**ActinFilamentPlane.py**

#Code template to create invidual market sets and label them by color

lattice\_spacing = 200 # Distance between parallel actin filaments, Angstroms.

actin\_length = 5000 # Actin length in Angstroms.

from math import sqrt

plane\_spacing = sqrt(3)/2 \* lattice\_spacing

radius = 30 # Radius of cylinder depicting actin filament, Angstroms.

# Colors

color1 = (1,1,.5,1) # Light yellow. (red, green, blue, opacity) 0-1 scale

color2 = (.5,.5,1,1) # Light blue.

color3 = (1,0,0,1) #Red

color4 = (0,1,0,1) #Green

color5 = (1,0,1,1) #Purple

color6 = (1,1,0,1)

color7 = (0.5,0.5,0.5,1)

from VolumePath import Marker\_Set, Marker, Link #this is to import the module for Chimera

mset = Marker\_Set('ActinFilamentPlane') #defining the name of the marker set

# Marker(marker\_set\_name, id\_number, (y\_coordinate, x\_coorinate, Z\_coordinate), color, radius)

nx = 40 #number of actin filaments along x axis

# This is first segment in YELLOW. This is the origin.

id = 0

for i in range(nx):

m1\_0 = Marker(mset, id, (i\*lattice\_spacing,0,0), color2, radius)

m2\_0 = Marker(mset, id, (i\*lattice\_spacing,actin\_length,0), color2, radius)

Link(m1\_0, m2\_0, color1, radius) #link the first segment

id += 1

**ActinFilamentCube20x20.py**

#Code template to create invidual market sets and label them by color

lattice\_spacing = 100 # Distance between parallel actin filaments, Angstroms.

actin\_length = 10000 # Actin length in Angstroms.

from math import sqrt

plane\_spacing = sqrt(3)/2 \* lattice\_spacing

radius = 30 # Radius of cylinder depicting actin filament, Angstroms.

# Colors

color1 = (1,1,.5,1) # Light yellow. (red, green, blue, opacity) 0-1 scale

color2 = (.5,.5,1,1) # Light blue.

color3 = (1,0,0,1) #Red

color4 = (0,1,0,1) #Green

color5 = (1,0,1,1) #Purple

color6 = (1,1,0,1)

color7 = (0.5,0.5,0.5,1)

from VolumePath import Marker\_Set, Marker, Link #this is to import the module for Chimera

mset = Marker\_Set('ActinFilamentPlane') #defining the name of the marker set

# Marker(marker\_set\_name, id\_number, (y\_coordinate, x\_coorinate, Z\_coordinate), color, radius)

nx = 20 #number of actin filaments along x axis

# This is first segment in YELLOW. This is the origin.

id = 0

for i in range(nx):

for j in range(nx):

m1\_0 = Marker(mset, id, (i\*lattice\_spacing,0,j \* plane\_spacing), color2, radius)

m2\_0 = Marker(mset, id, (i\*lattice\_spacing,actin\_length,j \* plane\_spacing), color2, radius)

Link(m1\_0, m2\_0, color1, radius) #link the first segment

id += 1

**FixingMarkerID.py**

# Renumber markers giving them unique ids.

# This is to fix a messed up marker set where all ids were the same.

from VolumePath import marker\_sets, Link

for mset in marker\_sets():

markers = mset.markers()

mmap = {}

for m in markers:

mmap[m.atom] = mset.place\_marker(m.xyz(), m.rgba(), m.radius())

links = mset.links()

for l in links:

a1, a2 = l.bond.atoms

Link(mmap[a1], mmap[a2], l.rgba(), l.radius())

for m in markers:

m.delete()

**DivideLinks.py**

# Insert N markers at each selected link. Example Chimera command

#

# runscript dividelinks.py 4

#

from VolumePath import markerset

def divide\_links(links, n):

for link in links:

m1, m2 = link.marker1, link.marker2

mprev = m1

mset = m1.marker\_set

for i in range(n):

f = float(i+1) / (n+1)

xyz = tuple((1-f)\*x1+f\*x2 for x1, x2 in zip(m1.xyz(), m2.xyz()))

m = mset.place\_marker(xyz, link.rgba(), m1.radius())

markerset.Link(mprev, m, link.rgba(), link.radius())

mprev = m

markerset.Link(mprev, m2, link.rgba(), link.radius())

link.delete()

n = int(arguments[0])

links = markerset.selected\_links()

divide\_links(links, n)

**RemoveCross.py**

# When one selected pseudobond crosses another in the current camera view

# remove the longer one.

# Keep every pseudobond that is not crossed by a shorter one

# as viewed along z axis.

def crossing\_pbonds(pbonds):

spbonds = list(pbonds)

spbonds.sort(key = lambda b: b.length())

pb\_cross = []

segments = [] # Segments not crossed by shorter one.

for pb in spbonds:

a1,a2 = pb.atoms

xy1 = a1.xformCoord().data()[:2]

xy2 = a2.xformCoord().data()[:2]

segment = (xy1,xy2)

if any\_cross(segment, segments):

pb\_cross.append(pb)

else:

segments.append(segment)

return pb\_cross

def any\_cross(segment, segments):

for seg2 in segments:

if crossing(seg2, segment):

return True

return False

def crossing(segment1, segment2):

(p1,p2),(p3,p4) = segment1,segment2

if p1 == p3 or p1 == p4 or p2 == p3 or p2 == p4:

return False # Endpoints match, no crossing

return opposite\_sides(p1, p2, segment2) and opposite\_sides(p3, p4, segment1)

# Are two points on opposite sides of a line.

def opposite\_sides(p1, p2, segment):

x1,y1 = p1

x2,y2 = p2

(x3,y3),(x4,y4) = segment

dx,dy = (x4-x3, y4-y3)

nx,ny = (-dy,dx) # Normal vector to segment

side1 = nx\*(x1-x3) + ny\*(y1-y3)

side2 = nx\*(x2-x3) + ny\*(y2-y3)

return (side1 < 0 and side2 > 0) or (side1 > 0 and side2 < 0)

# Use currently selected pseudobonds

from chimera import selection

pbonds = selection.currentPseudobonds()

# Delete longer crossing pseudobonds

cpbonds = crossing\_pbonds(pbonds)

for pb in cpbonds:

pb.pseudoBondGroup.deletePseudoBond(pb)

print ('%d of %d pseudobonds deleted' % (len(cpbonds), len(pbonds)))

**pblengths.py**

# Print lengths of currently selected pseudobonds

from chimera import selection

pbonds = selection.currentPseudobonds()

for pb in pbonds:

a1, a2 = pb.atoms

print ('%15s %15s %8.4g' % (a1.oslIdent(), a2.oslIdent(), pb.length()))

All Scripts that are listed in this documents has been written and provided by Tom Goddard of UCSF Chimera development team, the UCSF Resource for Biocomputing, Visualization, and Informatics (RBVI).