```
import maya.cmds as mc
import maya.OpenMaya as OpenMaya
from functools import partial
import random
import math

'''
04_09_14
- would be good to not include the last verts in the falloff range --
DONE --- use reduceThreshold to change
---------- how to operate only on correct mesh??? Maybe the key is just
to avoid BB object for this script and
other autoprocedure

'''
def distanceBetweenTwoPoints(a=[0, 0, 0], b=[0, 0, 0]):
    distanceX = a[0] - b[0]
    distanceY = a[1] - b[1]
    distanceZ = a[2] - b[2]
    distance = math.sqrt((distanceX * distanceX) + (distanceY *
distanceY) + (distanceZ * distanceZ))
    return distance

def normaliseList(myList):
    OldMax = max(myList)
    OldMin = min(myList)
    NewMax = 1
    NewMin = 0

    OldRange = (OldMax - OldMin)
    NewRange = (NewMax - NewMin)

    normalisedList = []
    for value in myList:
        OldValue = value
        try:
            NewValue = (((OldValue - OldMin) * NewRange) / OldRange) +
NewMin
            normalisedList.append(NewValue)
        except ZeroDivisionError:
            print ">>> cant reduce any further"

    return normalisedList

def getMObjectFromNode(mayaNode):
    ''' Get an MObject from a node name in Maya '''
    if not isinstance(mayaNode, basestring):
        print('Please pass in a string value')
        return None
    # Create a selection list object and add the node
    selectionList = OpenMaya.MSelectionList()
    selectionList.add(mayaNode)
    # Grab the node object from the selection
    result = OpenMaya.MObject()
    selectionList.getDependNode(0, result)
    # Return the MObject
    return result
```

```
def getBoundingBox(mayaNode):
    '''Get the bounding box for the specified node'''
    mObject = getMObjectFromNode(mayaNode)
    dgFn = OpenMaya.MFnDagNode(mObject)
    bbox = dgFn.boundingBox()

    return bbox

def selectWithinBB(mesh, BBmesh):
    includedVerts = []
    vertsFalloffList = []
    mesh = mesh
    BBmesh = BBmesh
    bBoxMobject = getBoundingBox(BBmesh)
    BBpos = mc.xform(BBmesh, query =True, translation=True,
worldSpace=True)
    # print BBpos

    meshMobject = getMObjectFromNode(mesh)
    # get dag path
    selectionList = OpenMaya.MSelectionList()
    selectionList.add(mesh)
    dagPath = OpenMaya.MDagPath()
    selectionList.getDagPath(0, dagPath)

    space = OpenMaya.MSpace.kWorld

    ####### iterate over meshh vert takes an DAG PATH OR MOBJECT!!!!
    vertIter = OpenMaya.MItMeshVertex(dagPath)


    while not vertIter.isDone():
    #     ## vertPos is an MPoint
        vertPos = vertIter.position(space)
        # print vertPos
        #check if its in BB
        inclusionStatus = bBoxMobject.contains(vertPos)
        # print inclusionStatus
        if inclusionStatus == True:
            vertIndex = vertIter.index()
            # print vertIndex
            vertName = '{mesh}.vtx[{vertIndex}]'.format(mesh=mesh,
vertIndex=vertIndex)
            includedVerts.append(vertName)
        vertIter.next()

    #### check that there are some verts within BB
    if len(includedVerts) > 0:
        #### make falloff list
        vertsDists = []
        for vert in includedVerts:
            vertPos = mc.xform(vert, query =True, translation=True,
worldSpace=True)
            distFromBB = distanceBetweenTwoPoints(a=vertPos, b=BBpos)
            vertsDists.append(distFromBB)

        vertsFalloffList = normaliseList(vertsDists)
        # print vertsDists
```

```
        # print vertsFalloffList

    return includedVerts, vertsFalloffList


class deformModel(object):
    #NB it doesnt matter whether I'm working on shape nodes or transforms
for this function

    def __init__(self, parentLayout, duplicateShapeStatus=False):
        self.parentLayout = parentLayout
        ### sets whether we are working on shape nodes or transform nodes
        self.duplicateShapeStatus = duplicateShapeStatus

        self.buildUI()

    def buildUI(self):
        clm1 = 70
        clm2 = 70
        clm3 = 110
        clm8 = (clm1 + clm2 + clm3) / 8

        self.labelNameRow = mc.rowLayout(numberOfColumns=2,
columnWidth2=[(clm1 + clm2), clm3], parent=self.parentLayout)
        mc.text('            Function name :', parent=self.labelNameRow)
        if self.duplicateShapeStatus == True:
            mc.text('deformModel (shape nodes)',
parent=self.labelNameRow)
        else:
            mc.text('deformModel (transform nodes)',
parent=self.labelNameRow)

        separator = mc.separator(style ="none", w=300, h=8,
parent=self.parentLayout)

        ## distance, distanceRandom, directionRandom text fields
        self.attrsLayout = mc.rowLayout(numberOfColumns=8,
parent=self.parentLayout)
        mc.text('distance:', parent=self.attrsLayout, align='right')
        self.distanceTF = mc.textField('distanceTF',
parent=self.attrsLayout, width=clm8, text='1.5')

        mc.text('distRand:', parent=self.attrsLayout, align='right')
        self.distRandTF = mc.textField('distRandTF',
parent=self.attrsLayout, width=clm8, text='1.5')

        mc.text('dirRand:', parent=self.attrsLayout, align='right')
        self.dirRandTF = mc.textField('dirRandTF',
parent=self.attrsLayout, width=clm8, text='0.1')
        # reduction slider or text field
        mc.text('reduction:', parent=self.attrsLayout, align='right')
        self.reductionTF = mc.textField('reductionTF',
parent=self.attrsLayout, width=clm8, text='40')

        separator = mc.separator(style ="none", w=300, h=8,
parent=self.parentLayout)
```

```
    def reduceDeform(self, *args):
        #### create BB if it doesnt exist
        if mc.objExists('BBoxObject'):
            BBobject = 'BBoxObject'
            print ">>> use existing BBobject"
        else:
            BBobject = mc.polyCube(name='BBoxObject', w=10, h=10, d=10)
            BBshape = mc.listRelatives(BBobject, children=True, type =
'mesh')[0]

mc.setAttr("{BBshape}.overrideEnabled".format(BBshape=BBshape), 1)

mc.setAttr("{BBshape}.overrideShading".format(BBshape=BBshape), 0)
            mc.setAttr('BBoxObjectShape.overrideColor', 16)

        #### get selected mesh --> test if its a mesh or transform
        mainMesh = self.getMesh()

        #### get percentage from TF
        reductionString = mc.textField(self.reductionTF, query=True,
text=True)
        if reductionString == '':
            reduction = 0
        else:
            reduction = float(reductionString)
            if reduction > 100 or reduction < 0:
                print ">>> Reduction is a percentage value. Enter a value
between 0 and 100"
                reduction = 0
        print '>>> using reduction value ', reduction

        #### get distance from TF
        distanceString = mc.textField(self.distanceTF, query=True,
text=True)
        distance = float(distanceString)

        distRandString = mc.textField(self.distRandTF, query=True,
text=True)
        distRand = float(distRandString)

        dirRandString = mc.textField(self.dirRandTF, query=True,
text=True)
        dirRand = float(dirRandString)

        if mainMesh != None:
            self.deformMesh(mainMesh, BBobject, distance, dirRand,
distRand)
            self.reduceMesh(mainMesh, BBobject, reduction)

        #### delete history on the model
        mc.delete(mainMesh, constructionHistory=True)

        return mainMesh

    def getMesh(self):
        mainMesh = None

        try:
```

```
            selectedNode = mc.ls(selection = True)[0]
        except IndexError:
            print ">>> nothing selected"
            selectedNode = []

        if selectedNode != []:
            #### get node type
            nodeType = mc.objectType(selectedNode)

            if nodeType == 'transform':
                print '>>> transform selected'
                ##test if its BBoxObject
                if selectedNode == 'BBoxObject':
                    print '>>> BBoxObject selected'
                else:
                    mainMesh = selectedNode

            #### component selected
            elif nodeType == 'mesh':
                print '>>> mesh selected'
                shape = mc.ls(selection=True, shapes=True, dag=True,
objectsOnly=True, long=True)
                # print 'shape == ', shape
                shapeNode = shape[0]
                # print 'meshNode == ', meshNode
                transformList = mc.listRelatives(shapeNode, parent=True,
type='transform')
                try:
                    transformNode = transformList[0]
                except IndexError:
                    print ">>> no transform node"
                    transformNode = None
                # print 'transformNode == ', transformNode
                if transformNode != None:
                    ##test if its BBoxObject
                    if transformNode == 'BBoxObject':
                        print '>>> BBoxObject selected'
                    else:
                        mainMesh = transformNode

            elif nodeType != 'mesh' or selectedNode != 'transform':
                print ">>> neither mesh nor component selected"

        return mainMesh


    def reduceMesh(self, mesh = 'pSphere1', BBobject = 'pCube1',
percentage=40, *args):
        closeVertsList = []
        threshold = 0.25
        #### get current selection
        currentSelection = mc.ls(selection = True)

        includedVerts, vertsFalloffList = selectWithinBB(mesh, BBobject)
        # print includedVerts


        if len(includedVerts) > 0 and len(vertsFalloffList) > 0:
```

```
            #### only use closest verts
            i = 0
            for value in vertsFalloffList:
                if value < threshold:
                    closeVertsList.append(includedVerts[i])

                i += 1

            if len(closeVertsList) > 0:
                faces = mc.polyListComponentConversion(closeVertsList,
toFace=True, fromVertex=True)
                mc.select(faces)
                reduceNode = mc.polyReduce(percentage=percentage,
keepQuadsWeight=1,line=0.0, keepBorder=True, detail=0.0, \
                    keepOriginalVertices=True)[0]
                ### set reduce attrs

mc.setAttr("{reduceNode}.triangulate".format(reduceNode=reduceNode), 0)

mc.setAttr("{reduceNode}.keepQuadsWeight".format(reduceNode=reduceNode),
1)

                ### resume current selection
                mc.select(currentSelection)
        else:
            print ">>> no verts within BB"



    def deformMesh(self, mesh = 'pSphere1', BBobject = 'pCube1', distance
= 1.5, \
        randomDirection = 0.1, randomDistance = 1.5, *args):

        includedVerts, vertsFalloffList = selectWithinBB(mesh, BBobject)
        if len(includedVerts) > 0 and len(vertsFalloffList) > 0:

            numVerts = len(includedVerts)


            for i in range(numVerts):
                ## get vert name
                vert = includedVerts[i]
                ## get vert falloff value
                vertFalloffValue = 1 - vertsFalloffList[i]

                #add random direction
                randomDirAmountX = (random.uniform((randomDirection * -
0.005), (randomDirection * 0.005)))*vertFalloffValue
                randomDirAmountY = (random.uniform((randomDirection * -
0.005), (randomDirection * 0.005)))*vertFalloffValue
                randomDirAmountZ = (random.uniform((randomDirection * -
0.005), (randomDirection * 0.005)))*vertFalloffValue
                # print randomDirAmountX, randomDirAmountY,
randomDirAmountZ
                mc.xform(vert, translation=[randomDirAmountX,
randomDirAmountY, randomDirAmountZ]\
                    , relative=True)
```

```
                ###move random amount along normal
                # dist = distance * vertFalloffValue
                dist = distance
                if randomDistance > 0:
                    rand = (random.uniform(0, randomDistance))
                else:
                    rand = 1

                moveAmount = (dist * rand) * vertFalloffValue
                # print vert, randomDistance
                mc.moveVertexAlongDirection(vert, normalDirection = 1.0 *
moveAmount)
```