

Constraints - Dynamic binding of ligands

We will work with the EphA2 receptor binding of an ephrin A5 ligand.

Open the molecular maya tab. In the Import tab download section, write 2X11 and press download then import.

Go to the main tab of mMaya and scroll down to meshing, expand meshing and check the mesh vis checkbox.

Modify the mesh until you are happy with the look, for the look in the first fig. These parameters were used:

Save the file as EphBinding.000.mb

In the outliner, expand the pdb_2X11 group and find the polygon objects representing the protein meshes. Click ctrl+D to duplicate the mesh geometry. MMB drag the new copies out of the pdb_2X11.

Delete the original mMaya object by selecting the pdb_2X11 group in the outliner and pressing delete. (we do this to avoid mMaya related crashes as we continue to work) Rename the duplicated meshes. The long chain should be called EphA2, the short protein: ephrinA5.

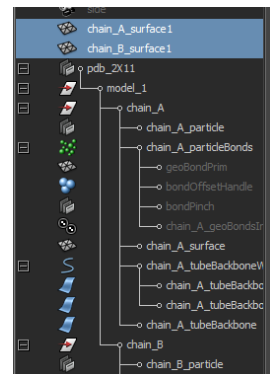
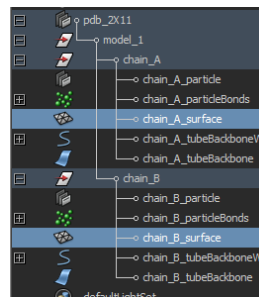
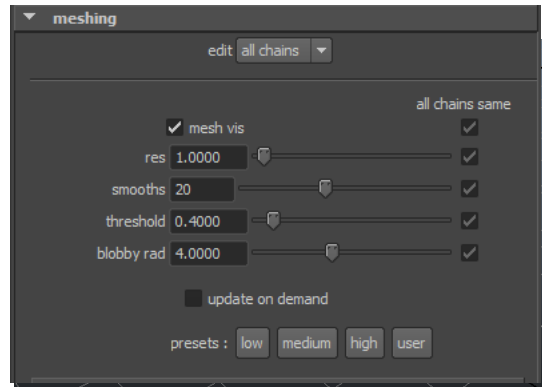
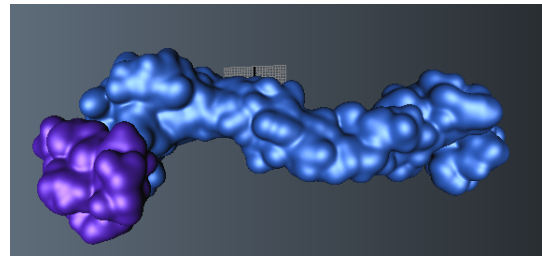
Group ctrl+G the two meshes, scale, translate and rotate the group down to fit nicely in the middle of the Maya grid with the ephrin on top of the Eph pointing up. In the outliner, MMB drag the EphA2 and ephrinA5 out of the group you just created, delete the group.

Save the file as EphBinding.001.mb

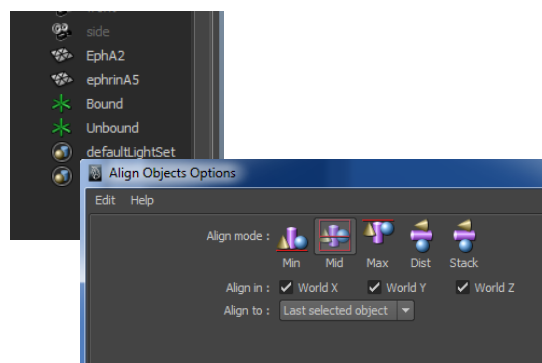
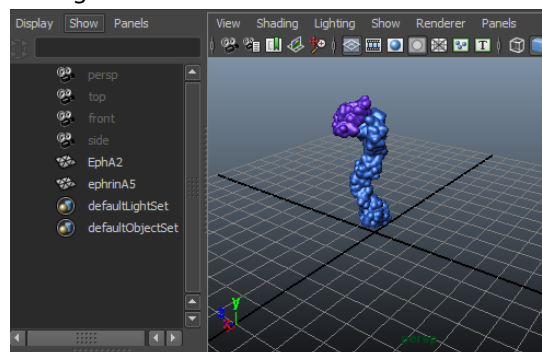
Select the ephrinA5 and do Modify->Center Pivot

Do Create->Locator, twice. In the outliner, rename the locators "Bound" and "Unbound" respectively.

Select the Bound locator first, then the ephrinA5, do Modify->Snap Align Objects -> Align Objects and choose the options box. Align to last selected objects and all coordinates have to get the Bound locator to align with the ephrinA5.

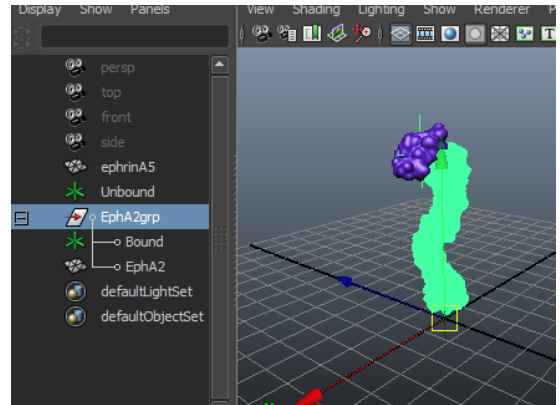


What it should look like after re-grouping and scaling:

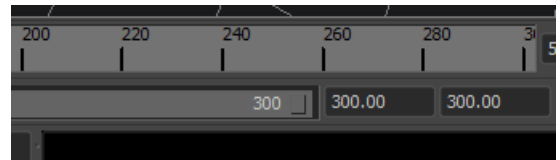


Group the Bound locator and the EphA2, name the new group "EphA2grp"

Select the EphA2grp and move its pivot down to the bottom tip of the protein. (to move the pivot switch to the move tool, press D and keep it pressed while moving the pivot)



To the right of the time slider, write 300 in the end-of-animation box. Expand the time slider to 300.



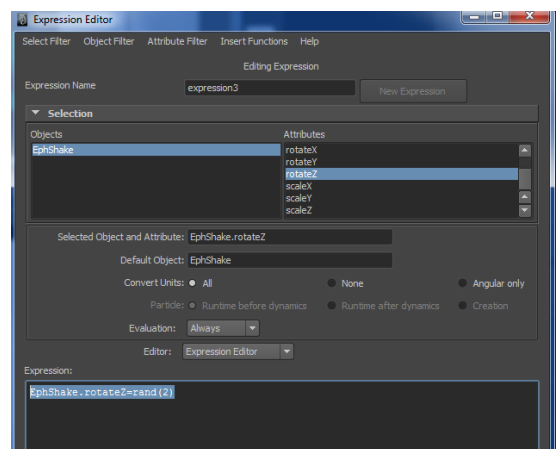
With the EphA2grp selected, set a 3-4 keys on the rotations in X, Y and Z to animate the Eph-receptor movement as it protrudes from a cell membrane.

Save as EphBinding.003.mb

Group the EphA2grp to itself. Rename the new parent group: EphShake. With the EphShake group selected, do Window -> Animation Editors -> Expression editor.

In the expression editor create these expressions:

EphShake.rotateX=rand(2)
 EphShake.rotateY=rand(2)
 EphShake.rotateZ=rand(2)



Create a cubic CV-curve in 3D space around the protein.

Group the Unbound locator to itself. Rename the group: UnboundTranslate

Select the UnboundTranslate group, shift-select the CV curve. In the Animations menu-set, do Animate->Motion Paths->Attach to Motion Path. Press play.

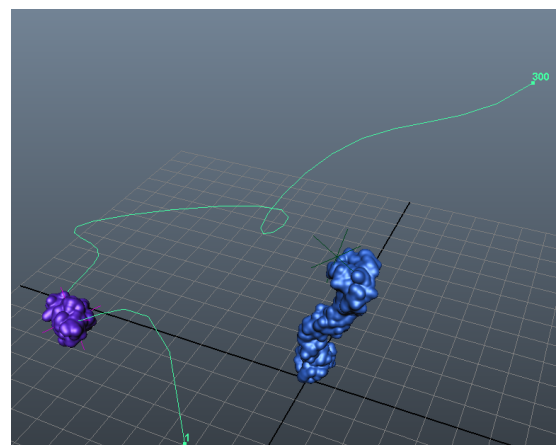
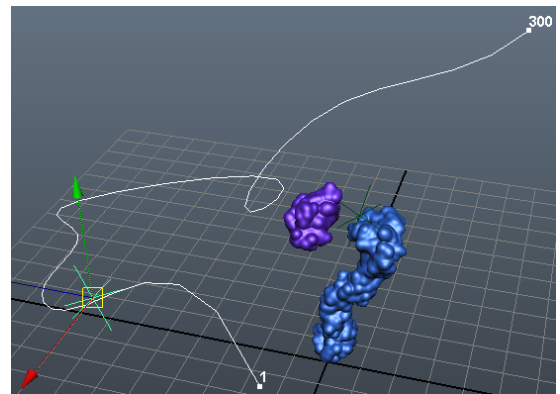
In the expression editor create these expressions:

Unbound.translateX=rand(0.2)
 Unbound.rotateX=rand(10)
 and the same for Y and Z

Now, lets bind the ephrinA5 to the locator we just animated.

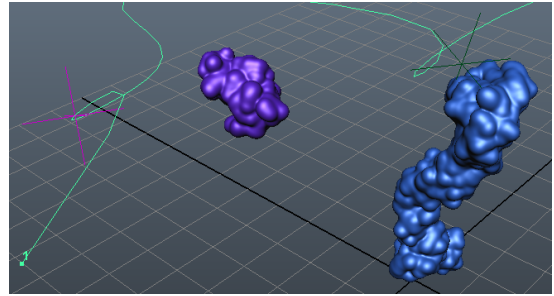
Firs select the Unbound locator, then shift-select the ephrin geometry, then (Animation menu-set) Constrain -> Point. The ephrin should snap into place, press play.

Then again, select the Unbound locator, then shift-select the ephrin geometry and do Constrain-> Orient.

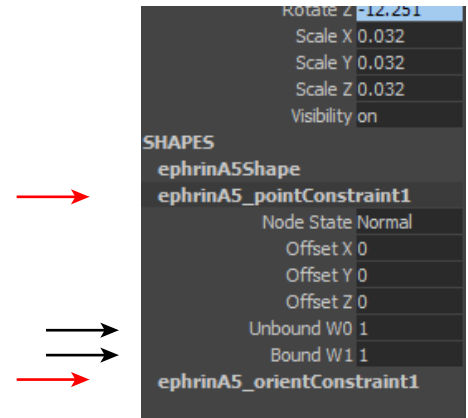


Now we will create the same type of constraints but this time, using the Bound locator as our target.

Repeat the Point- and Orient- constraints for the ephrin but using the Bound locator as the target this time (select that before the ephrin geometry). The ephrin should now end up somewhere in the middle between the Bound and Unbound locators.



Select the ephrinA5 geometry, and go to the channel box. Expand the ephrinA5_pointConstraint, and ephrinA5_orientConstraint (red arrows), notice that we now have some new weights to play with (black arrows) by changing these weights, the ephrin can now appear attached to the Bound or Unbound locator respectively.




Animate (set keys) on the weights of the Point and Orient constraints to animate binding of the ephrinA5 ligand to its EphA2 receptor.

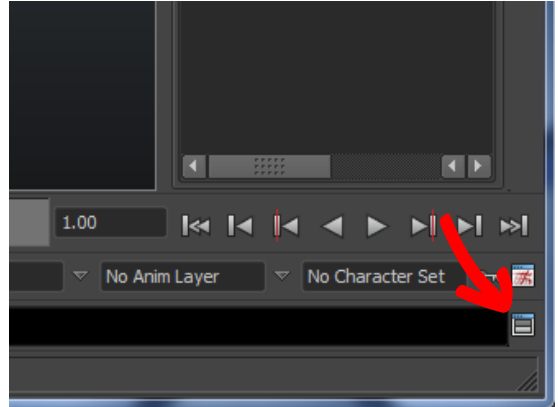
You can also easily go in and edit the CVs of the motion path so that the ephrinA5 appears to pass by close to the receptor.

Simple scripting in Maya with Python

In Maya, you have two options for scripting. Either you use MEL (Maya Embedded Language) which is Maya's native language. Or, you use Python to access the MEL commands.

Here we will look at Python scripting. Since Python is used extensively in many areas in science, getting to know a little bit about Python is not going to hurt.

Bring up the "Script Editor" either by clicking this icon:  or by doing Window->General Editors->Script Editor



To get the most "Pythonesque" experience when scripting in Maya we will use a Python module called "pymel". Pymel will allow us to work with Maya commands in an intuitive, object-oriented way.

The help files for PyMEL are here:

http://download.autodesk.com/global/docs/maya2013/en_us/PyMel/index.html#



Here is the Python documentation (highly recommended):

<http://docs.python.org/>

Our first line of code will be:

```
MEL Python
1 import pymel.core as pm
2
```

This line will import the pymel package

To run commands in the script editor. Press the  button to run all the code. Or select the lines you want and press the  button to execute only selected lines:

Try this and execute:

```
MEL Python
1 import pymel.core as pm
2
3 pm.polySphere()
```

The pm prefix tells the Python interpreter to use the pymel.core module for the polySphere() command (as we have defined as pm in the first line)

Go ahead and try (prefixed with pm.):

sphere()
polyCube()
plane()
polyTorus()

and see what they do.

When we run pm.polySphere() we see that we get the following output:

```
# Result: [nt.Transform(u'pSphere2'), nt.PolySphere(u'polySphere2')] #
```

A Python list with:

 Transform node

 Construction history

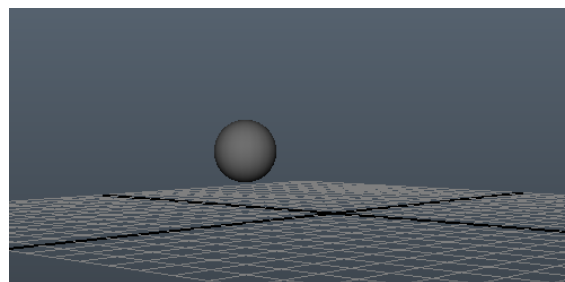
Lists in Python are arrays where each element can be accessed by the bracket operator []. To access the first element of a list called "theList" we would write "theList[0]"

In Maya, to change the translation, scaling, rotation and visibility, we access the *Transform Node* of that object.

Try the following code:

```
MEL Python
1 import pymel.core as pm
2
3 sp=pm.polySphere()
4
5 spTrans=sp[0]
6 spTrans.translateX.set(2)
7 spTrans.translateY.set(2)
8 spTrans.translateZ.set(2)
```

Here, we are saving the list in a temporary variable sp, then we get the transform node out and store it in the spTrans variable. Then we set all the translations to 2.



In Python you can create a list like so:

```
myList=[item1, item2, item3]
```

where each item could be a number, text, another list, etc.

Lets say we want to input spheres at these locations:

radius	x	y	z
0.5	1.0	-1.0	4.5
1.0	1.0	-2.3	5.3
1.5	2.0	0.5	8.3
2.0	2.5	3.0	9.0
2.5	3.0	5.0	2.0
3.0	3.2	4.3	5.0

To do that we first create a Python list called "coords":

```
coords=[
    [0.5 , 1.0 , -1.0 , 4.5],
    [1.0 , 1.0 , -2.3 , 5.3],
    [1.5 , 2.0 , 0.5 , 8.3],
    [2.0 , 2.5 , 3.0 , 9.0],
    [2.5 , 3.0 , 5.0 , 2.0],
    [3.0 , 3.2 , 4.3 , 5.0]
]
```

Now, try this code (you should be able to copy-paste the coord variable from above):

```
MEL Python
1 import pymel.core as pm
2
3 coords=[
4     [0.5 , 1.0 , -1.0 , 4.5],
5     [1.0 , 1.0 , -2.3 , 5.3],
6     [1.5 , 2.0 , 0.5 , 8.3],
7     [2.0 , 2.5 , 3.0 , 9.0],
8     [2.5 , 3.0 , 5.0 , 2.0],
9     [3.0 , 3.2 , 4.3 , 5.0]
10 ]
11 for entry in coords:
12     sp=pm.polySphere(r=entry[0])
13     spTrans=sp[0]
14     spTrans.translateX.set(entry[1])
15     spTrans.translateY.set(entry[2])
16     spTrans.translateZ.set(entry[3])
17
```

This is what the scene should look like when you are done executing the code:

Lets go through the code.

On line 3-10 we define the list variable "coord".

On line 11 we start a so called *for-loop*. For-loops in Python often go through a list. Here we go through the coords list and for each item in the list we extract the radius (entry[0]), the X-coordinates (entry[1]) and so forth for the Y and Z coordinates.

```
pm.polySphere(r=entry[0])
```

thus gives a polygon sphere of a radius that is determined by the first item in the `entry` variable, where the `entry` variable is, in turn, determined by current iteration of the for-loop.

A CV curve from data

Similar code can be used to create a CV curve. This time the list must be supplied as a collection of coordinates of the format (x,y,z) create this variable:

```
coords2=[
    ( 1.0 , -1.0 , 4.5),
    ( 1.0 , -2.3 , 5.3),
    ( 2.0 , 0.5 , 8.3),
    ( 2.5 , 3.0 , 9.0),
    ( 3.0 , 5.0 , 2.0),
    ( 3.2 , 4.3 , 5.0)
]
```

Now try:

```
pm.curve(p=coords2)
```

by default the CV curve will be cubic, if you want to use data, you probably would like the curve to pass through each point, thus using a linear CV curve. This can be obtained with:

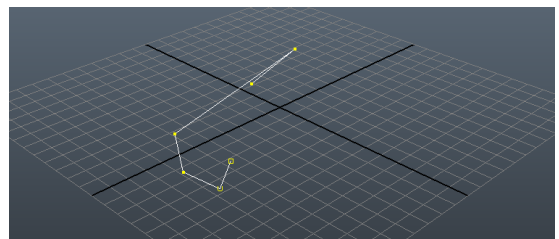
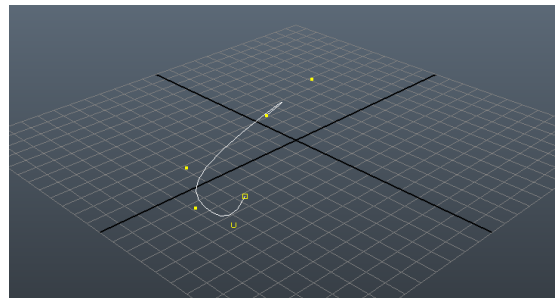
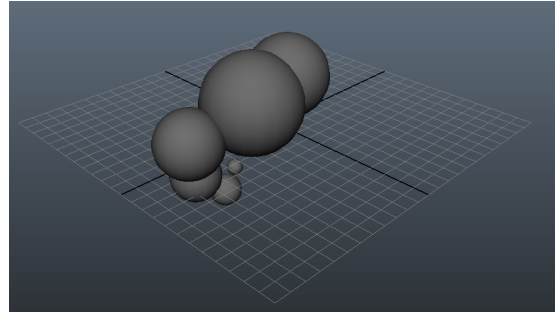
```
pm.curve(p=coords2, degree=1)
```

or

```
pm.curve(p=coords2, d=1)
```

To work with some general data, download the sample file `SomeData.csv` and note down the path to that. Using this code you should be able to load the data into maya (note that the path must have double forwards slashes):

```
import csv
pts=[] #initiate an empty list where we will store the coordinates
with open('C://..PATH...TO.../SomeData.csv', 'rb') as f:
    reader = csv.reader(f)
    reader.next() #skip the first row
    for row in reader:
        x=float(row[0]) #read and convert to number (float)
        y=float(row[1])
        z=0 #2D data, lets put z=0
        pts.append((x,y,z))
```



After you have made a curve from the .csv data, lets animate a growing graph.

If you didnt do so already, make the CV curve with degree set to 3 to get a somewhat smoother curve.

Make a NURBS circle and do an extrude along the path. *Note*, make sure you check Partial in Curve Range.

If you notice that the extruded surface looks really bad and jagged we can use the following trick to increase resolution of the curve. (*Note!* This will remove the old data points, the rebuilt curve will not have one CV per data point.)

Select the CV curve that you made from data. Surfaces menu-set, do Edit Curves -> Rebuild curve, options box.

Select Reduce in Rebuild type. Press Rebuild.

This will re-distribute the CVs so that they bunch up where the curve take sharp turns.

Try the extrusion now.

If you're still not happy, do another rebuild curve but choose Uniform this time and set the number of spans to some high value, like 1000. This will increase the resolution of the extruded tube when extruded from the rebuilt curve.

With the extruded surface selected, go to the channel box and expand the subCurve2 under INPUTS. If you animate the Max Value, you can make an animation of a growing chart. See linked YouTube video for reference.

