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

'''
24_06_14
- add ability to project ramps
'''
clm1 = 250
clm2 =100


class ColourTweakCntrls(object):

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


        self.buildUI()

    def buildUI(self):
        separation = 15

        row1 = mc.rowLayout(numberOfColumns=2, columnWidth2=[(clm1),
(clm2)], parent=self.parentLayout)
        mc.text('Add Tweak attr to selected SurfaceShader :  ',
align='right', parent=row1, w=clm1)
        # separator = mc.separator(style ="none", w=300, h=separation,
parent=self.parentLayout)
        addAttrsButton = mc.button(label = 'add tweak attrs',
command=self.addTweakAttrs, parent=row1, w=clm2)

    def addTweakAttrs(self, *args):
        # print "adding controls for cycling through the colour tweaks"
        #get the SScell shaders
        shaderList = mc.ls(selection=True, materials=True)

        for shader in shaderList:

            ## get the ramp node attached to color
            baseRampList =
mc.listConnections("{shader}.outColor".format(shader=shader),
type='ramp')
            print 'shader is ', shader
            print 'baseRampList is ', baseRampList

            if baseRampList != None:
                baseRamp = baseRampList[0]
                #### get the numEntries
                numEntries =
mc.getAttr('{baseRamp}.colorEntryList'.format(baseRamp=baseRamp),
size=True)
                print baseRamp, numEntries

                #### make sure that there are more than 1 colour entry
                if numEntries == 1:
                    print 'only one colorEntryList in ramp; no colour
tweaks'
                    return numEntries
```

```
                else:

                    #### add tweak attr to the shader
                    attrExistence = mc.attributeQuery('colourTweak',
node=shader, exists=True)
                    if attrExistence == False:
                        print 'adding
',"{shader}.colourTweak".format(shader=shader)
                        mc.addAttr(shader, shortName='colourTweak',
min=0, max=(numEntries-1), storable=True, \
                            attributeType='double', keyable=True)

                    #### work on each colorEntry in turn
                    colorIndex = 1
                    for entry in range(numEntries-1):
                        print 'entry is ', entry
                        print ' colorIndex is ', colorIndex
                        print 'numEntries is ', numEntries
                        print 'shader is ', shader

                        self.createNodeNetwork(numEntries, colorIndex,
shader)
                        colorIndex += 1


    def createNodeNetwork(self, numEntries, colorIndex, shader):
        cellIndexList =  shader.split('_')
        cellIndex = cellIndexList[1]
        print 'cellIndex is ', cellIndex

        ## NB the following is complicated by the fact of the reverse
node
        numOfTweaks = numEntries - 1
        tweakRange = 1 - (0.001 * numOfTweaks)
        tweakNum = colorIndex
        endPos = 0.001 * tweakNum
        startPos = endPos + tweakRange

        # print 'numEntries is ', numEntries
        # print 'shader is ', shader
        # print 'colorIndex is ', colorIndex
        # print 'cellIndex is ', cellIndex

        # print 'numOfTweaks is ', numOfTweaks
        # print 'tweakRange is ', tweakRange
        # print 'tweakNum is ', tweakNum
        # print 'endPos is ', endPos, 'startPos is ', startPos

        setRangeMin = (numOfTweaks * 0.001) - (0.001 * tweakNum)
        setRangeMax = setRangeMin + tweakRange

        clampName = 'clampTweak{i}_SS{cellIndex}'.format(i=colorIndex,
cellIndex=cellIndex)
        clampMin = colorIndex - 1
        clampMax = colorIndex
```

```
        setRangeName =
'setRangeTweak{i}_SS{cellIndex}'.format(i=colorIndex,
cellIndex=cellIndex)

        ### create clamp node and set values
        clampNode = mc.shadingNode('clamp', asUtility=True, name =
clampName)
        mc.setAttr("{clampNode}.minR".format(clampNode=clampNode),
clampMin)
        mc.setAttr("{clampNode}.maxR".format(clampNode=clampNode),
clampMax)

        # setRangeNode = mc.shadingNode('setRange', asUtility=True, name
= setRangeName)
        #
mc.setAttr("{setRangeNode}.minX".format(setRangeNode=setRangeNode),
setRangeMin)
        #
mc.setAttr("{setRangeNode}.maxX".format(setRangeNode=setRangeNode),
setRangeMax)

        setRangeNode = mc.shadingNode('setRange', asUtility=True, name =
setRangeName)

mc.setAttr("{setRangeNode}.minX".format(setRangeNode=setRangeNode),
setRangeMin)

mc.setAttr("{setRangeNode}.maxX".format(setRangeNode=setRangeNode),
setRangeMax)
        #### old min and old max

mc.setAttr("{setRangeNode}.oldMinX".format(setRangeNode=setRangeNode),
clampMin)

mc.setAttr("{setRangeNode}.oldMaxX".format(setRangeNode=setRangeNode),
clampMax)




        ##create reverse node
        reverseNode = mc.shadingNode('reverse', asUtility=True)

        ###### connect Nodes
        attrExistence = mc.attributeQuery('colourTweak', node=shader,
exists=True)
        if attrExistence == True:

            # shader into clamp
            mc.connectAttr('{shader}.colourTweak'.format(shader=shader),\
                '{clampNode}.input.inputR'.format(clampNode=clampNode),
force=True)


mc.connectAttr('{clampNode}.output.outputR'.format(clampNode=clampNode),
\
```

```
'{setRangeNode}.value.valueX'.format(setRangeNode=setRangeNode),
force=True)


mc.connectAttr('{setRangeNode}.outValue.outValueX'.format(setRangeNode=se
tRangeNode),\

'{reverseNode}.input.inputX'.format(reverseNode=reverseNode), force=True)

            #reverse to pos

mc.connectAttr('{reverseNode}.output.outputX'.format(reverseNode=reverseN
ode),\

'rampCell_{cellIndex}.colorEntryList[{colorIndex}].position'\
                .format(cellIndex=cellIndex, colorIndex=colorIndex),
force=True)


class TextureCntrls(object):

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


        self.buildUI()

    def buildUI(self):
        separation = 15

        row1 = mc.rowLayout(numberOfColumns=2, columnWidth2=[(clm1),
(clm2)]  , parent=self.parentLayout)
        mc.text('project texture of selected SS :  ', align='right',
parent=row1, w=clm1)
        # separator = mc.separator(style ="none", w=300, h=separation,
parent=self.parentLayout)
        addAttrsButton = mc.button(label = 'add projector',
command=self.addProjector, parent=row1, w=clm2)

        row2 = mc.rowLayout(numberOfColumns=2, columnWidth2=[(clm1),
(clm2)]  , parent=self.parentLayout)
        mc.text('set ramp type on selected SS :  ', align='right',
parent=row2, w=clm1)
        # separator = mc.separator(style ="none", w=300, h=separation,
parent=self.parentLayout)
        addAttrsButton = mc.button(label = 'set ramp type',
command=self.setRampType, parent=row2, w=clm2)

    def addProjector(self, *args):
        ### get the shader
        shaderList = mc.ls(selection=True, materials=True)
        ### get the base ramp
        for shader in shaderList:
            ## get the baseRamp node attached to color
            baseRampList =
mc.listConnections("{shader}.outColor".format(shader=shader),
type='ramp')
```

```
            print 'shader is ', shader
            print 'baseRampList is ', baseRampList
            if baseRampList != None:
                baseRamp = baseRampList[0]

                ### make 2d placement for ramp
                placement2D = mc.shadingNode('place2dTexture',
asUtility=True)
                #hook them up

mc.connectAttr('{placement2D}.outUV'.format(placement2D=placement2D), \
                    '{baseRamp}.uvCoord'.format(baseRamp=baseRamp),
force=True)

mc.connectAttr('{placement2D}.outUV'.format(placement2D=placement2D),
                 '{baseRamp}.uvFilterSize'.format(baseRamp=baseRamp),
force=True)

                #make a projector
                projector = mc.shadingNode('projection', asUtility=True)
                ## make a placement for projector
                placement3D = mc.shadingNode('place3dTexture',
asUtility=True)
                #hook them up

mc.connectAttr('{placement3D}.worldInverseMatrix[0]'.format(placement3D=p
lacement3D), \

'{projector}.placementMatrix'.format(projector=projector), force=True)

                ##connect ramp to projector

mc.connectAttr('{baseRamp}.outColor'.format(baseRamp=baseRamp), \
                 '{projector}.image'.format(projector=projector),
force=True)

                ### connect image to SS outColour

mc.connectAttr('{projector}.image'.format(projector=projector), \
                    '{shader}.outColor'.format(shader=shader),
force=True)


            else:
                print 'cant find base ramp'


    def setRampType(self, *args):
        shaderList = mc.ls(selection=True, materials=True)
        ### get the base ramp
        for shader in shaderList:
            ## get the baseRamp node attached to color
            projectorList =
mc.listConnections("{shader}.outColor".format(shader=shader),
type='projection')
            print 'shader is ', shader
            print 'projectorList is ', projectorList
```

```
            if projectorList != None:
                projector = projectorList[0]

                baseRampList =
mc.listConnections("{projector}.image".format(projector=projector),
type='ramp')
                if baseRampList != None:
                    baseRamp = baseRampList[0]

                    ###set ramp type to; Vramp = 0,  tartan = 8,
circular=4

mc.setAttr("{baseRamp}.type".format(baseRamp=baseRamp), 0)
        ###set ramp interp to linear
                    #
mc.setAttr("{baseRamp}.interpolation".format(baseRamp=baseRamp), 1)
        ###set ramp interp to exponential down
                    #
mc.setAttr("{baseRamp}.interpolation".format(baseRamp=baseRamp), 3)


class SetCameraBGcolour(object):
    # sets camera bg to ramp texture if it exists
    def __init__(self, parentLayout):
        self.parentLayout = parentLayout
        self.buildUI()

    def buildUI(self):
        separation = 15

        row1 = mc.rowLayout(numberOfColumns=2, columnWidth2=[(clm1),
(clm2)], parent=self.parentLayout)
        mc.text('Set camera BG to viewport BG colour :  ', align='right',
parent=row1, w=clm1)
        # separator = mc.separator(style ="none", w=300, h=separation,
parent=self.parentLayout)
        setCamBGButton = mc.button(label = 'set cam BG',
command=self.setcamBG, parent=row1, w=clm2)

    def setcamBG(self, *args):
        ###check that ramp exists make it if it doesnt
        viewportBGramp = 'viewportBGcolour'

        if mc.objExists(viewportBGramp) == False:
            viewportBGramp = mc.shadingNode("ramp",asTexture=True, name =
'viewportBGcolour')
                    #### if theres more than two entries delete the third
            numEntries =
mc.getAttr('{baseRamp}.colorEntryList'.format(baseRamp=baseRamp),
size=True)
            if numEntries >= 3:

mc.removeMultiInstance('viewportBGcolour.colorEntryList[2]')

        #### get the ramp colour (I guess its just an average for now
        colour = mc.getAttr('viewportBGcolour.outColor')[0]
        print 'setting cam BG colour to', colour
```

```
        #get list of cameras
        cameras = mc.ls(type = 'camera', long = True)
        #iterate over cameras
        for camera in cameras:
            ########## with image planes
            # imagePlane = mc.createNode('imagePlane')
            # mc.setAttr
("{imagePlane}.type".format(imagePlane=imagePlane), 1)
            # mc.connectAttr ( 'BGcolour.outColor',
'{imagePlane}.sourceTexture'.format(imagePlane=imagePlane, force=True))
            #
mc.connectAttr('{imagePlane}.message'.format(imagePlane=imagePlane),'{cam
era}.imagePlane[0]'.format(camera = camera))

            ###### using cam BG colour....NB is a flat colour
            mc.setAttr("{camera}.backgroundColor".format(camera=camera),
colour[0], colour[1], colour[2], type='double3')
```