```
import maya.cmds as mc
from functools import partial
import math
import operator
'''
21_06_14
- Add offset Z Attrs to camMAIN, one attr for each camera
    what if these attrs want to be updated? check the new workingCamList;
if camAttr is not in there delete the Attr; if camera doesnt have an
attr, add the attr
- add drift on camMAIN
'''

def amountRotated(rotation1=[], rotation2=[]):
    # print 'rrrrrrrrrr ', rotation1, rotation2
    if rotation1==[]:
        rot1X = 0
        rot1Y = 0
        rot1Z = 0
    else:
        rot1X = rotation1[0]
        rot1Y = rotation1[1]
        rot1Z = rotation1[2]

    if rotation2==[]:
        rot2X = 0
        rot2Y = 0
        rot2Z = 0
    else:
        rot2X = rotation2[0]
        rot2Y = rotation2[1]
        rot2Z = rotation2[2]

    rX = rot1X - rot2X
    rY = rot1Y - rot2Y
    rZ = rot1Z - rot2Z

    roatationAmount = math.sqrt( rX*rX + rY*rY + rZ*rZ )
    return roatationAmount

def distanceBetweenPoints(position1=[], position2=[]):
    if position1==[]:
        posAx = 0
        posAy = 0
        posAz = 0
    else:
        posAx = position1[0]
        posAy = position1[1]
        posAz = position1[2]
    if position2==[]:
        posBx = 0
        posBy = 0
        posBz = 0
    else:
        posBx = position2[0]
        posBy = position2[1]
        posBz = position2[2]
```

```
    dx = posAx - posBx
    dy = posAy - posBy
    dz = posAz - posBz

    distance = math.sqrt( dx*dx + dy*dy + dz*dz )

    return distance


def getNodeListFromGroup(groupNode='nodeGroup', transform=True):
    nodeList = None
    groupNode = groupNode

    if transform == True:
        if mc.objectType(groupNode) != 'transform':
            feedback = '>>> please select group node'
            mc.text(self.feedbackText, edit=True, label=feedback)
        else:
            nodeList = mc.listRelatives(groupNode, children=True)
    return nodeList


class CameraAnim(object):

    def __init__(self, parentLayout):
        self.parentLayout = parentLayout
        self.workingCamera = 'persp'
        self.towardNext = False
        self.fromPrevious = False

        self.duplicateStartStatus = True
        self.duplicateEndStatus = False

        # self.workingCamList = []
        self.constrainFrameList = []

        self.buildUI()

    def buildUI(self):
        parentLayout= self.parentLayout
        clm1 = 140
        clm2 = 90
        clm3 = 90
        separation = 15

        #
        self.workingCamGrpRow = mc.rowLayout(numberOfColumns=3,
columnWidth3=[(clm1), (clm2), (clm3)], parent=parentLayout)
        mc.text('             Working cam group :',
parent=self.workingCamGrpRow, align='right')
        self.camGroupTF = mc.textField('camGroupTF', text='cameraGROUP',
parent=self.workingCamGrpRow)

        ########### camMAIN
#####################################################
        mc.separator(style ="in", h=separation, parent=self.parentLayout)
        mc.text('--- MAIN camera controls ------------------',
parent=self.parentLayout)
```

```
        mc.separator(style ="none", h=10, parent=self.parentLayout)
        self.createCamMainRow = mc.rowLayout(numberOfColumns=3,
columnWidth3=[(clm1), (clm2), (clm3)], parent=parentLayout)
        mc.text('               camMAIN name :',
parent=self.createCamMainRow)
        self.camMAINtransformNameTF =
mc.textField('camMAINtransformNameTF', text='camMAIN1',
parent=self.createCamMainRow)
        createCamButton = mc.button(label='Create camMAIN', width=clm3,
command=self.createRenderCam, parent=self.createCamMainRow)

        self.row2 = mc.rowLayout(numberOfColumns=3, columnWidth3=[(clm1),
(clm2), (clm3)], parent=parentLayout)
        mc.text('               camMAIN offset :', parent=self.row2)
        self.keyOffsetTF = mc.textField('keyOffsetTF', text='0.0',
parent=self.row2)
        camAnimbutton = mc.button(label='Key constraints', width=clm3,
command=self.keyConstraints, parent=self.row2)

        ########### add drift camMAIN
#########################################################
        mc.separator(style ="in", w=100, h=separation,
parent=self.parentLayout)
        addDriftButton = mc.button(label='Add drift on camera MAIN',
command=self.addDriftCamMain, \
            parent=self.parentLayout, w=clm1+clm2+clm3)
        self.snapPointSlider = mc.floatSliderGrp(field=True, label="Drift
amount : ", \
        minValue=0.01, maxValue=0.99, fieldMaxValue=0.99, value=0.8,
parent=self.parentLayout)


        # ##### working cam controls
        mc.separator(style ="in", h=separation, parent=self.parentLayout)
        mc.text('--- working camera controls ------------------',
parent=self.parentLayout)
        mc.separator(style ="none", w=100, h=separation,
parent=self.parentLayout)
        ########### consolidatetWorkingCams
#########################################################
        consolCamButton = mc.button(label='Animate and consolidate
working cameras', command=self.consolidatetWorkingCams, \
            parent=self.parentLayout, w=clm1+clm2+clm3)
        self.distanceThresholdSlider = mc.intSliderGrp(field=True,
label='Distance threshold : ', \
        minValue=1, maxValue=100, fieldMaxValue=10000, value=50,
parent=self.parentLayout)
        self.rotationThresholdSlider = mc.intSliderGrp(field=True,
label='Rotation threshold : ', \
        minValue=1, maxValue=200, fieldMaxValue=360, value=25,
parent=self.parentLayout)


        mc.separator(style ="in", w=100, h=separation,
parent=self.parentLayout)
        ########### animateCurrentCam
#########################################################
```

```
        animCurrentCamButton = mc.button(label='Animate current working
camera', command=self.animateCurrentCam, \
            parent=self.parentLayout, w=clm1+clm2+clm3)
                #### create radio buttons
        radioCollection1 = mc.radioCollection()
        RBlayout = mc.rowLayout(numberOfColumns=4, columnWidth4=[110,
110, 55, 20], parent=self.parentLayout)
        self.towardNextRB = mc.radioButton(label='toward next',
parent=RBlayout, select=False, \
            onCommand=(partial(self.setTowardNextStatus, True)))
        self.fromPrevioustRB = mc.radioButton(label='from previous',
parent=RBlayout, select=False, \
            onCommand=(partial(self.setFromPreviousStatus, True)))
        mc.text('Ease (1-4):', parent=RBlayout)
        self.easeTF = mc.textField('easeTF', text='4', parent=RBlayout,
w=20)


        mc.separator(style ="in", w=100, h=separation,
parent=self.parentLayout)
        ########### insertKeysCurrentCam
#####################################################
        insertKeysCurrentCamButton = mc.button(label='Insert keys on
current working camera', \
            command=self.insertKeysCurrentCam, parent=self.parentLayout,
w=clm1+clm2+clm3)
        radioCollection2 = mc.radioCollection()
        RBlayout2 = mc.rowLayout(numberOfColumns=4, columnWidth4=[110,
110, 55, 20], parent=self.parentLayout)
        self.duplicateStartRB = mc.radioButton(label='duplicate start
keys', parent=RBlayout2, select=True, \
            onCommand=(partial(self.setDuplicateStartStatus, True)))
        self.duplicateEndRB = mc.radioButton(label='duplicate end keys',
parent=RBlayout2, select=False, \
            onCommand=(partial(self.setDuplicateStartStatus, False)))

        self.feedbackText = mc.text( label='', parent=parentLayout)

    def getWorkingCamList(self):
        camMAIN =  mc.textField(self.camMAINtransformNameTF, query=True,
text=True)
        workingCamList = []
        transformList = []

        ### get camList from camMAIN constraint
        connectedNodeList =
mc.listConnections('{camMAIN}_parentConstraint1'.format(camMAIN=camMAIN)\
            , destination=False, source=True)

        ### get rid of doubles in teh list
        connectedNodes = list(set(connectedNodeList))
        print 'connectedNodes == ', connectedNodes

        for node in connectedNodes:
            if mc.objectType(node) == 'transform':
                transformList.append(node)

        for transform in transformList:
```

```
            if transform == camMAIN:
                transformList.remove(transform)

        workingCamList = transformList
        return workingCamList

    def getCameraTransforms(self, *args):
        cameraPosList = []
        cameraRotList = []

        for item in ['x','y','z']:
            position =
mc.getAttr('{camera}.t{item}'.format(camera=self.workingCamera,
item=item))
            cameraPosList.append(position)

            rotation =
mc.getAttr('{camera}.r{item}'.format(camera=self.workingCamera,
item=item))
            cameraRotList.append(rotation)

        return cameraPosList, cameraRotList


    def makePosRotDict(self, camList, constrainTimeDic):
        workingCamList = camList
        constrainTimeDic = constrainTimeDic
        #### make empty dict
        posRotDict = dict()
        # ontimePos = []
        # offTimePos = []
        # ontimeRot = []
        # offTimeRot = []

        #### for each camera in teh list get teh position and rotations
at timeOn and timeOff
        for camera in workingCamList:
            onTime = constrainTimeDic[camera][0]
            offTime = constrainTimeDic[camera][1]
            print '>>>>>>>>> for camera ', camera, 'onTime is ', onTime,
'offTime is ', offTime

            if onTime == None:
                ontimePos = []
                ontimeRot = []
            else:
                #### get ontime details
                ontimePosX =
mc.getAttr('{camera}.tx'.format(camera=camera), time=onTime)
                ontimePosY =
mc.getAttr('{camera}.ty'.format(camera=camera), time=onTime)
                ontimePosZ =
mc.getAttr('{camera}.tz'.format(camera=camera), time=onTime)
                ontimePos = [ontimePosX, ontimePosY, ontimePosZ]

                ontimeRotX =
mc.getAttr('{camera}.rx'.format(camera=camera), time=onTime)
```

```
                ontimeRotY =
mc.getAttr('{camera}.ry'.format(camera=camera), time=onTime)
                ontimeRotZ =
mc.getAttr('{camera}.rz'.format(camera=camera), time=onTime)
                ontimeRot = [ontimeRotX, ontimeRotY, ontimeRotZ]

            if offTime == None:
                offtimePos = []
                offtimeRot = []
            else:
                #### get offtime details
                offtimePosX =
mc.getAttr('{camera}.tx'.format(camera=camera), time=onTime)
                offtimePosY =
mc.getAttr('{camera}.ty'.format(camera=camera), time=onTime)
                offtimePosZ =
mc.getAttr('{camera}.tz'.format(camera=camera), time=onTime)
                offtimePos = [offtimePosX, offtimePosY, offtimePosZ]

                offtimeRotX =
mc.getAttr('{camera}.rx'.format(camera=camera), time=onTime)
                offtimeRotY =
mc.getAttr('{camera}.ry'.format(camera=camera), time=onTime)
                offtimeRotZ =
mc.getAttr('{camera}.rz'.format(camera=camera), time=onTime)
                offtimeRot = [offtimeRotX, offtimeRotY, offtimeRotZ]

            posRotDict[camera] = [ontimePos, ontimeRot, offtimePos,
offtimeRot]

        return posRotDict

    def makeConstrainTimeDict(self, cameraList):
        camMAIN =  mc.textField(self.camMAINtransformNameTF, query=True,
text=True)
        workingCamList = cameraList
        constrainTimeList = []
        constrainTimeDict = dict()

        for camera in workingCamList:
            ## get the target index
            destTranslatePlug =
mc.connectionInfo('{camera}.translate'.format(camera=camera)\
                , destinationFromSource=True)[0]

            ### get rid of all except key number
            x =
destTranslatePlug.replace('{camMAIN}_parentConstraint1.target['.format(ca
mMAIN=camMAIN), '')
            keyNum = x.replace('].targetTranslate', '')

            camConstrainKeyTimes =
mc.keyframe('{camMAIN}_parentConstraint1_{camera}W{keyNum}'\
                .format(camMAIN=camMAIN, camera=camera, keyNum=keyNum),
query=True)

            ### get the constrainOn and constrainOff time based on a
search for teh value of 1
```

```
            ##set initial on and off times
            constrainOnTime = None
            constrainOffTime = None
            ontimeOfftime = []

            index = 0
            for keyTime in camConstrainKeyTimes:
                value =
mc.keyframe('{camMAIN}_parentConstraint1_{camera}W{keyNum}'\
                    .format(camMAIN=camMAIN, camera=camera,
keyNum=keyNum),\
                t=(keyTime, keyTime), q=True, eval=True)[0]

                # print '>>>>>>>>>>>  camera == ', camera, 'keyTime == ',
keyTime, 'value == ', value
                if value >= 1.0:
                    constrainOnTime = keyTime
                    ### what if there is no index+1 ???
                    if index <= (len(camConstrainKeyTimes)-2):
                        constrainOffTime = camConstrainKeyTimes[index+1]

                index += 1

                ontimeOfftimeKeyNum = [constrainOnTime, constrainOffTime,
keyNum]
                # print ' ontimeOfftime == ', ontimeOfftime

            ### maybe make a dictionary with camera as the key and
ontimeOfftime as the value
            constrainTimeDict[camera] = ontimeOfftimeKeyNum

            # print 'destTranslatePlug == ', destTranslatePlug
            # print 'keyNum == ', keyNum
            # print 'camera == ' , camera
            # # print 'camConstrainKeyValues == ', camConstrainKeyTimes
            # print 'constrainOnTime == ', constrainOnTime,
'constrainOffTime == ', constrainOffTime
        return constrainTimeDict

    def consolidatetWorkingCams(self, *args):
        ############### NOT WORKING PREDICTABLY
        ####get distance threshold  #### get rotation threshold
        distanceThreshold = mc.intSliderGrp(self.distanceThresholdSlider,
query=True, value=True)
        rotationThreshold = mc.intSliderGrp(self.rotationThresholdSlider,
query=True, value=True)

        ### get a list of working cameras
        workingCamList = self.getWorkingCamList()
        # print 'workingCamList == ', workingCamList

        ### get a list of constrain Times
        constrainTimeDict = self.makeConstrainTimeDict(workingCamList)
        # print 'constrainTimeDict == ', constrainTimeDict

        ### get a list of translations and rotations at ontime and at
offtime
        ### format = cam; ontimePos, ontimeRot, offtimePos, offtimeRot
```

```
        posRotDict = self.makePosRotDict(workingCamList,
constrainTimeDict)
        # print 'posRotDict == ', posRotDict

        ################ NB. cam List is not in order... I need to sort
on basis of constainOnTime
        sortedCamList = self.sortCamList(constrainTimeDict)
        print '>>>> sortedCamList == ', sortedCamList

        ## this will compare one value to the next and remove value from
the list if its exceeded
        self.compareNext(sortedCamList, posRotDict, constrainTimeDict,
distanceThreshold, rotationThreshold)


    def compareNext(self, cameraList, posRotDict, constrainTimeDict,
distanceThreshold, rotationThreshold):
        # print 'posRotDict == ', posRotDict

        if not cameraList:
            newCameraList = []
        else:
            # print 'operating on list ',  cameraList
            listLen = len(cameraList)
            currentCam = cameraList[0]
            # print 'currentCam ', currentCam

            if listLen > 1:
                nextCam = cameraList[1]
            else:
                nextCam = cameraList[0]
            # print 'nextCam ', nextCam

            ### get offTimePos and offTimeRot of current cam
            currentCamOffPos,  currentCamOffRot =
posRotDict[currentCam][2], posRotDict[currentCam][3]
            # print ' currentCamOffPos,  currentCamOffRot ',
currentCamOffPos,  currentCamOffRot
            ### get onTime and onTimeRot of next cam
            nextCamOnPos,  nextCamOnRot = posRotDict[nextCam][0],
posRotDict[nextCam][1]
            # print ' nextCamOnPos,  nextCamOnRot ', nextCamOnPos,
nextCamOnRot

            ## get distanceBetween offTimePosCurrentCam and
onTimePosNextCam
            distanceBetween = distanceBetweenPoints(currentCamOffPos,
nextCamOnPos)
            ### get rotationAmount offTimeRotCurrentCam and
onTimeRotNextCam
            rotAmount = amountRotated(currentCamOffRot, nextCamOnRot)
            # print 'distanceBetween and rotAmount == ', distanceBetween,
rotAmount

            if distanceBetween > distanceThreshold or rotAmount >
rotationThreshold:
                self.consolidateCams(currentCam, nextCam,
constrainTimeDict, posRotDict)
```

```
                ########## remove the next (consolidated) camera
                cameraList.remove(nextCam)
                print '>>>>    removed ', nextCam

            newCameraList = self.compareNext(cameraList[1:], posRotDict,
constrainTimeDict, distanceThreshold, rotationThreshold)
        return newCameraList

    def consolidateCams(self, currentCam, nextCam, constrainTimeDict,
posRotDict):
        camMAIN =  mc.textField(self.camMAINtransformNameTF, query=True,
text=True)
        # print 'consolidating      >>>', currentCam, nextCam

        connectionLists = mc.listConnections(nextCam, destination=True,
plugs=True, connections=True)
        numConnections = (len(connectionLists))*0.5

        for i in range(numConnections):
            camAttr = connectionLists[i*2]
            constraintAttr = connectionLists[(i*2)+1]
            print '>>>> disconnecting attrs;    ', camAttr,
constraintAttr
            mc.disconnectAttr(camAttr, constraintAttr)

        ### shift currentCamOfftime to nextCamOffTime set key on
        # mc.keyframe('camMAIN2_parentConstraint1_cam1W0',
time=(204,204),timeChange=120)

        ###### currentTime = currentCamOffTime; newTime = nextCamOffTime
        currentTime =  constrainTimeDict[currentCam][1]
        keyNum = constrainTimeDict[currentCam][2]
        newTime = constrainTimeDict[nextCam][1]
        print '>>>> currentTime keyNum  newTime ', currentTime, keyNum,
newTime
        if currentTime == None or keyNum == None or newTime == None:
            print ' >>>> not laying keys cause there is a None value'
        else:
            attrToShift =
'{camMAIN}_parentConstraint1_{currentCam}W{keyNum}'\
            .format(camMAIN=camMAIN, currentCam=currentCam,
keyNum=keyNum)
            mc.keyframe(attrToShift,
time=(currentTime,currentTime),timeChange=newTime)

            ### keyframe position and rotation of currentCam at
constrOnTimeCurrentCam
            constrOnTimeCurrentCam = constrainTimeDict[currentCam][0]
            posValue1 = posRotDict[currentCam][0]
            rotValue1 = posRotDict[currentCam][1]
            self.setPosRotKeys(currentCam=currentCam,
time=constrOnTimeCurrentCam, \
                posValue=posValue1, rotValue=rotValue1)
                ### keyframe position and rotation of currentCam at
constrOffTimeNextCam
            constrOffTimeNextCam = constrainTimeDict[nextCam][1]
            posValue2 = posRotDict[nextCam][2]
```

```
            rotValue2 = posRotDict[nextCam][3]
            self.setPosRotKeys(currentCam=currentCam,
time=constrOffTimeNextCam, \
                posValue=posValue2, rotValue=rotValue2)


    def setPosRotKeys(self,currentCam, time, posValue, rotValue):
        print '>>>>>>>>>>>>> setting PosRotKeys '
        intTan = 'linear'
        outTan = 'linear'
        ####### set pos keys
        mc.setKeyframe(currentCam, at='tx', inTangentType=intTan,
outTangentType=outTan, value=posValue[0], time =time)
        mc.setKeyframe(currentCam, at='ty', inTangentType=intTan,
outTangentType=outTan, value=posValue[1], time =time)
        mc.setKeyframe(currentCam, at='tz', inTangentType=intTan,
outTangentType=outTan, value=posValue[2], time =time)

        ##set rot keys
        mc.setKeyframe(currentCam, at='rx', inTangentType=intTan,
outTangentType=outTan, value=rotValue[0], time =time)
        mc.setKeyframe(currentCam, at='ry', inTangentType=intTan,
outTangentType=outTan, value=rotValue[1], time =time)
        mc.setKeyframe(currentCam, at='rz', inTangentType=intTan,
outTangentType=outTan, value=rotValue[2], time =time)

    def sortCamList(self, constrainTimeDict):
        constrainTimeDict = constrainTimeDict
        sortedTimeLists = sorted(constrainTimeDict.iteritems(),
key=operator.itemgetter(1))

        sortedCamList = []
        for entry in sortedTimeLists:
                ### add the cam to sortedCamList
                sortedCamList.append(entry[0])

        return sortedCamList


    def setTowardNextStatus(self, towardNext=False, *args):
        self.towardNext = towardNext

    def setFromPreviousStatus(self, fromPrevious=False, *args):
        self.fromPrevious = fromPrevious

    def animateCurrentCam(self, *args):
        print 'animates current working cameras'
        print 'self.towardNext == ', self.towardNext
        print 'self.fromPrevious == ', self.fromPrevious


    def setDuplicateStartStatus(self, duplicateStartStatus=False, *args):
        if duplicateStartStatus == True:
            self.duplicateStartStatus = True
            self.duplicateEndStatus = False

        else:
            self.duplicateStartStatus = False
```

```
            self.duplicateEndStatus = True

    def insertKeysCurrentCam(self, *args):
        print 'inserts keyframes on current working camera'
        print 'duplicateStartStatus is ', self.duplicateStartStatus
        print 'duplicateEndStatus is ', self.duplicateEndStatus

    ############## camMAIN stuff
########################################
    def createRenderCam(self, *args):
        '''
        creates and constrains camMAIN
        '''
        cameraName = mc.textField(self.camMAINtransformNameTF,
query=True, text=True)

        ###get working cam (persp) transform details
        cameraPosList = self.getCameraTransforms()[0]
        cameraRotList = self.getCameraTransforms()[1]
        #### ########### !!! create new camera
        camMAINtransform = mc.camera(name=cameraName,
displayFilmGate=True, displayResolution=True, \
            position=(cameraPosList[0], cameraPosList[1],
cameraPosList[2]),\
            rotation=(cameraRotList[0], cameraRotList[1],
cameraRotList[2]))[0]
        # self.camMAINshape = mc.listRelatives(self.camMAINtransform,
children=True, shapes=True, fullPath=True)[0]
        ############## set attrs on teh camMAIN
        self.setCameraAttrs(camMAINtransform)

        ########### get a list of the working cams from selected group
node - can I test that group node is selected?
        workingCamGroupNode = mc.textField(self.camGroupTF, query=True,
text=True)
        workingCamList =
getNodeListFromGroup(groupNode=workingCamGroupNode, transform=True)

        if workingCamList == None:
            feedback = '>>> no camera list found'
            mc.text(self.feedbackText, edit=True, label=feedback)
        else:

            i = 0
            for workingCam in workingCamList:
                feedback = '>>> working Cam {i} is {workingCam}
'.format(i=i, workingCam=workingCam)
                # mc.text(self.feedbackText, edit=True, label=feedback)
                print feedback
                mc.parentConstraint(workingCam, camMAINtransform,
maintainOffset=False, weight=1)
                i += 1
        print '>>> {camMAINtransform} render camera (camMAIN)
successfully created with
constraints'.format(camMAINtransform=camMAINtransform)
        # return self.camMAINtransform
```

```
    def setCameraAttrs(self, cameraTransform):
        #
mc.setAttr("{cameraTransform}.displayFilmGate".format(cameraTransform=cam
eraTransform), 1)
        #
mc.setAttr("{cameraTransform}.displayResolution".format(cameraTransform=c
ameraTransform), 1)

mc.setAttr("{cameraTransform}.overscan".format(cameraTransform=cameraTran
sform), 1.2)
        #
mc.setAttr("{cameraTransform}.v".format(cameraTransform=cameraTransform),
1)
        print 'camMAIN attrs set'


    def getKeyOnOffTimes(self, node, visKeyList):
        visKeyList = visKeyList
        node = node
        ##set initial on and off times
        visKeyOnTime = 0
        visKeyOffTime = 0

        index = 0
        for keyTime in visKeyList:
            value =  mc.keyframe(node, at='v',t=(keyTime, keyTime),
q=True, eval=True)[0]
            print 'value = ', value
            if value == 1.0:
                visKeyOnTime = keyTime
                visKeyOffTime = visKeyList[index+1]

            index += 1

        return visKeyOnTime, visKeyOffTime

    def keyConstraints(self, *args):
        ##### freshly get camMAIN name from the UI
        camMAINtransform = mc.textField(self.camMAINtransformNameTF,
query=True, text=True)

        #### freshly get the workingCamList
        workingCamGroupNode = mc.textField(self.camGroupTF, query=True,
text=True)
        workingCamList =
getNodeListFromGroup(groupNode=workingCamGroupNode, transform=True)

        constrainTimeList =
self.getConstrainTimeList(workingCamList=workingCamList)

        if len(constrainTimeList) == 0:
            feedback = '>>> constrainTimeList is empty'
            mc.text(self.feedbackText, edit=True, label=feedback)
        if constrainTimeList == None:
            feedback = '>>> constrainTimeList == None'
            mc.text(self.feedbackText, edit=True, label=feedback)
        else:
            index = 0
```

```
            for workingCam in workingCamList:
                ## delete the camera constraint keys if they already
exist

mc.cutKey('{camMAINtransform}_parentConstraint1.w{i}'.format(camMAINtrans
form=camMAINtransform, i=index))
                #### >>>>>>>>>>>>    No object matches name:
camMAIN_parentConstraint1.w0

                ### set constraint key at frame 0

mc.setKeyframe('{camMAINtransform}_parentConstraint1.{workingCam}W{i}'.fo
rmat(camMAINtransform=camMAINtransform, \
                    workingCam=workingCam, i=index), time = 0, value = 0)
                ######  >>>>>>>>>>>>>>>>>>       No object matches name:
camMAIN_parentConstraint1.cam1W0

                ### set constraint ON key
                constrainOnTime = constrainTimeList[index]

mc.setKeyframe('{camMAINtransform}_parentConstraint1.{workingCam}W{i}'.fo
rmat(camMAINtransform=camMAINtransform, \
                    workingCam=workingCam, i=index), time =
constrainOnTime, value = 1)

                if index <= (len(workingCamList)-2):
                    ### set constraint OFF key
                    constrainOffTime = constrainTimeList[index+1]

mc.setKeyframe('{camMAINtransform}_parentConstraint1.{workingCam}W{i}'.fo
rmat(camMAINtransform=camMAINtransform, \
                        workingCam=workingCam, i=index), time =
constrainOffTime, value = 0)



                ## set curve tangents
                mc.keyTangent(
'{camMAINtransform}_parentConstraint1'.format(camMAINtransform=camMAINtra
nsform, \
                        workingCam=workingCam), outTangentType='step')

                index += 1

    def getConstrainTimeList(self, workingCamList):
        '''
        get the visOn and visOff frame number for the mesh corresponding
to each camera
        '''
        workingCamList = workingCamList
        constrainTimeList = []

        ## get the cam key offset value as a float < 1
        keyOffset = float(mc.textField(self.keyOffsetTF, query=True,
text=True))

        if len(workingCamList) == 0:
            feedback = '>>> working camera list is empty'
```

```
            mc.text(self.feedbackText, edit=True, label=feedback)
        else:
            index = 0
            for workingCam in workingCamList:
                # get cameras corresponding Mesh
                meshNumber =
mc.getAttr('{workingCam}.meshNumber'.format(workingCam=workingCam))
                meshName =
mc.getAttr('{workingCam}.meshName'.format(workingCam=workingCam))
                fullmMeshName =
'{meshName}_{meshNumber}'.format(meshName=meshName,
meshNumber=meshNumber)
                ## get that mesh's vis on and vis off
                visKeyList =
mc.keyframe('{fullmMeshName}.v'.format(fullmMeshName=fullmMeshName),
query=True)

                # print self.getKeyOnOffTimes(node=fullmMeshName,
visKeyList=visKeyList)
                visKeyOnTime, visKeyOffTime =
self.getKeyOnOffTimes(node=fullmMeshName, visKeyList=visKeyList)

                print 'visKeyList == ', visKeyList
                print 'fullmMeshName == ', fullmMeshName
                print 'visKeyOnTime == ', visKeyOnTime
                print 'visKeyOffTime == ', visKeyOffTime

                offsetAmount = (visKeyOffTime - visKeyOnTime) * keyOffset
                camConstrainTime = visKeyOnTime + offsetAmount

                print 'offsetAmount is ', offsetAmount
                constrainTimeList.append(camConstrainTime)

                index += 1

        print 'offsetAmount is ', offsetAmount
        return constrainTimeList

    ########## add DRIFT
    def addDriftCamMain(self, *args):
        print 'addDriftCamMain'
        ### get camMAIN
        camMAIN =  mc.textField(self.camMAINtransformNameTF, query=True,
text=True)
        #### change keys on parentConstraint to linear
        ####  camMAIN1_parentConstraint1_cam1W0
        ########## get each parentConstraint curve
        self.getConstraintKeysFromCam(camMAIN)

        ### get each of the curves on the parentConstraint
        ### add a key prior to offKey on currentCam; value = driftValue
        ### add a key prior to onKey on nexCam; value = 1 - driftValue

    def getConstraintKeysFromCam(self, camera):
        ### get constraint
        camConstraint = mc.listRelatives(camera, children=True,
type='constraint')
        print camMAIN
```

```
class TransparencyAnim(object):
    ### constrain transparency projector to working camera locations
    ### can I also use working cam positions etc???
    #### maybe this should be part of previous class ?????????

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

        self.buildUI()

    def buildUI(self):
        parentLayout= self.parentLayout

        mc.separator(style ="in", h=separation, parent=self.parentLayout)
        mc.text('- create traparency based on working cams -',
parent=self.parentLayout)
        # mc.separator(style ="none", h=10, parent=self.parentLayout)
        # self.createCamMainRow = mc.rowLayout(numberOfColumns=3,
columnWidth3=[(clm1), (clm2), (clm3)], parent=parentLayout)
        # mc.text('               camMAIN name :',
parent=self.createCamMainRow)
        # self.camMAINtransformNameTF =
mc.textField('camMAINtransformNameTF', text='camMAIN1',
parent=self.createCamMainRow)
        # createCamButton = mc.button(label='Create camMAIN', width=clm3,
command=self.createRenderCam, parent=self.createCamMainRow)
```