```
import maya.cmds as mc
import math
# import maya.OpenMaya as om

'''
14_06_14

'''


def distanceBetweenPoints(position1=[], position2=[]):
    posAx = position1[0]
    posAy = position1[1]
    posAz = position1[2]

    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


class autoCameraAnim(object):

    def __init__(self, parentLayout):
        self.parentLayout = parentLayout
        self.workingCamera = 'persp'
        self.attribute = 'tz'

        self.newCameraNumber = 0
        self.previousCamPosList = [0,0,0]

        self.buildUI()

    def buildUI(self):
        clm1 = 70
        clm2 = 70
        clm3 = 200

        #the distance needed to travel before a new camera is made
        self.distanceThresholdSlider = mc.intSliderGrp(field=True,
label='Distance threshold : ',\
        minValue=1, maxValue=60, fieldMaxValue=1000, value=10,
parent=self.parentLayout )

        self.buttonRow = mc.rowLayout(numberOfColumns=4,
columnWidth4=[((clm1+clm2) *0.5), \
            ((clm1+clm2) *0.5),(clm3*0.5), (clm3*0.5)],
parent=self.parentLayout)

        mc.button('START', command=(self.createScriptJob),
width=clm3*0.25, parent=self.buttonRow)
```

```
        mc.button('STOP', command=(self.killScriptJob), width=clm3*0.25,
parent=self.buttonRow)

        # mc.button('TEST', command=(self.scriptedJob), width=clm3*0.25,
parent=self.buttonRow)

        #add manual execute function
        manualButton = mc.button('execute function',
command=(self.scriptedJob), width=(clm1+clm2), parent=self.buttonRow)

        # self.procedureRow = mc.rowLayout(numberOfColumns=2,
columnWidth2=[(clm1 ), clm3], parent=self.parentLayout)
        # mc.text('  Function :', parent=self.procedureRow, align='left')
        # self.procedureTF = mc.textField('functionText',
parent=self.procedureRow, width=clm3, text=self.defaultFunction)

        ### feedback text
        self.feedbackText = mc.text('  Feedback:',
parent=self.parentLayout)


    def createScriptJob(self, *args):
        self.startTime = mc.timerX()
        # create job to execute when the perspective camera t? is changed
        self.myJob =
mc.scriptJob(attributeChange=['{camera}.{attribute}'.format(camera=self.w
orkingCamera, attribute=self.attribute)\
            , self.scriptedJob], parent=self.parentLayout,
protected=True)
        ### print feedback text
        feedback = '>>>> Autosave Activated >>>> creating ScriptJob.
Start Time = {time}'.format(time=self.startTime)
        mc.text(self.feedbackText, edit=True, label=feedback)


    def killScriptJob(self, *args):
        mc.scriptJob(kill=self.myJob, force=True)
        feedback = ">>>> Autosave Deactivated >>>>"
        mc.text(self.feedbackText, edit=True, label=feedback)

    def scriptedJob(self, *args):
        distanceThreshold = mc.intSliderGrp(self.distanceThresholdSlider,
query=True, value=True)
        print 'distanceThreshold is ', distanceThreshold
        cameraPosList = self.getCameraTransforms()[0]
        cameraRotList = self.getCameraTransforms()[1]

        #get the distance between two points
        distanceTravelled =
distanceBetweenPoints(self.previousCamPosList, cameraPosList)
        ### if the distanceTravelled is larger than distance threshold
make a camera
        if distanceTravelled > distanceThreshold:
            print ' distance excedes threshold'
            cameraNumber = self.newCameraNumber + 1
            cameraName = 'cam{number}'.format(number=cameraNumber)

            #### ########## OR duplicate the perspective camera
```

```
            newCamTransform = mc.duplicate(self.workingCamera,
name=cameraName)[0]
            newCamShape = mc.listRelatives(newCamTransform,
children=True, shapes=True, fullPath=True)[0]
            print 'newCamShape is ', newCamShape
            ### make group for cameras if it doesnt exist
            if mc.objExists('cameraGROUP') == False:
                cameraGroup = mc.group(empty=True, name='cameraGROUP')
            else:
                cameraGroup = 'cameraGROUP'
            ### put camera in group
            mc.parent(newCamTransform, cameraGroup)

            self.setCameraAttrs(newCamTransform, cameraNumber)



            ## add 1 to self.newCameraNumber
            self.newCameraNumber = cameraNumber


        ### #### last thing is to store as "previous camera position
list"
        self.previousCamPosList = cameraPosList

        print distanceTravelled

    def setCameraAttrs(self, newCamTransform, cameraNumber):
        print 'setCameraAttrs -- newCamTransform is ', newCamTransform


mc.setAttr("{newCamTransform}.displayFilmGate".format(newCamTransform=new
CamTransform), 1)

mc.setAttr("{newCamTransform}.displayResolution".format(newCamTransform=n
ewCamTransform), 1)

mc.setAttr("{newCamTransform}.overscan".format(newCamTransform=newCamTran
sform), 1.2)


mc.setAttr("{newCamTransform}.v".format(newCamTransform=newCamTransform),
1)
        ### add attr to store lookAtMesh
        mc.addAttr(newCamTransform, dataType="string",
longName='correspondingMesh', storable=True)
        ## set the correspondingMesh attr to the correspondingMesh
        meshName =
'mesh_{cameraNumber}'.format(cameraNumber=cameraNumber)

mc.setAttr('{newCamTransform}.correspondingMesh'.format(newCamTransform=n
ewCamTransform), \
            meshName, type='string')
        print
mc.getAttr('{newCamTransform}.correspondingMesh'.format(newCamTransform=n
ewCamTransform))

    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 launchUI():
    clm1 = 50
    clm2 = 50
    clm3 = 100
    separation = 15

    if mc.window('autoCamera', query=True, exists=True):
        mc.deleteUI('autoCamera')
    myWindow = mc.window('autoCamera', title='Create Automatic Camera
Animation', w=(clm1+clm2+clm3), h=100)
    mc.showWindow(myWindow)

    mainColumn = mc.columnLayout()
    # ### add colour faces tool to the UI
    autoCameraAnim(parentLayout=mainColumn)

    mc.showWindow(myWindow)

launchUI()
```