form while following the given path.

**inParameters:**

This plug transfers all needed information from one XfrogObject to another XfrogObject assigned as child/shape/head-object to the first XfrogObject. In this connection the plug is used as the second XfrogObjects input. For correct functionality, the object, to which this plug is connected, must be the Maya-parent of the object this plug belongs to. All information passed to this plug is in relation to the positions matrix of the parent-object of the XfrogObject owning this plug.

**inMesh[]:**

Connections to this plug are being made from primitive mesh-geometry-objects delivering mesh-input forming the children of the XfrogObject using multiplication. For correct functionality the input-primitive should be child of the XfrogObject and be made invisible or intermediate. Additionally a polySurface-object should be connected to the corresponding outMeshMesh-plug to visualize the multiplied geometry. For Phyllotaxis and Hydra each connection means a multiplication as child. For a Branch each connection to inMesh[n*3] means multiplication as child, to inMesh[n*3+1] means multiplication as shape and to inMesh[n*3+2] means multiplication as head. For Variation-objects the type of multiplication depends on the number of Variations children. For example for a number of 4 children of the Variation, a connection to inMesh[0*4+2] means multiplication of the first primitive of the third child, a connection to inMesh[2*4+2] means multiplication of the third primitive of the third child. So the formula for connection of a primitive of the x-th child to a Variation with count children would be: inMesh[n*count+x].

**inNurbs[]:**

Connections to this plug are being made from primitive nurbs-geometry-objects delivering nurbs-input forming the children of the XfrogObject using multiplication. For correct functionality the input-primitive should be child of the XfrogObject and be made invisible or intermediate. Additionally a polySurface-object should be connected to the corresponding outNurbsMesh-plug to visualize the multiplied geometry. For Phyllotaxis and Hydra each connection means a multiplication as child. For a Branch each connection to inNurbs[n*3] means multiplication as child, to inNurbs[n*3+1] means multiplication as shape and to inNurbs[n*3+2] means multiplication as head. For Variation-objects the type of multiplication depends on the number of Variations children. For example for a number of 4 children of the Variation, a connection to inNurbs[0*4+2] means multiplication of the first primitive of the third child, a connection to inNurbs[2*4+2] means multiplication of the third primitive of the third child. So the formula for connection of a primitive of the x-th child to a Variation with count children would be: inNurbs[n*count+x].

**inSubdiv[]:**

Connections to this plug are being made from primitive subdiv-geometry-objects delivering subdiv-input forming the children of the XfrogObject using multiplication. For correct functionality the input-primitive should be child of the XfrogObject and be made invisible or intermediate. Additionally a polySurface-object should be connected to the corresponding outSubdivMesh-plug to visualize the multiplied geometry. For Phyllotaxis and Hydra each connection means a multiplication as child. For a Branch each connection to inSubdiv[n*3] means multiplication as child, to inSubdiv[n*3+1] means multiplication as shape and to inSubdiv[n*3+2] means multiplication as head. For Variation-objects the type of multiplication depends on the number of Variations children. For example for a number of 4 children of the Variation, a connection to inSubdiv[0*4+2] means multiplication of the first primitive of the third child, a connection to inSubdiv[2*4+2] means multiplication of the third primitive of the third child. So the formula for connection of a primitive of the x-th child to a Variation with count children would be: inSubdiv[n*count+x].

**timeConnector:**

This plug is needed, if some input-CurveParameters are animated using expression instead of keyframing. Because the CurveParameters expression is evaluated inside the XfrogObject and not inside the CurveParam itself, an update to such XfrogObjects has to be made each frame. To achieve this, easily connect the global-time-nodes outTime to this plug.

**{inCurveParams}:**

The number of needed CurveParams is different for each kind of XfrogObject. A CurveParam is a separate object to be connected using one of the various inCurveParam-plugs.

**outPathCurvesData:**

This plug provides geometry of splines with extended functionality (CurveNurbs, CurveNurbs or native Spline influenced by Tropism). So CurveNurbs or Tropism can deliver input for

Branches, Phyllotaxises, Hydras or other Tropisms making a connection from this plug to their inPathCurvesData- or inShapeCurvesData-plug.

**outChildParameters[]:**

This plug provides information about the location, instance-number etc. of XfrogObjects children. All information passed by this plug is relative to the matrix of the XfrogObject owning this plug. So any XfrogObject being connected to this plug using its inParameters-plug should be a child of the XfrogObject owning this plug. For Branch, Phyllotaxis, Hydra only the first element is used, for Variation, the number of existing elements is equal to the number of Variations children.

**outShapeParameters:**

This plug provides information about the location, instance-number etc. of XfrogObjects shape-objects. All information passed by this plug is relative to the matrix of the XfrogObject owning this plug. So any XfrogObject being connected to this plug using its inParameters-plug should be a child of the XfrogObject owning this plug. This plug is only valid for Branch.

**outHeadParameters:**

This plug provides information about the location, instance-number etc. of XfrogObjects head-objects. All information passed by this plug is relative to the matrix of the XfrogObject owning this plug. So any XfrogObject being connected to this plug using its inParameters-plug should be a child of the XfrogObject owning this plug. This plug is only valid for Branch.

**outChildT/R/S[]:**

Multiplication of primitive children can be done in many ways. One possible solution is the geometry-multiplication using the inMesh/inNurbs/inSubdiv to out…Mesh combination. Another way can be copying or instancing the primitives and connecting their transform-node to this plug. Because this plug directly influences the transform-matrix, the primitive's transform-node should be grouped under another transform-node, which is connected then to this plug. So the flexibility to move the primitive can be maintained. The primitive should be child of the XfrogObject owning this plug, because all information provided is relative to the XfrogObjects matrix. For Branch, Phyllotaxis and Hydra each element delivers a matrix of an instance of a child. For Variation-objects the matrix-assignment delivered by an element depends on the number of Variations children. For example for a number of 4 children of the Variation, a connection to outChild…[0*4+2] means delivery of the first instances matrix of the third child, a connection to outChild…[2*4+2] means delivery of the third instances matrix of the third child. So the formula for delivery of the n-th instances matrix of the x-th child of a Variation with count children would be: outChild…[n*count+x].

**outShapeT/R/S[]:**

Multiplication of primitive shape-objects can be done in many ways. One possible solution is the geometry-multiplication using the inMesh/inNurbs/inSubdiv to out…Mesh combination. Another way can be copying or instancing the primitives and connecting their transform-node to this plug. Because this plug directly influences the transform-matrix, the primitive's transform-node should be grouped under another transform-node, which is connected then to this plug. So the flexibility to move the primitive can be maintained. The primitive should be child of the XfrogObject owning this plug, because all information provided is relative to the XfrogObjects matrix.

**outHeadT/R/S[]:**

Multiplication of primitive head-objects can be done in many ways. One possible solution is the geometry-multiplication using the inMesh/inNurbs/inSubdiv to out…Mesh combination. Another way can be copying or instancing the primitives and connecting their transform-node to this plug. Because this plug directly influences the transform-matrix, the primitive's transform-node should be grouped under another transform-node, which is connected then to this plug. So the flexibility to move the primitive can be maintained. The primitive should be child of the XfrogObject owning this plug, because all information provided is relative to the XfrogObjects matrix.

**outMesh:**

This plug provides mesh-geometry-information provided by a Branch formed by a path- and a shape-spline. This plug should be connected to a polySurface-node for correct functionality.

**outMeshMesh[]:**

This plug forms the counterpart to the inMesh[]-plug. Each primitive connected to the inMesh-plug at element n is multiplied and its multiplied geometry is output via this plugs element n. For correct functionality for each n, where a primitive is connected at inMesh, a polySurface-node should be connected to this plug to visualize the information.

**outNurbsMesh[]:**

This plug forms the counterpart to the inNurbs []-plug. Each primitive connected to the inNurbs-plug at element n is multiplied and its multiplied geometry is output via this plugs element n. For correct functionality for each n, where a primitive is connected at inNurbs, a polySurface-node should be connected to this plug to visualize the information.

**outSubdivMesh[]:**

This plug forms the counterpart to the inSubdiv[]-plug. Each primitive connected to the inSubdiv-plug at element n is multiplied and its multiplied geometry is output via this plugs element n. For correct functionality for each n, where a primitive is connected at inSubdiv, a polySurface-node should be connected to this plug to visualize the information.

All connectivity between XfrogObjects and other XfrogObjects or primitives can be done by hand following the instructions given above. For convenience for all of these connection-operations exist UI-combinations known from some workflow-examples or how-tos and also MEL-Commands to do the job. Later some examples are given to see, how connections known from tutorials, where they were made by UI, can also be done by hand or via MEL-calls.

# Xfrog – MEL-Command Overview

**aboutXF:**
Simple command, that shows Xfrog-About-Box.
*Syntax: aboutXF;*
*Return: none*

**XFCreateCurveEmulation:**
Creates curveEmulation-node (Xfrog CurveParam). These nodes are used as input from XfrogObjects. For UI-use they are generated automatically, but the can be generated separately using this command.
*Syntax: XFCreateCurveEmulation;*
*Return: Name of curveEmulations transform-node*

**XFCreateCurveNurbs;**
Creates CurveNurbs-node (Xfrog Curvature) and structure around. In this case simply a spline for visualzation-purposes.
*Syntax: XFCreateCurveNurbs;*
*Return: Name of CurveNurbs' transform-node*

**XFCreateBranch:**
Creates Branch-Node and structure around. In this case transform-nodes-structure to keep added child/shape/head-primitives organized, a standard path-, a standard shape-spline and a polyShape for mesh-output.
*Syntax: XFCreateBranch;*
*Return: Name of Branchs transform-node*

**XFCreateTropism:**
Creates Tropism-node.
*Syntax: XFCreateTropism;*
*Return: Name of Tropisms transform-node*

**XFCreateVariation:**
Creates Variation-Node and structure around. In this case transform-nodes-structure to keep added

child-primitives organized.
*Syntax: XFCreateVariation;*
*Return: Name of Variations transform-node*

**XFCreatePhyllotaxis:**
Creates Phyllotaxis-Node and structure around. In this case transform-nodes-structure to keep added child-primitives organized and a standard path-spline (half circle).
*Syntax: XFCreatePhyllotaxis;*
*Return: Name of Phyllotaxis' transform-node*

**XFCreateHydra:**
Creates Hydra-Node and structure around. In this case transform-nodes-structure to keep added child-primitives organized.
*Syntax: XFCreateHydra;*
*Return: Name of Hydras transform-node*

**XFConnectCurveAttr:**
Connects a curveEmulation (CurveParam) –node to a defined CurveParamIn-plug of an XfrogObject.
Syntax: XFConnectCurveAttr curveEmulationName XfrogObjectName XfrogObjectParamInPlugName;
Return: Name of curveEmulations transform-node after connection
It is allowed to use the names of either the transform-node or the shape-node of the curveEmulation. The name of the shape-node must be given for the XfrogObject. The name must be unique. If not, the full dag-path must be given as name. It is also allowed to, having selected nothing, select first the curveEmulation-node, then add-select the XfrogObject-node and then execute the command using the syntax:
*Syntax: XFConnectCurveAttr XfrogObjectParamInPlugName;*
*Return: Name of curveEmulations transform-node after connection*
The standard parameters valid for the given plug are set to the curveEmulation-node, which can be further used to edit the CurveParam.

**XFConnectChild:**
Connects an XfrogObject or primitive geometry-object as child, shape or head to an XfrogObject.
*Syntax: XFConnectChild childName parentName childType;*
*Return: Name of child's transform-node after connection*
The parentName can be the transform- or shape-node of a Branch, Phyllotaxis, Hydra or Variation. The childName can be the transform-node or the shape-node of Branch, Phyllotaxis, Hydra, Variation or any object or group of objects only consisting of transform-, mesh-, nurbs- and subdiv-nodes. If the parentObjects multiplication-type is set to instance or copy, additionally any object or group of objects not containing any XfrogObjects can be used as primitive geometry. (This way, it is possible to add PaintFX-branches and –leafs to trees).

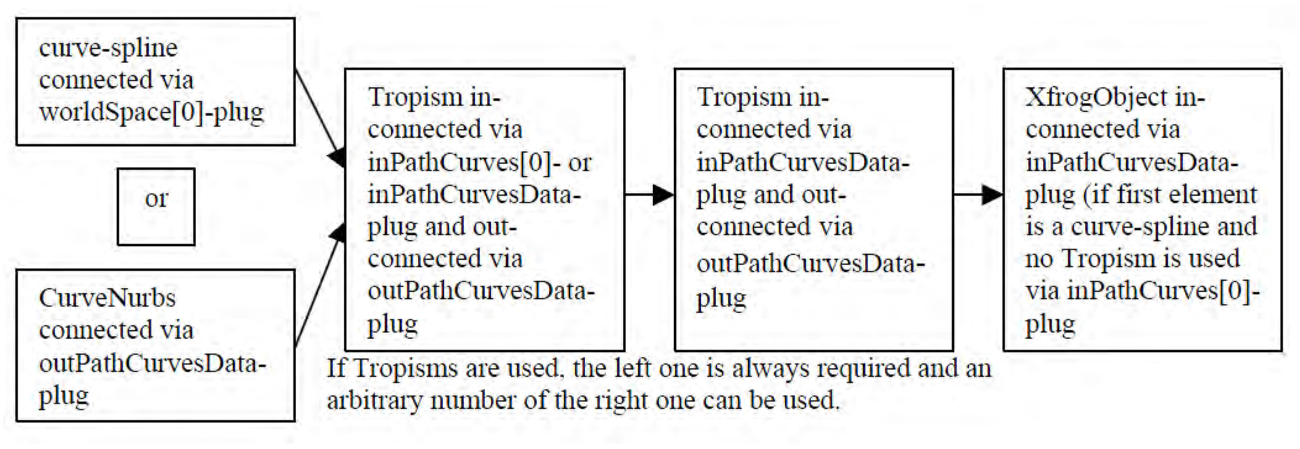
*Explanation:*
*If the given child is an XfrogObject, it is childed under the parent XfrogObject and its inParameters-plug is connected to the parent-out…Parameters-plug defined by childType. For primitives applies the following: If the parentObjects multiplication-type is set to instance or copy the primitives are childed somehow and copied or instanced by the needed child-count and the top-transform-nodes of the created instances/copies are connected to the parents out…T/R/S-plugs defined by childType. If the multiplication-type is set to mesh otherwise, all mesh-, subdiv- and nurbs-nodes in the given primitive object or group of primitives are childed to parent somehow and connected to the associated inMesh-, inNurbs- and inSubdiv-plugs and for each an also childed polyShape-node is created and connected to the concurrent out…-plugs.*

The childType defines the way the objects are connected. For Hydra and Phyllotaxis only 0 for child is allowed. For Branch any number between 0 and 2 is allowed, 0 for child, 1 for shape and 2 for head. For Variation any number between 0 and number of Variations childs –1 is allowed, defining the childNumber to be used for the child. If the children were of primitive type, they are made invisible to be just used as geometry-source. It is also allowed to, having nothing selected, select first the childNode, the add-select the parentNode and then execute the command using the syntax:
*Syntax: XFConnectChild childType;*
*Return: Name of child's transform-node after connection*

**XFConnectCurve:**

Connects Tropisms to splines with extended functionality (CurveNurbs, CurveNurbs or native Spline influenced by Tropism) or standard nurbs-splines or these to XfrogObjects.

To understand the behaviour of this command it is needed to know how the curve-information-flow works in the dependency graph.


curve-spline connected via worldSpace[0]-plug
or
CurveNurbs connected via outPathCurvesData-plug
Tropism in-connected via inPathCurves[0]- or inPathCurvesData-plug and out-connected via outPathCurvesData-plug
Tropism in-connected via inPathCurvesData-plug and out-connected via outPathCurvesData-plug
XfrogObject in-connected via inPathCurvesData-plug (if first element is a curve-spline and no Tropism is used via inPathCurves[0]-plug
If Tropisms are used, the left one is always required and an arbitrary number of the right one can be used.

To connect curves (with or without Tropisms attached) to an XfrogObject the following syntax can be used:

*Syntax: XFConnectCurve curveName objectName connectionType;*

*Return: Name of the curve's transform-node after connection*

The curveName can be the name of the transform- or the shape-node of a nurbs-spline or a Curve-Nurbs. The objectName can be the transform-or the shape-node of a Branch or a Phyllotaxis. The connectionType can be of value 0 for Branches and Phyllotaxis' to connect the given spline as path-spline and of value 1 for Branches to connect the given spline as shape-spline. It is also allowed to, having nothing selected, first select the curve, the add-select the XfrogObject and to use the following syntax then:

*Syntax: XFConnectCurve connectionType;*

*Return: Name of the curve's transform-node after connection*

To connect a free Tropism to curves (with or without Tropisms and Object attached) the following syntax can be used.

*Syntax: XFConnectCurve curveName_or_boundTropismName freeTropismName 0;*

*Return: Name of the object in information-flow, the freeTropism was inserted after*

The Tropism to be connected needs to be free and can be given by its shape- or transform-node. The object to connect the free tropism to can be either a curve or a Tropism directly or via other Tropisms connected to a curve. In the information-flow seen above, the free Tropism is attached or inserted after the curve or Tropism given by name. This can be given by transform- or shape-node. The returned object-name is the name of the object the freeTropism was inserted after. To receive the eventually new Dag-name of the former free- and now boundTropism follow the connection of the outPathCurvesData-plug of the returned object-name. It is also allowed to, having nothing selected, first select the curve or boundTropism, then add-select the freeTropism and to use the following syntax for connection then:

*Syntax: XFConnectCurve 0;*

*Return: Name of the object in information-flow, the freeTropism was inserted after*

**XFFreeObject:**

This command is used to group the selected object under world and to cleanly remove any connections to XfrogObjects. For example a boundTropism is grouped under world, its connections to binding-partners are broken and the object in the dg-flow before and the object in the dg-flow after are connected. To give another example, a primitive connected to an XfrogObject is disconnected and grouped under world. Its connection to the XfrogObject is broken and the polyShape connected to concurrent plug for visualization is deleted.

*Syntax: XFFreeObject objectName;*

*Return: Name of the object grouped under world*

If the given object is an XfrogObject (which may have other XfrogObjects or primitives childed to it), it can be given by transform- or shape-node, otherwise the top-level transform-node of an object or group containing no XfrogObjects is expected. It is also allowed to, having nothing selected, select the object to free and use the following syntax for execution then:

*Syntax: XFFreeObject;*

*Return: Name of the object grouped under world*

**XFUpdateObject:**

This command is used to clean up XfrogObjects having any of the update…-plug values set to true. All XfrogObjects childed to the given XfrogObject are cleaned up too.

*Syntax: XFUpdateObject objectName;*

*Return: none*

The XfrogObject can be given by shape- or transform-node. This command is needed, after a change to a Variations number of children using the Variations requestedChildCount-plug, to keep existing children connected to the correct plugs. Also after freeing an XfrogObject formerly childed to another one, the command may be needed to update the changed number of primitive children using copies or instances. All Xfrog-MEL-commands automatically use this command, so it should be not needed directly after child-connection or freeing. Nevertheless it can be useful, if a polySurface-node to visualize multiplied primitive input is missing or the texture of a primitive input was changed and is needed to be updated at the visualization-surface too. This is done too by this command. It is also allowed to, having nothing selected use this command after selection of the object to be updated with the following syntax:

*Syntax: XFUpdateObject;*

*Return: none*

**XFSelect:**

To switch a relatively slowly updating XfrogObject, with in most cases a lot of connected Xfrog- and primitive children to a faster updating simple mesh, this command can be used. It recursively selects all shape-nodes of XfrogObjects starting by a given top-level-node. If these are deleted then, the remaining objects are simply the primitive mesh. It is also possible to select all meshes generated by XfrogObjects below given node. This may be needed for texturing or similar purposes. It means then, that all meshes forming Branches geometry and all copies, instances and multiplication visualizing polySurfaces of used primitives are selected. Further it is possible to decide, if the selection to be made shall be executed rejecting current selection or in addition to the current selection.

*Syntax: XFSelect -mesh/-object -add/-replace objectName;*

*Return: none*

The top-level XfrogObject can be given by shape- or transform-node. It is also allowed to, having nothing selected use this command after selection of the top-level-object with the following syntax:

*Syntax: XFSelectObject -mesh/-object -replace;*

*Return: none*

**XFInputVisibility:**

All primitives used as input by XfrogObjects are hidden by default delivering just the geometry for the multiplied surfaces, copies or instances. For editing purposes it may be needed to show or hide them during the process of editing. To show or hide the input-primitives for a given XfrogObject use the following syntax:

*Syntax: XFInputVisibility -show/-hide objectName;*

*Return: none*

The XfrogObject can be given by shape- or transform-node. It is also allowed to, having nothing selected use this command after selection of the XfrogObject with the following syntax:

*Syntax: XFInputVisibility -show/-hide;*

*Return: none*

## Xfrog – Examples of non-UI-Operation

As said in former chapters, all Xfrog-operations can be made without UI using MEL-commands. This is presented in examples here. It is also not required to use Xfrog-MEL, everything can be made using standard-MEL. For chosen examples this is demonstrated here too. It is important to know, that the structure generated by example-standard-MEL is not equal to structure generated by Xfrog-MEL although the results are. It is possible to also generate equal structure, but far too complex for these examples. By making the examples with Xfrog-MEL and then looking at the generated structure using Hypergraph, the way to do this can be understood.

**Adding CurveParam:**

```
$branch1 = `XFCreateBranch`;
$curveEmulation1 = `XFCreateCurveEmulation`;
string $branchKids[] = `listRelatives $branch1`;
XFConnectCurveAttr $curveEmulation1 $branchKids[0] thickness;
```

**Connecting XfrogObject to another as child:**

```
$branch1 = `XFCreateBranch`;
$branch2 = `XFCreateBranch`;
XFConnectChild $branch2 $branch1 0;
```

or shorter:

```
XFConnectChild `XFCreateBranch` `XFCreateBranch` 0;
```

**Connecting XfrogObject to another as child without Xfrog-MEL:**

```
$branch1 = `createNode Branch`;
$branchParent1 = `listRelatives -p $branch1`;
$shape1 = `circle -c 0 0 0 -nr 0 1 0 -sw 360 -r 1 -d 3 -ut 0 -tol 0.01 -s 8 -ch 1`;
$path1 = `curve -d 3 -p 0 0 0 -p 0 0.33 0 -p 0 1 0 -p 0 2 0 -p 0 3 0 -p 0 4 0 -p 0 5 0 -p 0 6 0 -p 0 7 0
-p 0 8 0 -p 0 9 0 -p 0 9.66 0 -p 0 10 0 -k 0 -k 0 -k 0 -k 1 -k 2 -k 3 -k 4 -k 5 -k 6 -k 7 -k 8 -k 9 -k 10 -k
10 -k 10`;
$mesh1 = `createNode -p $branchParent1[0] mesh`;
$branch2 = `createNode Branch`;
$branchParent2 = `listRelatives -p $branch2`;
$shape2 = `circle -c 0 0 0 -nr 0 1 0 -sw 360 -r 1 -d 3 -ut 0 -tol 0.01 -s 8 -ch 1`;
$path2 = `curve -d 3 -p 0 0 0 -p 0 0.33 0 -p 0 1 0 -p 0 2 0 -p 0 3 0 -p 0 4 0 -p 0 5 0 -p 0 6 0 -p 0 7 0
-p 0 8 0 -p 0 9 0 -p 0 9.66 0 -p 0 10 0 -k 0 -k 0 -k 0 -k 1 -k 2 -k 3 -k 4 -k 5 -k 6 -k 7 -k 8 -k 9 -k 10 -k
10 -k 10`;
$mesh2 = `createNode -p $branchParent2[0] mesh`;
connectAttr ($shape1[0]+”.worldSpace[0]”) ($branch1+”.inShapeCurves[0]”);
connectAttr ($path1+”.worldSpace[0]”) ($branch1+”.inPathCurves[0]”);
connectAttr ($branch1+”.outMesh[0]”) ($mesh1+”.inMesh”);
connectAttr ($shape2[0]+”.worldSpace[0]”) ($branch2+”.inShapeCurves[0]”);
connectAttr ($path2+”.worldSpace[0]”) ($branch2+”.inPathCurves[0]”);
connectAttr ($branch2+”.outMesh[0]”) ($mesh2+”.inMesh”);
connectAttr ($branch1+”.outChildParameters[0]”) ($branch2+”.inParameters”);
parent $shape1[0] $branchParent1;
parent $path1 $branchParent1;
parent $shape2[0] $branchParent2;
parent $path2 $branchParent2;
parent $branchParent2 $branchParent1;
```

**Connecting primitive object to XfrogObject as child:**

```
$branch1 = `XFCreateBranch`;
$primitive1 = ` torus -p 0 0 0 -ax 0 1 0 -ssw 0 -esw 360 -msw 360 -r 1 -hr 0.5 -d 3 -ut 0 -tol 0.01 -s 8
-nsp 4 -ch 1`;
XFConnectChild $primitive1[0] $branch1 1;
```

**Connecting primitive object to XfrogObject as child without Xfrog-MEL:**

```
$branch1 = `createNode Branch`;
$branchParent1 = `listRelatives -p $branch1`;
$shape1 = `circle -c 0 0 0 -nr 0 1 0 -sw 360 -r 1 -d 3 -ut 0 -tol 0.01 -s 8 -ch 1`;
$path1 = `curve -d 3 -p 0 0 0 -p 0 0.33 0 -p 0 1 0 -p 0 2 0 -p 0 3 0 -p 0 4 0 -p 0 5 0 -p 0 6 0 -p 0 7 0
-p 0 8 0 -p 0 9 0 -p 0 9.66 0 -p 0 10 0 -k 0 -k 0 -k 0 -k 1 -k 2 -k 3 -k 4 -k 5 -k 6 -k 7 -k 8 -k 9 -k 10 -k
10 -k 10`;
$mesh1 = `createNode -p $branchParent1[0] mesh`;
connectAttr ($shape1[0]+”.worldSpace[0]”) ($branch1+”.inShapeCurves[0]”);
connectAttr ($path1+”.worldSpace[0]”) ($branch1+”.inPathCurves[0]”);
connectAttr ($branch1+”.outMesh[0]”) ($mesh1+”.inMesh”);
$primitive1 = `torus -p 0 0 0 -ax 0 1 0 -ssw 0 -esw 360 -msw 360 -r 1 -hr 0.5 -d 3 -ut 0 -tol 0.01 -s 8
-nsp 4 -ch 1`;
connectAttr ($primitive1[0]+”.worldSpace[0]”) ($branch1+”.inNurbs[1]”);
$nurbsMesh1 = `createNode -p $branchParent1[0] mesh`;
connectAttr ($branch1+”.outNurbsMesh[1]”) ($nurbsMesh1+”.inMesh”);
parent $shape1[0] $branchParent1;
parent $path1 $branchParent1;
parent $primitive1[0] $branchParent1;
```

**Connecting CurveNurbs to XfrogObject:**

```
$branch1 = `XFCreateBranch`;
delete `listConnections ($branch1+”.inPathCurves[0]”)`;
$curve1 = `XFCreateCurveNurbs`;
XFConnectCurve $curve1 $branch1 0;
```

**Connecting CurveNurbs to XfrogObject without Xfrog-MEL:**

```
$branch1 = `createNode Branch`;
$branchParent1 = `listRelatives -p $branch1`;
$shape1 = `circle -c 0 0 0 -nr 0 1 0 -sw 360 -r 1 -d 3 -ut 0 -tol 0.01 -s 8 -ch 1`;
$curve1 = `createNode CurveNurbs`;
$curveParent1 = `listRelatives -p $curve1`;
$mesh1 = `createNode -p $branchParent1[0] mesh`;
connectAttr ($shape1[0]+”.worldSpace[0]”) ($branch1+”.inShapeCurves[0]”);
connectAttr ($curve1+”.outPathCurvesData”) ($branch1+”.inPathCurvesData”);
connectAttr ($branch1+”.outMesh[0]”) ($mesh1+”.inMesh”);
parent $shape1[0] $branchParent1;
parent $curveParent1 $branchParent1;
```

**Connecting Tropism to curve of previous example:**

```
$branch1 = `XFCreateBranch`;
delete `listConnections ($branch1+”.inPathCurves[0]”)`;
$curve1 = `XFCreateCurveNurbs`;
$tropism1 = `XFCreateTropism`;
XFConnectCurve $curve1 $tropism1 0;
XFConnectCurve $curve1 $branch1 0;
```

**Complex scripting example:**

```
$branch1 = `XFCreateBranch`;
delete `listConnections ($branch1+”.inPathCurves[0]”)`;
$curve1 = `XFCreateCurveNurbs`;
$tropism1 = `XFCreateTropism`;
setAttr ($tropism1+”.axis”) 3;
XFConnectCurve $curve1 $tropism1 0;
XFConnectCurve $curve1 $branch1 0;
$primitive1 = `sphere -p 0 0 0 -ax 0 1 0 -ssw 0 -esw 360 -r 1 -d 3 -ut 0 -tol 0.01 -s 8 -nsp 4 -ch 1`;
XFConnectChild $primitive1[0] $branch1 2;
$branch2 = `XFCreateBranch`;
XFConnectChild $branch1 $branch2 0;
```

It is easy to see, that even more complex examples are easy to realize using Xfrog-MEL-commands, which integrate well in the Maya-workflow. Also standard-MEL-commands may be used, but combinations are far more complex although, however everything is possible either. If the standard-MEL-commands are used in the mix with Xfrog-MEL, the structure provided my Xfrog-MEL should be used. Although Xfrog-MEL commands should function with other structures, weird behavior may occur, because the should be structures in some users mind, that can lead to problems while mixing the commands.

## Xfrog – Installation

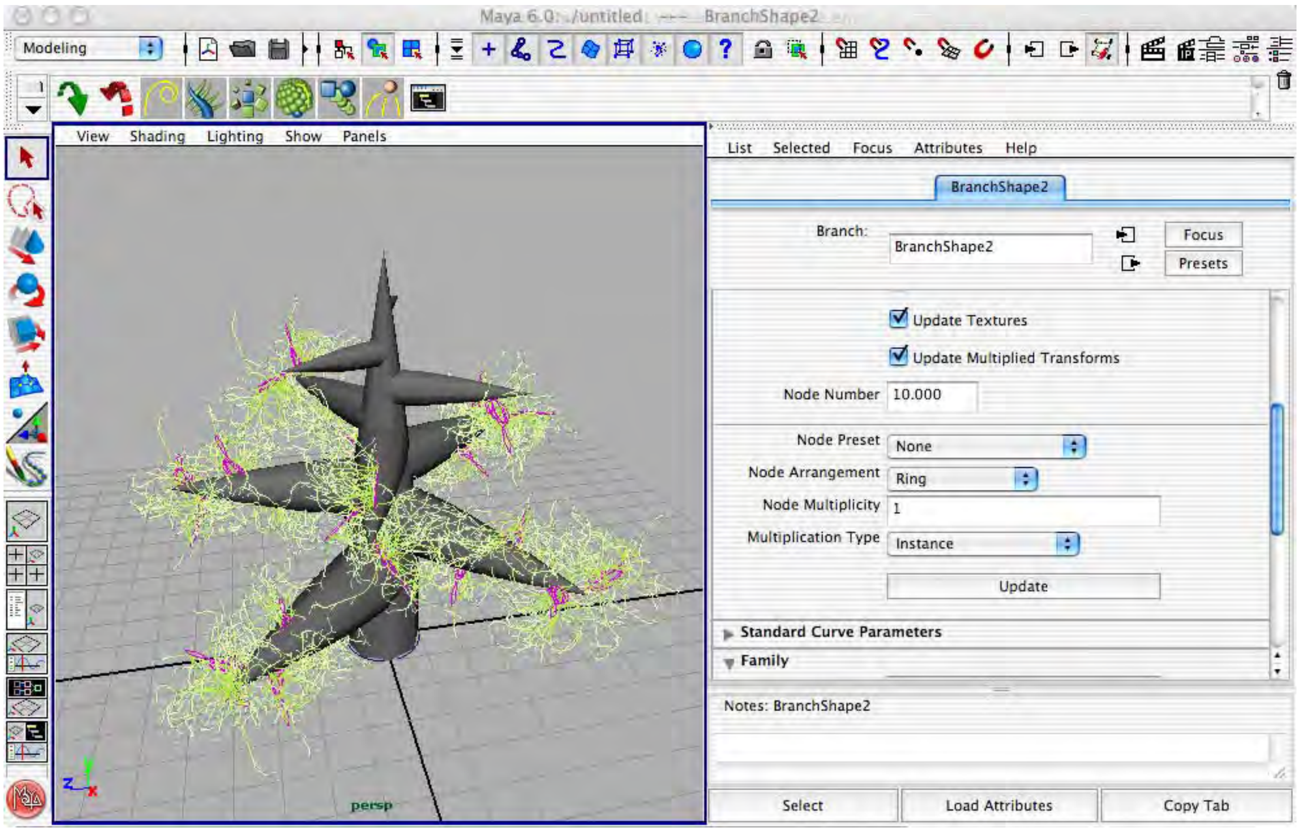
Currently all Xfrog-installation is done directly into the Maya-core directories. It would be much more clean to have them installed into a separate directory, which is possible and very recommended. To do this, follow the described steps:

- Generate a directory at a place accessible by the user using Xfrog4Maya. The path to this directory will be further called install-dir in this paper.
- Extract the Xfrog4Maya-files with all paths into install-dir.
- Locate the Maya.env file holding the environment variables for the user using Xfrog4Maya.
  - o On Windows-systems it is found under:
    *drive:\Documents and Settings\username\My Documents\maya\version*
    *or drive:\Dokumente und Einstellungen\username\Eigene Dateien\maya\version*
  - o On Mac-systems it is found under:
    */Users/username/Library/Preferences/Alias/maya/version*
  - o On Linux-systems it is found under:
    *~/maya/version*
- If not already existing add the following environment variables and append to them so that the following paths can be found:
  - o On Windows-systems:
    *MAYA_SCRIPT_PATH = install-dir\script\/others;*
    *MAYA_PLUG_IN_PATH = install-dir\bin\plug-ins;*
    *XBMLANGPATH = install-dir\/icons;*
  - o On Mac-systems:
    *MAYA_SCRIPT_PATH = install-dir/Maya.app/Contents/scripts/others:*
    *MAYA_PLUG_IN_PATH = install-dir/Maya.app/Contents/MacOS/plug-ins:*
    *XBMLANGPATH = install-dir/Maya.app/Contents/icons:*
  - o On Linux-systems:
    *MAYA_SCRIPT_PATH = install-dir/scripts/others/*
    *MAYA_PLUG_IN_PATH = install-dir/bin/plug-ins/*
    *XBMLANGPATH = install-dir/icons/%B*
- On Windows-systems copy the dll files from install-dir\bin\ to your Windows system32 folder or to the bin folder of your Maya-installation

## Xfrog – Useful hints

Use of paintfx-objects as leaf-objects:

1. Draw simple, short stroke near origin.
2. Make the stroke parent of the stroke-curve.
3. Switch the multiply-type of the branch, which shall get the leafs, to instance.
4. Select the stroke.
5. Add-select the branch.
6. Make the stroke child using + or XFConnectChild 0;.
7. Regard that instancing with too many instances is f…ing slow in Maya.


Maya 6.0: ./untitled --- BranchShape2
Modeling
View Shading Lighting Show Panels
persp
List Selected Focus Attributes Help
BranchShape2
Focus
Presets
Update
Standard Curve Parameters
Family
Select
Load Attributes
Copy Tab

Branch: BranchShape2

☑ Update Textures

☑ Update Multiplied Transforms

Node Number 10.000

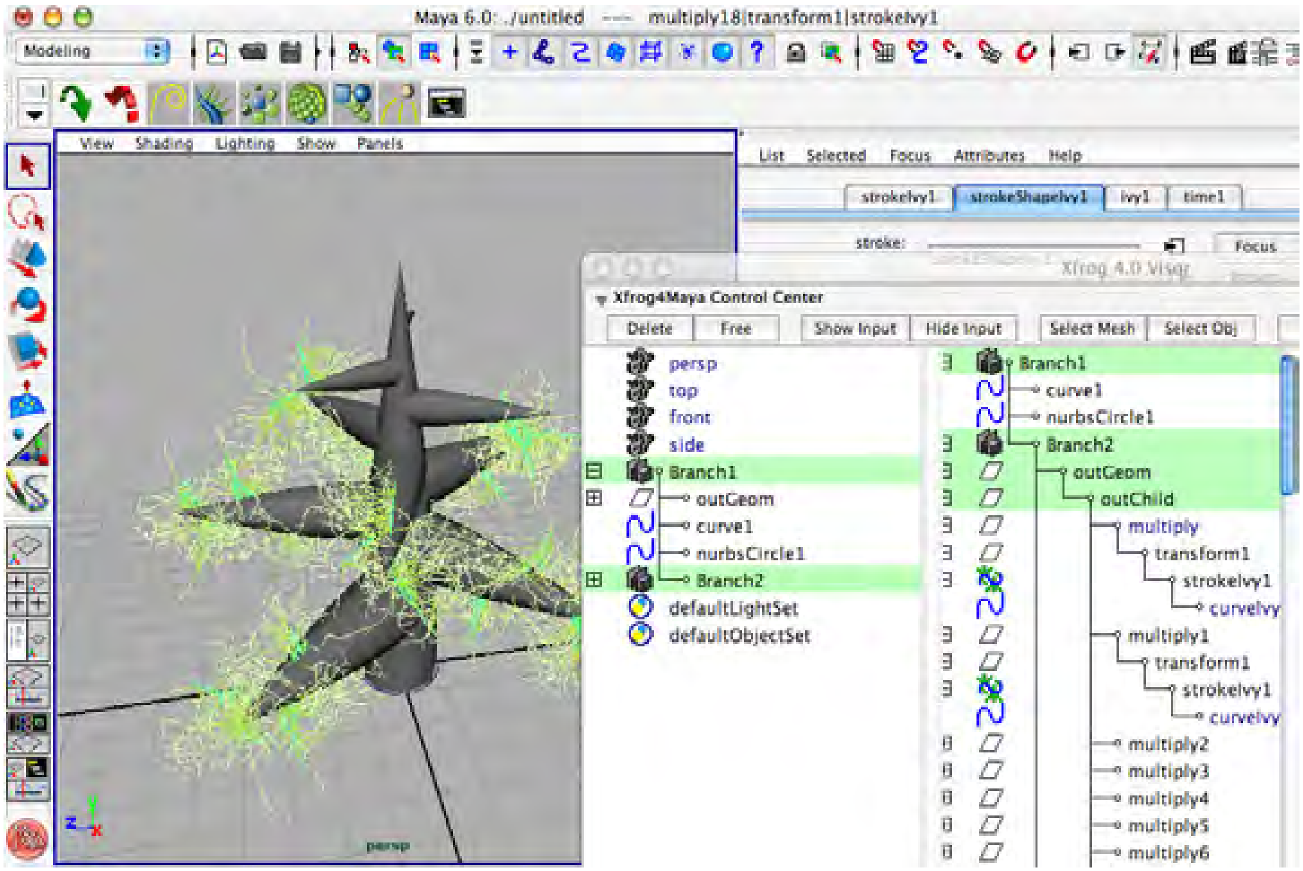
Node Preset None

Node Arrangement Ring

Node Multiplicity 1

Multiplication Type Instance

Notes: BranchShape2


Maya 6.0: ../untitled ---- multiply18|transform1|strokeIvy1
Modeling
View
Shading
Lighting
Show
Panels
persp
List
Selected
Focus
Attributes
Help
strokeIvy1
strokeShapeIvy1
ivy1
time1
stroke:
Focus
Xfrog 4.0 Visor
Xfrog4Maya Control Center
Delete
Free
Show Input
Hide Input
Select Mesh
Select Obj
persp
top
front
side
Branch1
outGeom
curve1
nurbsCircle1
Branch2
defaultLightSet
defaultObjectSet
Branch1
curve1
nurbsCircle1
Branch2
outGeom
outChild
multiply
transform1
strokeIvy1
curveIvy
multiply1
transform1
strokeIvy1
curveIvy
multiply2
multiply3
multiply4
multiply5
multiply6

## Improved accessibility of primitive child-objects:

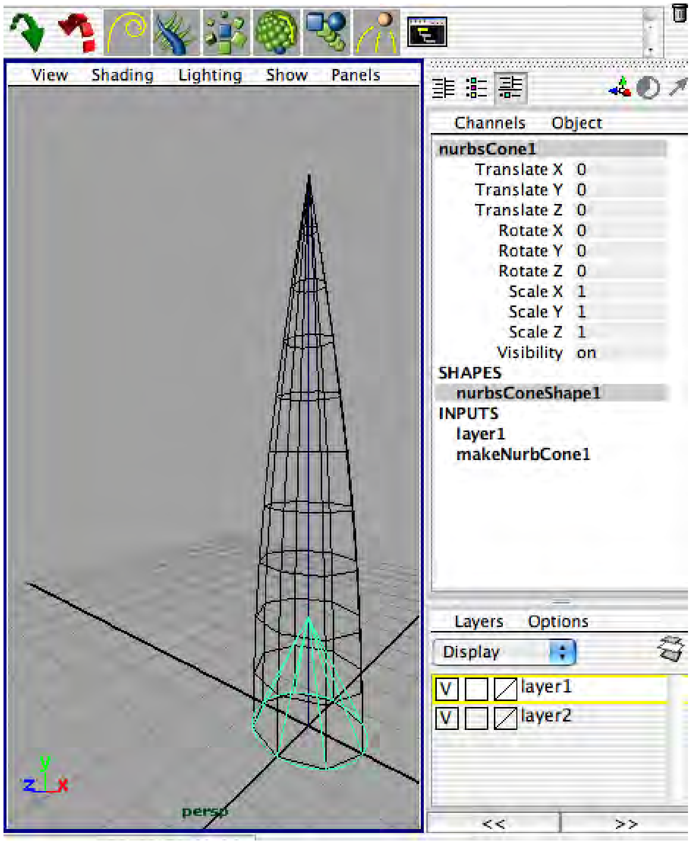
Normally primitive objects assigned as child to an XfrogObject become hidden and work further only as source for geometry. To edit them you need to make them visible again. This can be done in the visor. If you select an XfrogObject, in the right column all primitive geometry-sources appear. By selecting one of them, its attributes appear in the attribute-editor, where visibility can be toggled.

This whole process can be done easier using the XFInputVisibility-MEL-command or the Show Input-button of the visor. Select the object, which primitive child should be visible for editing and then make them visible using the Show Input-button. To make them disappear again after editing use the Hide Input-button.

But this is just the standard way. It is also possible to work with layers that will override the internal visibility-settings. Best you will add the primitive to a layer, before you assign it as a child to an XfrogObject. After the assignment, the primitive will also disappear, but after re-enabling its visibility using the Show Input-button, visibility is steered by the layer. This way you can control the visibility now by the layer-settings.

The same is valid for the created geometry of XfrogObjects. Often they disturb while editing the primitives. You can easily select the top-level XfrogObject, afterwards use the Select Mesh-button or the XFSelect-MEL-command to select just the created geometry. After that you can add all selected objects to a new layer. This layer steers the visibility of the XfrogObjects created geometry, while the first layer controls the input-primitives-geometry. This way it is possible to visualize just the components you need for editing, to have a smoother access to the input.

If you have your XfrogObject with already assigned primitives, it is also possible to make the input-objects visible using the described way and then to add them to a separate layer later.

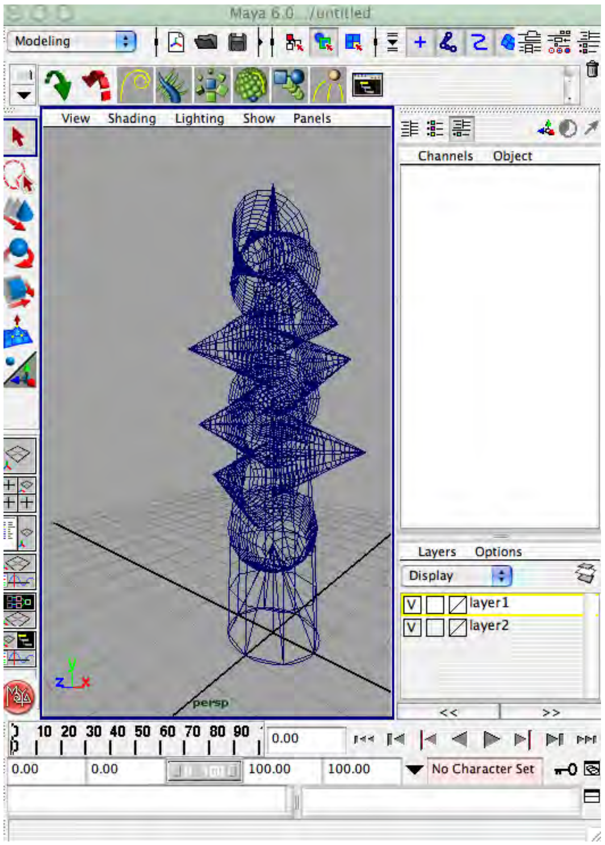


nurbsCone1

| | |
|---|---|
| Translate X | 0 |
| Translate Y | 0 |
| Translate Z | 0 |
| Rotate X | 0 |
| Rotate Y | 0 |
| Rotate Z | 0 |
| Scale X | 1 |
| Scale Y | 1 |
| Scale Z | 1 |
| Visibility | on |

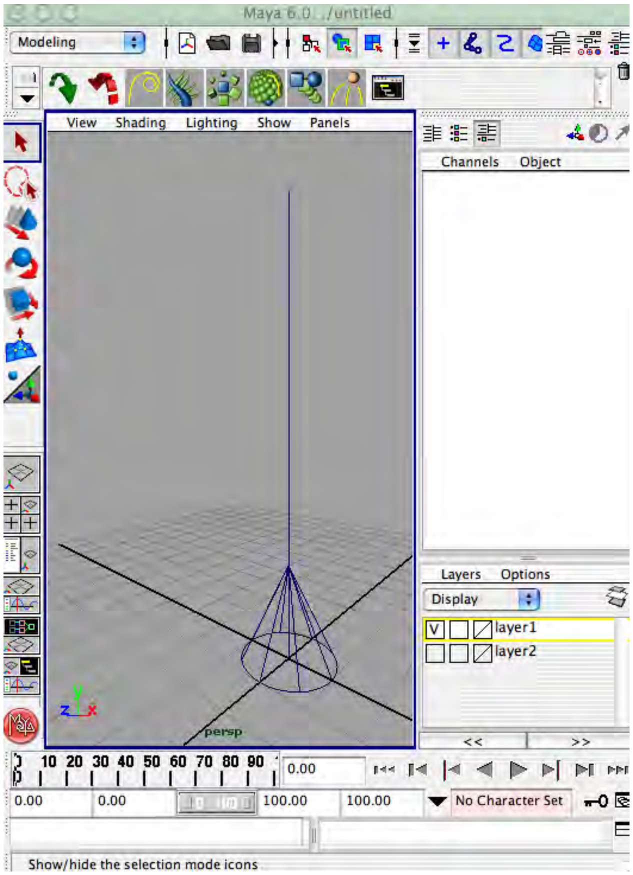
SHAPES
nurbsConeShape1
INPUTS
layer1
makeNurbCone1

V layer1
V layer2

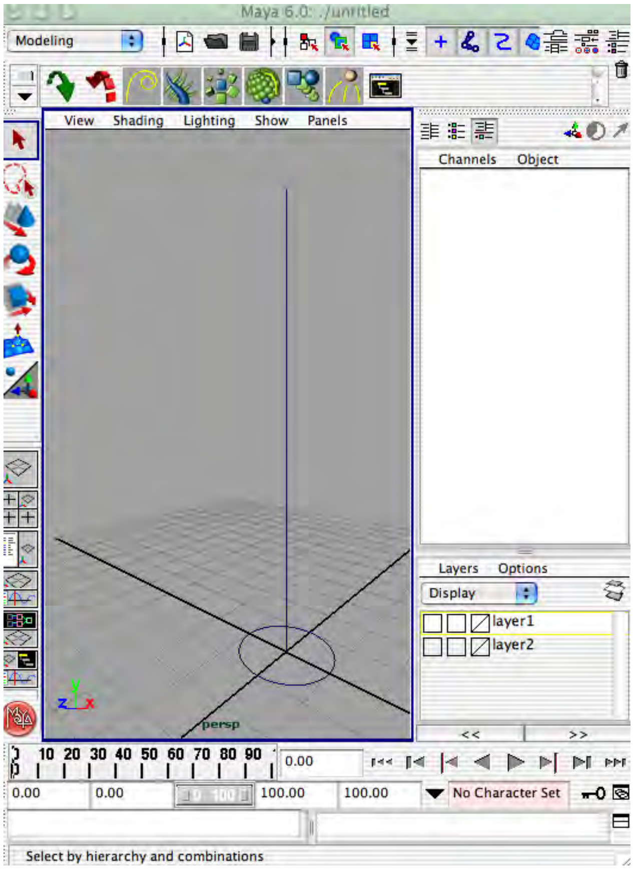


V layer1
V layer2

0.00
0.00 0.00 100.00 100.00
No Character Set



Show/hide the selection mode icons



Select by hierarchy and combinations

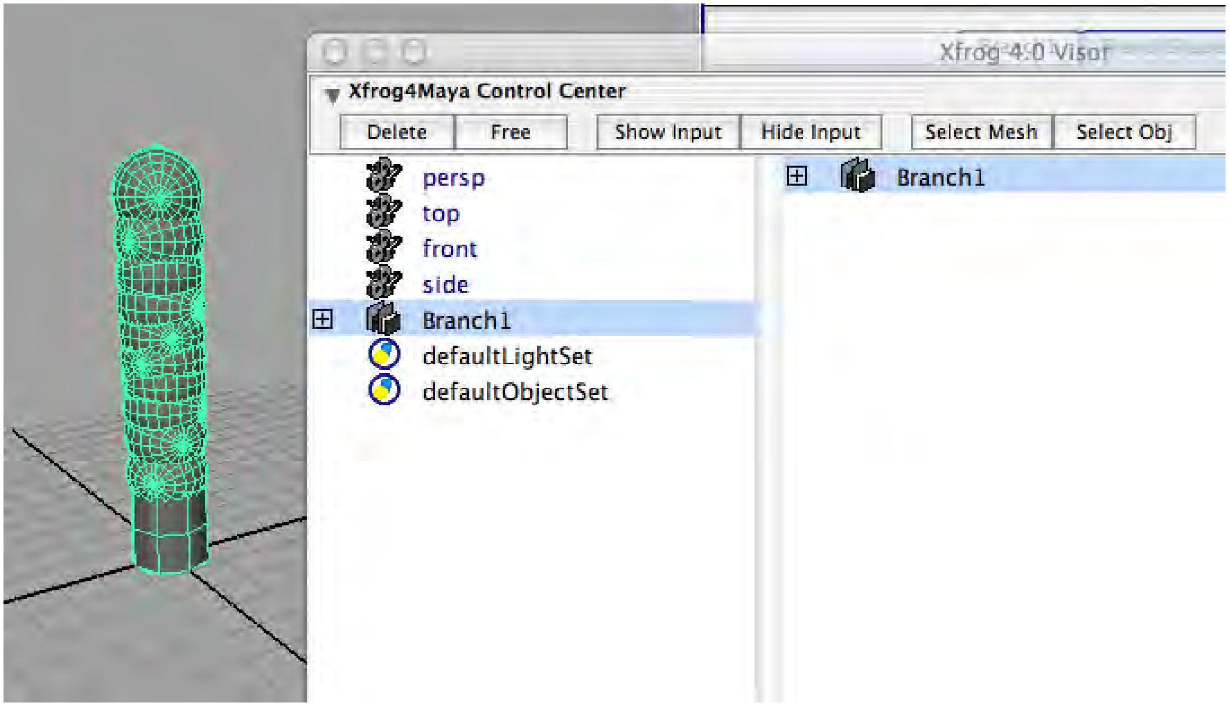
## Improved performance using multiple top-level XfrogObjects:

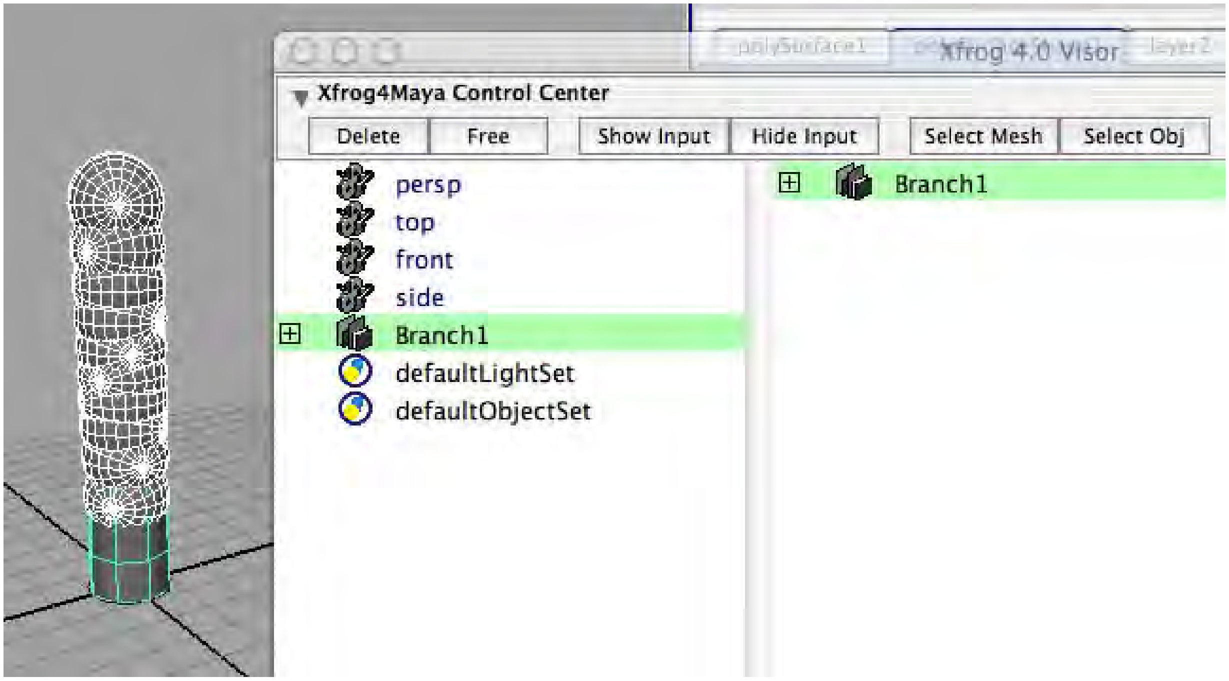
If you have to handle multiple top-level XfrogObjects, it is often a performance issue, to always recalculate all the instances. For the rendering process this must be and is tolerable, but for editing this may disturb. The easiest way to bypass this is to make XfrogObjects currently being not edited not to evaluate. This can be done using the Evaluation-checkBox of each top-level XfrogObject. The whole object will not be calculated then, but will also disappear from the scene. This way you can make only the objects evaluate needed for current editing having an improved performance.

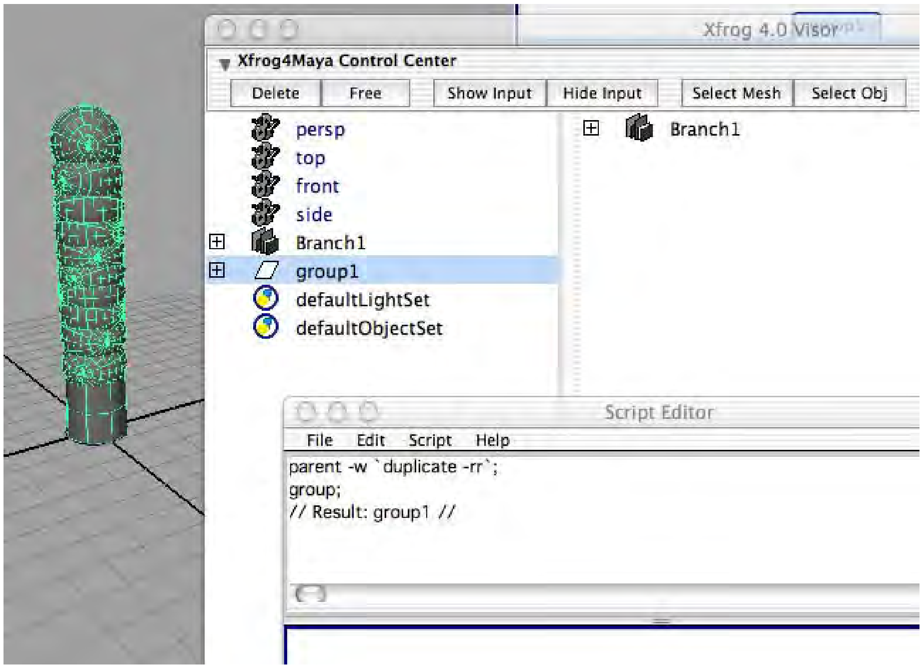
Before rendering easily switch all objects to evaluation and all objects of the scene will be rendered.
Often objects in the scene are static, what means they are neither animated nor change their geometry. These objects do not need to be calculated as XfrogObjects, it is enough to have them as simple mesh. If you are sure, that the objects are no longer needed to be editable, you can use the Select Obj-button of the visor or the XFSelect-MEL-command to just select the Xfrog-generator-nodes of the previously selected top-level XfrogObject. Afterwards you can delete these nodes using the delete-MEL-command or the Delete-visor-button. After this the remaining object is just the geometry-representation of the former XfrogObject. If you are not sure, the generator-nodes can be deleted, you can also select the top-level object and then select its output-mesh using the XFSelect-MEL-command or the Select Mesh-visor-button.
The selected output-geometry may be duplicated then to create a simple mesh-representation-copy of the XfrogObject. The MEL-statement to create a duplicate as a separate group is:

```
parent -w `duplicate -rr`;
group;
```

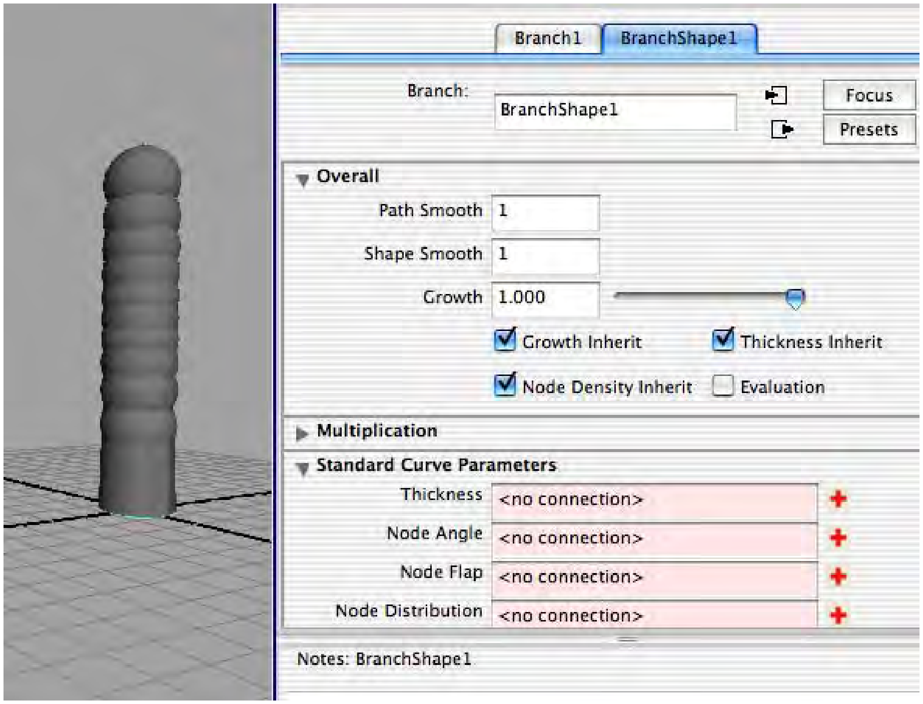
After executing this statement, you can make the original XfrogObject not evaluate. It will remain as potential source of changed geometry in future, but for faster evaluation the mesh-copy does the job well. This may be also used to speed up editing of other XfrogObjects. If you edit an XfrogObject, which may be near to another you do not edit, but you would like to see, to better adapt the objects view to the other one, you can work with the other objects mesh-representation instead of its original for faster interaction at edit-time. For rendering then the other objects mesh-representation can be hidden and the original may be used.


Xfrog 4.0 Visor
Xfrog4Maya Control Center
Delete
Free
Show Input
Hide Input
Select Mesh
Select Obj
persp
top
front
side
Branch1
defaultLightSet
defaultObjectSet
Branch1


polySurface1
Xfrog 4.0 Visor
layer2
Xfrog4Maya Control Center
Delete
Free
Show Input
Hide Input
Select Mesh
Select Obj
persp
top
front
side
Branch1
defaultLightSet
defaultObjectSet
Branch1



```
parent -w `duplicate -rr`;
group;
// Result: group1 //
```



Branch: BranchShape1

Path Smooth 1

Shape Smooth 1

Growth 1.000

✓ Growth Inherit ✓ Thickness Inherit

✓ Node Density Inherit ☐ Evaluation

Thickness <no connection>

Node Angle <no connection>

Node Flap <no connection>

Node Distribution <no connection>

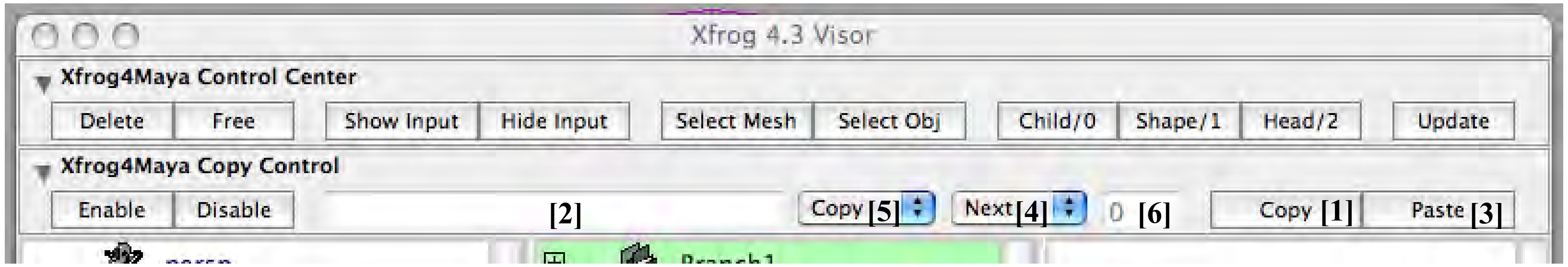
Notes: BranchShape1

# Xfrog 5 – Improvements, Remarks, Copy and Paste

## 0) Stability

- Effects of VisorPanel loosing contact should be gone
- Stability should be improved after some causes for potential or already occurved crashes are removed
- File saving and loading should now work fine for *.ma and *.mb

## I) Copy and Paste



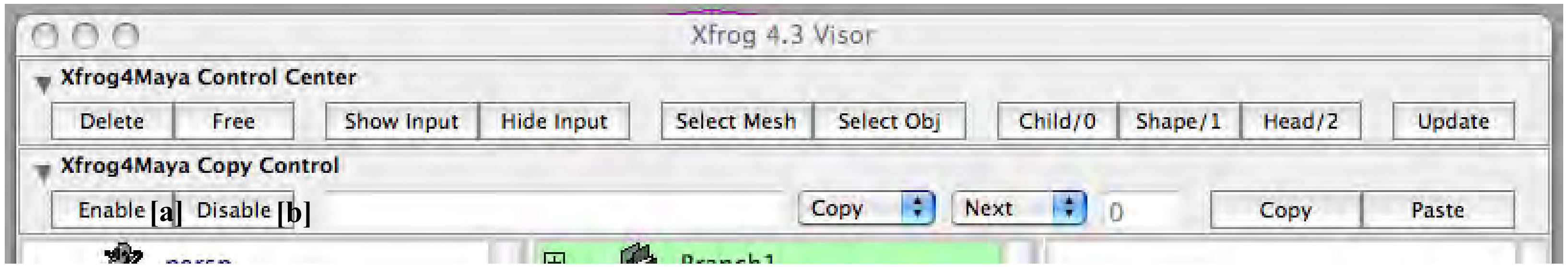
Use the Xfrog Copy Control according to the following workflow:

| | |
|---|---|
| **1.** select object of source type | |
| **2.** klick on copy-button **[1]** | (copied object appears in the clipboard-textfield **[2]**) |
| **3.** choose kind of copy **[4]** | |
| - Move | (object is removed from source position and placed under destination position) |
| - Copy | (object remains in former position and a copy is placed under destination position) |
| **4.** choose copy type **[5]** | (chosen type must be one of the possible types for source and destination) |
| - Child/Path | (object is always placed as path or child with offset entered in the textfield) |
| - Shape | (object is always placed as shape or second child) |
| - Head | (object is always placed as head or third child) |
| - Next | (object is placed at the lowest free offset (1. Child 0/path, 2. Shape/Child 1, 3. Head/Child 2, 4. Child 3, ...) |
| **5.** enter child offset into textfield **[6]** | (only needed for chosen copy type Child/Path, higher numbers than 0 only needed for Variation) |
| **6.** select object of matching destination type or deselect all for world as destination | |
| **7.** klick on paste-button **[3]** | |
| **8.** selected source is moved/copied below selected destination as chosen copy-type | |

Overview of source-types with matching destination-types and possible copy-types for the combination:

| *source type* <br> *-> matching destination type* | *(possible copy types)* |
|---|---|
| Tropism | |
| -> Nurbs/CurveNurbs | |
| -> Tropism | |
| -> world | |
| Nurbs/CurveNurbs | |
| -> Branch | (Path, Shape, Next (1. Path, 2. Shape)) |
| -> Phyllotaxis | (Path, Next (1. Path)) |
| -> world/transform | |
| Branch/Hydra/Phyllotaxis/Variation | |
| -> Branch | (Child, Shape, Head, Next (1. Child, 2. Shape, 3. Head)) |
| -> Hydra/Phyllotaxis | (Child, Next (1. Child)) |
| -> Variation | (Child (Child with offset entered in the textfield), Shape (Child 1), Head (Child 2), Next (1. Child 0, 2. Child 1, 3. Child 2, ...)) |
| -> world/transform | |
| mesh/nurbs/subdiv | |
| -> Branch | (Child, Shape, Head, Next (1. Child, 2. Shape, 3. Head)) |
| -> Hydra/Phyllotaxis | (Child, Next (1. Child)) |
| -> Variation | (Child (Child with offset entered in the textfield), Shape (Child 1), Head (Child 2), Next (1. Child 0, 2. Child 1, 3. Child 2, ...)) |
| -> world/transform | |

## II) Automatic connection of Xfrog-objects by parenting



Use Enable-button **[a]** to activate automatic connection. Use Disable-button **[b]** to deactivate automatic connection.

Having automatic connection activated:

- objects childed unter Xfrog-Objects are automatically connected
- tropisms childed under curves are automatically connected
- parenting-control works for kinds of reparenting
  - for mel-commands
  - for drag-and-drop in Visor/VisorPanel
- parenting-control works similar to Copy and paste with the following settings (compare **I)**)

| | |
|---|---|
| **1.** drag-source | = selected object of source type |
| **3.** kind of copy | = Move |
| **4.** copy type | = Next |
| **6.** drag-destination | = selected object of matching destination type |

Having automatic connection deactivated:

- everything works as known before Xfrog Copy and Paste

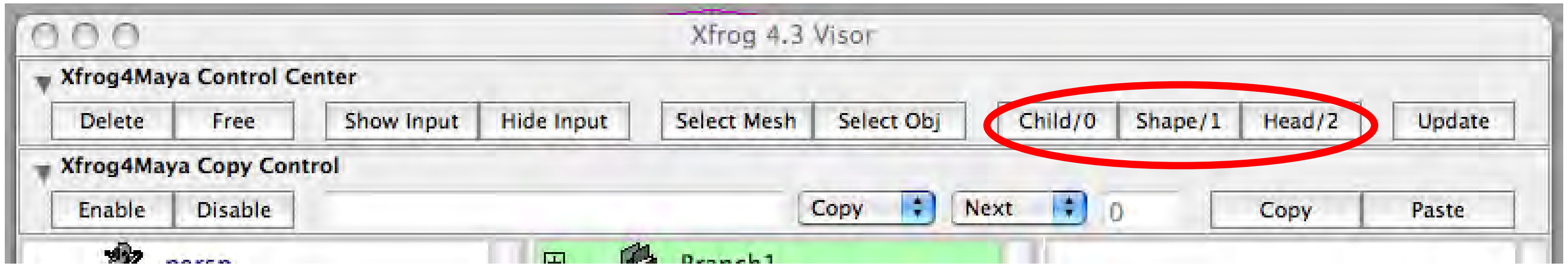
Pitfalls:

- making automatic connections clears undo-history for each made connection
- so be sure not to loose important undo-steps
- this behaviour is due to the maya parenting-callbacks and cannot be circumvented

Workaround for not losing undo-history:

- deactivate automatic connections
- use copy and paste for reparenting using copy-kind "Move" as described under **I)** with the given settings for **1.**, **3.**, **4.** and **6.** above
- Copy and Paste is undo-proof

## III) Wallis misunderstanding of Child/Shape/Head-buttons in Visor/VisorPanel



Behaviour for setting children is not inconsistent. In Xfrog for Maya p.e. a Branch can have more than one child and also more than one shape, etc.

So the possiblity to add children/shapes/heads using Visor/VisorPanel is the following:

- add children using the Child-button
- to add 2 children to the branch usethe Child-button again, both children are used as Branch's (first) child
- this means that you need to use the Shape-button to add an object as Branch's shape
- the same is true for children of the variation - a variation can have multiple first children, added using the Child-button
- second children for the variation are added using the Shape-button, third children using the Head-button
- 4th and higher children cannot be added using the Visor/VisorPanel
- so visor-behaviour is consistent
- to weaken the potential of misunderstanding I added /0, /1 and /2 to the button-texts for the matching child-offsets
- if the /0, /1 and /2 extension do not please your taste, I can remove them afterwards

The other possibility to add children is given in the Xfrog-objects parameters:

- you have plus-buttons for all possible types of child/shape/head
- using the plus buttons more than once for each type enables you to have p.e. more than one (first) children of a Branch as described above
- for Branch you have buttons for child/shape/head here, for Variation you have a buttons for each existing child-offset
- so consistent here too

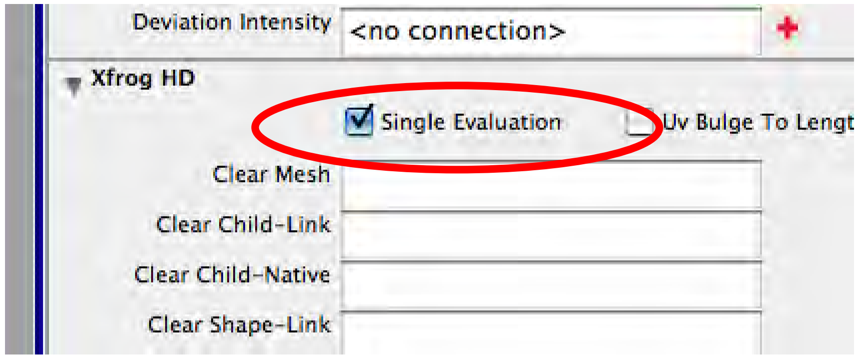
You have this way to separate possibilities for adding children. Each of them is consistent, choose the one preferable for your current working-step. Understanding may be improved by the /0, /1 and /2 extensions - have a look.

## <u>IV) Selective evaluation of Xfrog-object-parts as requested by Walli</u>

Up to now you could only set the evaluation-check for the top-level-object.

From now on at the top of the XfrogHD-area in the object-parameters there is an additional single-evaluation-check, which can be set on each object in the hierarchy seperately.

This way you can deactivate parts of your hierarchies, like nuts/flowers/etc.


Deviation Intensity <no connection>

Xfrog HD

[✓ Single Evaluation] Uv Bulge To Lengt

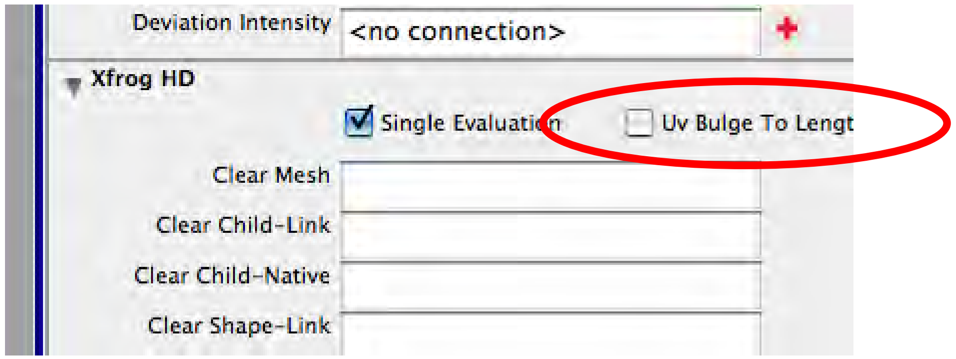
Clear Mesh

Clear Child-Link

Clear Child-Native

Clear Shape-Link

## V) Switching UV-placement of Branch from 0-maximum to 0-currentGrowth



Deviation Intensity <no connection>

Xfrog HD

[✓] Single Evaluation  [ ] Uv Bulge To Lengt

Clear Mesh

Clear Child-Link

Clear Child-Native

Clear Shape-Link

You can switch the UV-placement for the Branch using the uvBulgeToLength-check.

Having uvBulgeToLength checked, the UV-coordinate starts at 0 and ends at 1 for the current growth.

## VI) Editing of CurveParameters

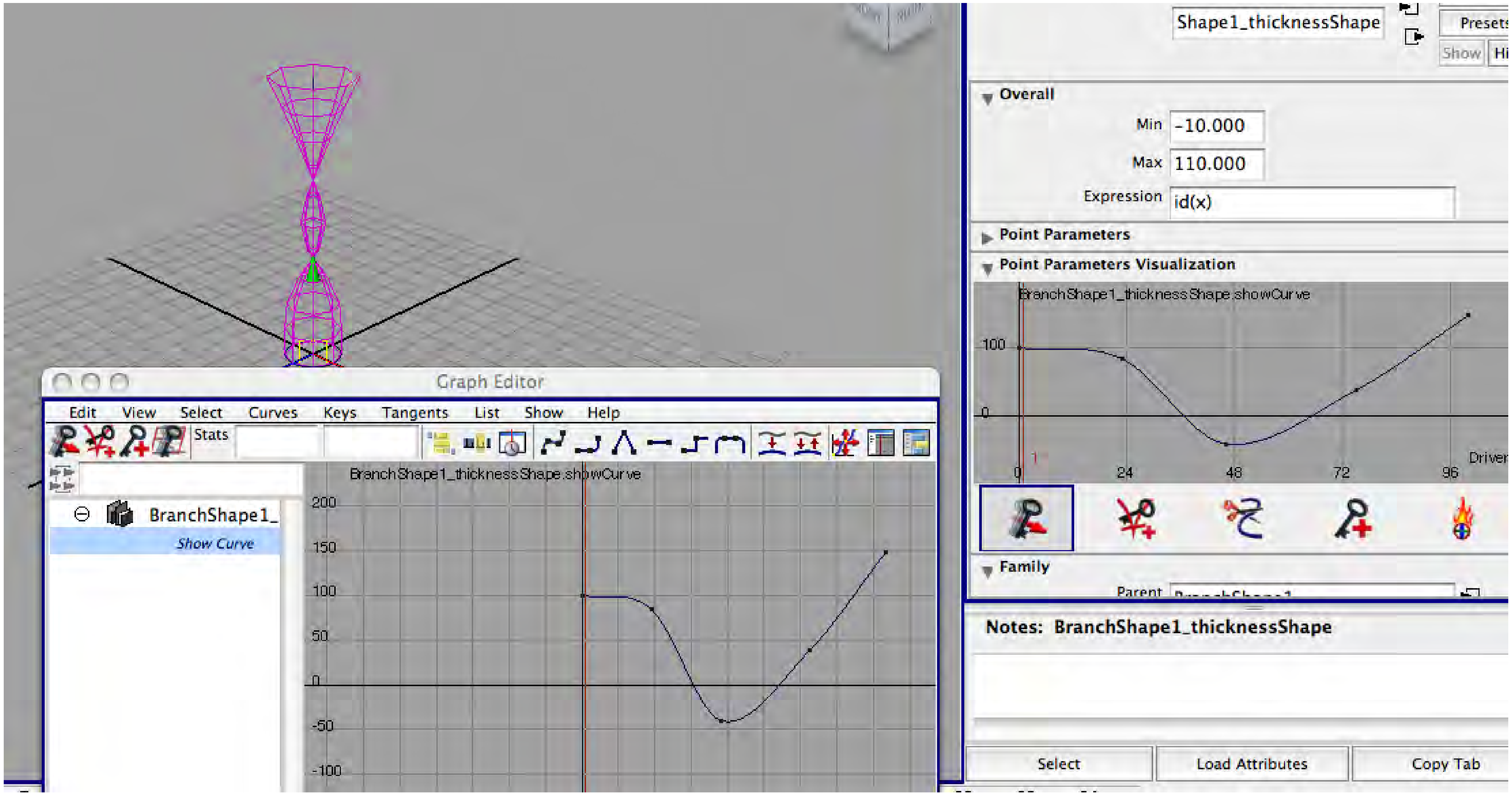
Orio had the wish for the editing of curve-parameters similar to the cinema-implementation.
**This is not possible with Maya and will probably never be!!!**
The Maya interface is very strict and provides no possiblities for completely custom graphic-components.

But if you are familiar with editing maya-anim-curves, the usage of the curve-editing capabilities is straight forward and not much a disadvantage against cinema-implementation:

- moving points: maya 2 clicks (left to select, middle to move) - cinema 1 click
- adding points: maya 1 clicks (middle to add) - cinema 1 double-click
- removing points: maya 2 clicks (left to select, button to remove) - cinema 1 click

Walli mentioned that curve-parameters are way to small inside the parameters.
Maya has the Graph-Editor (Menu Window->Animation Editors->Graph Editor) for that:

- select the object you want to edit curve-parameters for
- open the Graph-Editor or keep it in a panel
- you have listed curve-paramters of the selected object in the object-list of Graph-Editor
- select the curve-parameter (or better the anim-curve "showCurve" below the curve-parameter) you want to edit
- edit the curve-parameter in the size you want
- the buttons/values in the curve-parameters object-parameters can be used in parallel



Shape1_thicknessShape

Overall

Min -10.000

Max 110.000

Expression id(x)

Notes: BranchShape1_thicknessShape