```
import maya.cmds as mc
import math

'''
02_09_14
- add secondary function to deform mesh at intervals (eg every 20th
execution or timestep)
'''
def amountRotated(rotation1=[], rotation2=[]):
    ### same as distance tween two points
    rot1X = rotation1[0]
    rot1Y = rotation1[1]
    rot1Z = rotation1[2]

    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=[]):
    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 AutomaticProcedures(object):
    ### this one is changed from using selectionChanged event to using
attribut change on perspective camera

    def __init__(self, parentLayout=0, functionObject=0, functionText =
"mc.polyCube()", createCameras=False):
        self.parentLayout = parentLayout
        self.functionObject = functionObject
        ############### nb. functionReturn value for dupemesh is
currentMeshNumber
        self.functionReturnValue = ['meshName', 0]

        #### when the persp camera tz attr is changed the scripted job is
executed
        #### its also the persp camera that is duplicated
```

```
        self.workingCamera = 'persp'
        self.attribute = 'tz'

        #### for function execution (which happens after a certain time
period)
        self.functionText = functionText
        self.lastExecutionTime = 0

        #### for camera creation (which happens after a certain distance
is covered)
        self.createCameras = createCameras
        self.newCameraNumber = 0
        self.previousCamPosList = [0, 0, 0]
        self.previousCamRotList = [0, 0, 0]

        self.buildUI()


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

        mc.text('--- Controls for automatic execution of the above
function ---', parent=self.parentLayout, align='left')
        self.secondsSlider = mc.intSliderGrp(field=True, label='Seconds
between function : ', \
        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)

        ##add manual execute function
        manualButton = mc.button('execute function',
command=(self.executeFunction), 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.functionText)

        ### feedback text
        self.feedbackText = mc.text('>>>> model function feedback',
parent=self.parentLayout)

        ############## camera UI stuff
        if self.createCameras == True:
```

```
            mc.text('----  Camera ---------------',
parent=self.parentLayout, align='left')
            # separator = mc.separator(style ="none", w=300, h=15,
parent=self.parentLayout)
                    #the distance needed to travel before a new camera is
made
            self.distanceThresholdSlider = mc.intSliderGrp(field=True,
label='Distance threshold : ',\
            minValue=1, maxValue=100, fieldMaxValue=1000, value=10,
parent=self.parentLayout )
            self.rotationThresholdSlider = mc.intSliderGrp(field=True,
label='Rotation threshold : ',\
            minValue=1, maxValue=100, fieldMaxValue=400, value=10,
parent=self.parentLayout )

            self.buttonRow = mc.rowLayout(numberOfColumns=3,
columnWidth3=[(clm1+clm2),(clm3*0.5), (clm3*0.5)], \
                parent=self.parentLayout)
            #add manual execute function
            mc.text("create new working cam\'", parent=self.buttonRow)
            # mc.text('spacer2', parent=self.buttonRow)
            manualCameraButton = mc.button('create camera',
command=(self.createCamera), width=(clm1+clm2), parent=self.buttonRow)
            ### feedback text
            self.cameraFeedbackText = mc.text('>>>> camera animation
feedback', parent=self.parentLayout)

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

    def createScriptJob(self, *args):
        self.startTime = mc.timerX()
        self.myJob =
mc.scriptJob(attributeChange=['{camera}.{attribute}'.format(camera=self.w
orkingCamera, attribute=self.attribute),\
            self.scriptedJob], parent=self.parentLayout, protected=True)

        feedback = '>>>> creating ScriptJob >>>> Start Time =
{time}'.format(time=self.startTime)
        mc.text(self.feedbackText, edit=True, label=feedback)

    def scriptedJob(self):
        ## this job executes checkTimer() on a certain condition or event
        totalElapsedTime = mc.timerX(startTime = self.startTime)
        # print "totalElapsedTime is ", totalElapsedTime
        currentTime = mc.timerX()
        ## set self.lastExecutionTime to the time the scriptJob was
initiated
        if self.lastExecutionTime == 0:
            self.lastExecutionTime = currentTime
            elapsedTime = totalElapsedTime
            feedback = '>>>> SCRIPT HASNT BEEN EXECUTED YET >>>> '
            mc.text(self.feedbackText, edit=True, label=feedback)
        else:
            elapsedTime = currentTime - self.lastExecutionTime
```

```
        self.checkTimer(elapsedTime)

        ######### create camera job
        if self.createCameras == True:
            #### nb. mesh number is the function return value
            self.scriptedCameraJob()

    def checkTimer(self, elapsedTime):
        interval = mc.intSliderGrp(self.secondsSlider, query=True,
value=True)
        function = mc.textField(self.procedureTF, query=True, text=True)

        if elapsedTime >= interval:
            feedback = '>>>> Performing function >>>>
{function}'.format(function=function)
            mc.text(self.feedbackText, edit=True, label=feedback)

            ## this is where it evalutaes the function
            self.executeFunction()

            currentTime = mc.timerX()
            self.lastExecutionTime = currentTime
            elapsedTime = 0
        else:
            feedback =  '>>>> NOT TIME yet >>>> elapsedTime =
{elapsedTime}'.format(elapsedTime=elapsedTime)
            mc.text(self.feedbackText, edit=True, label=feedback)

    def executeFunction(self, *args):
        function = mc.textField(self.procedureTF, query=True, text=True)
        functionReturn = eval(function)

        ######################### in the case of dupeModel
functionReturnValue is currentMeshNumber
        # if functionReturn != None:
        self.functionReturnValue = functionReturn

        print 'function return value is ', functionReturn
        return functionReturn


    def createCamera(self, *args):
        ############################### manual create camera button
##########################
        ### NB. I need to store original selection
        currentSelection = mc.ls(selection=True)

        meshName = self.functionReturnValue[0]
        meshNumber = self.functionReturnValue[1]

        cameraPosList = self.getCameraTransforms()[0]
        cameraRotList = self.getCameraTransforms()[1]
        #get the distance between two points
        distanceTravelled =
distanceBetweenPoints(self.previousCamPosList, cameraPosList)
        rotationAmount = amountRotated(self.previousCamRotList,
cameraRotList)
```

```
        cameraNumber = self.newCameraNumber + 1
        cameraName = 'cam{number}'.format(number=cameraNumber)

        # #### ########### create newCam and position according to
cameraPosList and cameraRotList
        newCamTransform = mc.camera(name=cameraName,
displayFilmGate=True, displayResolution=True, \
            position=(cameraPosList[0], cameraPosList[1],
cameraPosList[2]),\
            rotation=(cameraRotList[0], cameraRotList[1],
cameraRotList[2]))[0]

        ### make group for cameras if it doesnt exist
        if mc.objExists('cameraGROUP') == False:
            cameraGroup = mc.group(empty=True, name='cameraGROUP')
            mc.setAttr('{cameraGroup}.v'.format(cameraGroup=cameraGroup),
0)
        else:
            cameraGroup = 'cameraGROUP'

        ### put camera in group
        mc.parent(newCamTransform, cameraGroup)

        self.setCameraAttrs(newCamTransform, cameraNumber, meshName,
meshNumber)

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

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

        ############ resume currentSelection
        if len(currentSelection) != 0:
            mc.select(currentSelection)

        print distanceTravelled
        print rotationAmount

        feedback = '>>>> distanceTravelled is {distanceTravelled} >>>>
rotationAmount is {rotationAmount}'\
            .format(distanceTravelled=distanceTravelled,
rotationAmount=rotationAmount)
        mc.text(self.cameraFeedbackText, edit=True, label=feedback)

        # return distanceTravelled

    def scriptedCameraJob(self, *args):
        '''
        19_06_14
        I'm not sure why I have doubled up and not used createCamera()
here ??
        '''
                ### NB. I need to stor original selection
        currentSelection = mc.ls(selection=True)
```

```
        meshName = self.functionReturnValue[0]
        meshNumber = self.functionReturnValue[1]

        print 'self.functionReturnValue is ', self.functionReturnValue
        # print 'meshNumber is ', meshNumber

        distanceThreshold = mc.intSliderGrp(self.distanceThresholdSlider,
query=True, value=True)
        rotationThreshold = mc.intSliderGrp(self.rotationThresholdSlider,
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)
        rotationAmount = amountRotated(self.previousCamRotList,
cameraRotList)

        ### if the distanceTravelled is larger than distance threshold
make a camera
        if distanceTravelled > distanceThreshold or rotationAmount >
rotationThreshold:
            # print ' distance excedes threshold'
            cameraNumber = self.newCameraNumber + 1
            cameraName = 'cam{number}'.format(number=cameraNumber)

            #### ########### !!! create new camera
            newCamTransform = mc.camera(name=cameraName,
displayFilmGate=True, displayResolution=True, \
            position=(cameraPosList[0], cameraPosList[1],
cameraPosList[2]),\
            rotation=(cameraRotList[0], cameraRotList[1],
cameraRotList[2]))[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')

mc.setAttr('{cameraGroup}.v'.format(cameraGroup=cameraGroup), 0)
            else:
                cameraGroup = 'cameraGROUP'
            ### put camera in group
            mc.parent(newCamTransform, cameraGroup)

            self.setCameraAttrs(newCamTransform, cameraNumber, meshName,
meshNumber)

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


        ### #### last thing is to store as "previous camera position
list"
```

```
        self.previousCamPosList = cameraPosList
        self.previousCamRotList = cameraRotList

        ############ resume currentSelection
        if len(currentSelection) != 0:
            mc.select(currentSelection)

        print distanceTravelled
        print rotationAmount
        feedback = '>>>> distanceTravelled is {distanceTravelled} >>>>
rotationAmount is {rotationAmount}'\
            .format(distanceTravelled=distanceTravelled,
rotationAmount=rotationAmount)
        mc.text(self.cameraFeedbackText, edit=True, label=feedback)

    def setCameraAttrs(self, newCamTransform, cameraNumber, meshName,
meshNumber):
        ############## dont know how meshNumber can get to here and be
NoneType... but anyway
        if meshNumber != None:
            meshNumber = int(meshNumber)

            #
mc.setAttr("{newCamTransform}.displayFilmGate".format(newCamTransform=new
CamTransform), 1)
            #
mc.setAttr("{newCamTransform}.displayResolution".format(newCamTransform=n
ewCamTransform), 1)
            #
mc.setAttr("{newCamTransform}.v".format(newCamTransform=newCamTransform),
1)

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

            ### add attr to store associated mesh number as INTEGER
            mc.addAttr(newCamTransform, attributeType="byte",
longName='meshNumber', keyable=True, storable=True)
            ## set the correspondingMesh attr to the correspondingMesh

mc.setAttr('{newCamTransform}.meshNumber'.format(newCamTransform=newCamTr
ansform), meshNumber)

        if meshName != None:
                        ### add string attr for mesh name
            mc.addAttr(newCamTransform, dataType="string",
longName='meshName', storable=True)
            ### set the meshName attr

mc.setAttr('{newCamTransform}.meshName'.format(newCamTransform=newCamTran
sform), meshName, type="string")
            print
mc.getAttr('{newCamTransform}.meshName'.format(newCamTransform=newCamTran
sform))

    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
```