```
import maya.cmds as mc
from functools import partial
from operator import itemgetter

'''
29_07_2015
--> duplicate selected shapes under new transform

-- change shaders to Blinn

-- select verts of selected shape nodes
-- box with transform node name
'''



class MeshCntrls(object):

    def __init__(self, parentLayout, workOnTransformNodeStatus=False):
        self.parentLayout=parentLayout
        self.workOnTransformNodeStatus=workOnTransformNodeStatus

        self.workOnChildNodes = True
        self.transformName = "mesh_Transform"
        self.namedTransform = "mesh_Transform1"
        # self.blendOriginIsWorld = True

        self.buildUI()

    def buildUI(self):
        clm1 = 125
        clm2 = 100
        clm3 = 75
        separation = 15

        # text box with name of transform node
        row1 = mc.rowLayout(numberOfColumns=4, parent=self.parentLayout)
        mc.text('tranform node name :', width=clm2+clm3, parent=row1,
align='left')
        self.tranformNameTF = mc.textField('tranformNameTF', parent=row1,
width=clm1, \
                text='mesh_Transform',
enterCommand=self.populateTranformName,
alwaysInvokeEnterCommandOnReturn=True)

        separator = mc.separator(style ="in", w=300, h=separation,
parent=self.parentLayout)
        rowLabel = mc.rowLayout(numberOfColumns=2,
parent=self.parentLayout)
        mc.text('--> work with currently VISIBLE shape nodes <-- ',
parent=rowLabel)

        rowSelect = mc.rowLayout(numberOfColumns=4,
parent=self.parentLayout)
        mc.text('    select currently visible mesh :', width=clm1+clm2,
align="right", parent=rowSelect)
        mc.button(label = 'select', width=clm3,
command=self.selectShapeNode, parent=rowSelect)
```

```
        row2 = mc.rowLayout(numberOfColumns=4, parent=self.parentLayout)
        mc.text('    delete currently visible mesh :', width=clm1+clm2,
align="right")
        mc.button(label = "DELETE", width=clm3,
command=self.deleteShapeNode, parent=row2)

        row3 = mc.rowLayout(numberOfColumns=4, parent=self.parentLayout)
        mc.text('    duplicate visible mesh under new transform :',
width=clm1+clm2, align='right')
        mc.button(label = 'duplicate', width=clm3,
command=self.duplicateVisibleShapeNode, parent=row3)

        rowDupeMesh2 = mc.rowLayout(numberOfColumns=4,
parent=self.parentLayout)
        mc.text('    duplicate visible mesh under NAMED transform :',
width=clm1+clm2, align='right')
        mc.button(label = 'dupe move', width=clm3,
command=self.dupeVisShapeNamedTransform, parent=rowDupeMesh2)
                # text box with NAMED transform
        rowNamedTransform = mc.rowLayout(numberOfColumns=4,
parent=self.parentLayout)
        mc.text('tranform node name :', width=clm2+clm3,
parent=rowNamedTransform, align='left')
        self.namedTransformTF = mc.textField('namedTransformTF',
parent=rowNamedTransform, width=clm1, \
                text='mesh_Transform1',
enterCommand=self.populateNamedTranform,
alwaysInvokeEnterCommandOnReturn=True)


        keyVisRow = mc.rowLayout(numberOfColumns=4,
parent=self.parentLayout)
        mc.text('set visibility keys :', width=clm1+clm2 - clm3,
align='right')
        mc.button(label = 'off', width=clm3, command=self.keyOffVisShape,
parent=keyVisRow)
        mc.button(label = 'On', width=clm3, command=self.keyOnVisShape,
parent=keyVisRow)


        separator = mc.separator(style ="in", w=300, h=separation,
parent=self.parentLayout)
        rowLabel2 = mc.rowLayout(numberOfColumns=2,
parent=self.parentLayout)
        mc.text('--> work with currently SELECTED shape nodes <-- ',
parent=rowLabel2)

        row6 = mc.rowLayout(numberOfColumns=4, parent=self.parentLayout)
        mc.text('    duplicate selected shapes nodes under new transform
:', width=clm1+clm2, align='right')
        mc.button(label = 'duplicate', width=clm3,
command=self.duplicateSelectedShapeNode, parent=row6)


    def populateTranformName(self, *args):
```

```
        self.transformName = mc.textField(self.tranformNameTF,
query=True, text=True)
        # check if node exists and if its a transform
        if mc.objExists(self.transformName) == True:
            print "tranformName == ", self.transformName
            if mc.nodeType(self.transformName) != "transform":
                print "WARNING : NAMED NODE IS NOT A TRANSFORM NODE"
        else:
            print "NODE DOES NOT EXIST"

    def populateNamedTranform(self, *args):
        self.namedTransform = mc.textField(self.namedTransformTF,
query=True, text=True)
        # check if node exists and if its a transform
        if mc.objExists(self.namedTransform) == True:
            print "tranformName == ", self.namedTransform
            if mc.nodeType(self.namedTransform) != "transform":
                print "WARNING : NAMED NODE IS NOT A TRANSFORM NODE"
        else:
            print "NODE DOES NOT EXIST"


    def duplicateVisibleShapeNode(self, *args):
        shapeNodes = getVisibleShapeNode(self.transformName)
        for node in shapeNodes:
            duplicateShapeUnderNewTransform(self.transformName, node)

    def duplicateSelectedShapeNode(self, *args):
        shapeNodeList = getSelectedShapeNode()
        print "havent worked this out yet"
        # for node in shapeNodes:
        #     # duplicateShapeUnderNewTransform(self.transformName, node)

        # # meshShapeLN =
"{transform}|{meshShape}".format(transform=transform,
meshShape=meshShape)
        # newTransform = mc.duplicate(shapeNodeList)[0]
        # print "newTransform == ", newTransform
        # newNodeLN =
"|{newTransform}|{meshShape}".format(newTransform=newTransform,
meshShape=meshShape)
        # # print "newNodeLN == ", newNodeLN
        # # get list of shape nodes
        # shapeNodes = mc.listRelatives(newTransform, fullPath=True,
allDescendents=True)
        # for shape in shapeNodes:
        #     print "shape == ", shape
        #     if shape != newNodeLN:
        #         mc.delete(shape)
        #     else:
        #         print "newShape == ", shapeNodes
        # return shape
```

```
    def dupeVisShapeNamedTransform(self, *args):
        shapeNodes = getVisibleShapeNode(self.transformName)
        for node in shapeNodes:
            duplicateShapeUnderNamedTransform(self.transformName, node,
self.namedTransform)

    def selectShapeNode(self, *args):
        shapeNodes = getVisibleShapeNode(self.transformName)
        for node in shapeNodes:
            mc.select(node, add=True)
            print '>>> NODE ', node, " SELECTED"

    def deleteShapeNode(self, *args):
        shapeNodes = getVisibleShapeNode(self.transformName)
        for node in shapeNodes:
            mc.delete(node, constructionHistory=True)
            mc.delete(node)
            print '>>> NODE ', node, " DELETED"

    def keyOnVisShape(self, *args):
        print ">> executing keyOnVisShape"
        shapeNodes = getVisibleShapeNode(self.transformName)
        for node in shapeNodes:
            setVisOn(self.transformName, node)

    def keyOffVisShape(self, *args):
        print ">> executing keyOffVisShape"
        shapeNodes = getVisibleShapeNode(self.transformName)
        for node in shapeNodes:
            setVisOff(self.transformName, node)


    # #### functions for working on selected nodes




# HELPERS

def duplicateShapeUnderNewTransform(transform, meshShape):
    # change to use long name
    # meshTransform = mc.listRelatives(meshShape, parent=True)
    # newMeshName = "{meshShape}new".format(meshShape=meshShape)

    # duplicate and rename mesh transform with "new" added
    meshShapeLN = "{transform}|{meshShape}".format(transform=transform,
meshShape=meshShape)
    newTransform = mc.duplicate(meshShapeLN)[0]
    # print "newTransform == ", newTransform
    newNodeLN =
"|{newTransform}|{meshShape}".format(newTransform=newTransform,
meshShape=meshShape)
    # print "newNodeLN == ", newNodeLN
    # get list of shape nodes
    shapeNodes = mc.listRelatives(newTransform, fullPath=True,
allDescendents=True)
    for shape in shapeNodes:
        print "shape == ", shape
```

```
        if shape != newNodeLN:
            mc.delete(shape)
        else:
            print "newShape == ", shapeNodes
    return shape

def duplicateShapeUnderNamedTransform(transform, meshShape,
namedTransform):
    # duplicate and delete all shapes except the one in question
    newTransform = mc.duplicate(meshShape)[0]
    print "newTransform == ", newTransform
    newNodeLN =
"|{newTransform}|{meshShape}".format(newTransform=newTransform,
meshShape=meshShape)
    print "newNodeLN == ", newNodeLN

    mc.parent(newNodeLN, namedTransform, relative=True, shape=True)
    # delete newTransform
    mc.delete(newTransform)


def orderList(nodeList):
    extendedNodeList = []

    for node in nodeList:
        nodenameList = node.split('_')

        if nodenameList[-1] == '':
            nodeNumber = 0
        else:
            nodeNumber = int(nodenameList[-1])

        extendedNodeList.append([node,nodeNumber])

    newExtendedNodeList = sorted(extendedNodeList, key=itemgetter(1))
    # print 'newExtendedNodeList == ', newExtendedNodeList

    newNodeList = []
    for item in newExtendedNodeList:
        newNodeList.append(item[0])

    # print 'newNodeList == ', newNodeList
    return newNodeList



def getNodeListFromTransform(transformNode):
    # transformNode = mc.listRelatives(node, parent=True,
type='transform')[0]
    shapeNodeList = mc.listRelatives(transformNode, children=True, type =
'shape')

    #### get rid of mesh_Orig and make new list ##########
    newList = []
    checkString1 = "Orig"
    checkString2 = "polySurfaceShape"
    for node in shapeNodeList:
        if checkString1 in node or checkString2 in node:
```

```
            print '>>> removing ', node
        else:
            newList.append(node)

    nodeList = orderList(newList)
    return nodeList


def getVisibleShapeNode(transformNode):
    shapeNodeList = getNodeListFromTransform(transformNode)
    print "shape Node List == ", shapeNodeList

    visibleShapeList = []
    for shape in shapeNodeList:
        #check if it is visible
        visValue =
mc.getAttr('{transform}|{shape}.visibility'.format(shape=shape,
transform=transformNode))
        if visValue == True:
            visibleShapeList.append(shape)
    print "visible Shape List == ", visibleShapeList
    return visibleShapeList




def setVisOff(meshTransform, meshShape):
    time = mc.currentTime( query=True )
    mc.setKeyframe(
"{transform}|{meshShape}.v".format(transform=meshTransform,
meshShape=meshShape), time = time, value = 0)

def setVisOn(meshTransform, meshShape):
    time = mc.currentTime( query=True )
    mc.setKeyframe(
"{transform}|{meshShape}.v".format(transform=meshTransform,
meshShape=meshShape), time = time, value = 1)

# ################################### EXTRAS ########################
# to cycle through selected shape nodes and parent them to scripted
transform
def utilities():
    shapeList = mc.ls(selection=True)
    transform = "standingFig_Transform"
    for shape in shapeList:
        mc.parent(shape, transform, shape=True, relative=True)


# ##############

# def parentSelectedShapesUnderSelectedTransform(transform):
#     shapeNodeList = getSelectedShapeNode()
#     transform = "standingFig_Transform"
#     for shape in shapeNodeList:
#         mc.parent(shape, transform, shape=True, relative=True)

# def getSelectedShapeNode():
#     shapeNodeList = mc.ls(selection=True)
```

```
#     print "shape Node List == ", shapeNodeList
#     return shapeNodeList

# parentSelectedShapesUnderSelectedTransform(transform)
```