```
import maya.cmds as mc

'''
from SHELF button ONE -->
NB. this one works most of the time (as long as ordered selection works
in the list command)
    ---------NOT SURE will now try to not retain offset on constrain to
rotation control
select 4 vertices in order Front1, Front2, Back1, Back2,
    the script will create a locator,
    position it at the location of each vertex,
    then point on poly constrain it to the vertex
'''

def createPolySelectedJoints(bind=True):
    '''
    select two joints to make a poly face between them
    it create locators and then deletes them
    '''
    if mc.objExists("rayGeo") == True:
        print 'rayGeo group already exists'
    else:
        mc.group(empty=True, name = 'rayGeo')

    parentJointList = mc.ls(selection = True, dependencyNodes=True)
    joint1 = parentJointList[0]
    joint4 = parentJointList[1]

    joint2 = mc.listRelatives(joint1, children=True, type='joint')
    joint3 = mc.listRelatives(joint4, children=True, type='joint')
    orderedJoints = [joint1,joint2,joint3,joint4]
    print orderedJoints

    #create locator and parent to joint1
    loc1 = mc.spaceLocator()[0]
    mc.parent(loc1, joint1, relative=True)
    mc.parent(loc1, world=True)
    #create locator and parent to joint2
    loc2 = mc.spaceLocator()[0]
    mc.parent(loc2, joint2, relative=True)
    mc.parent(loc2, world=True)
    #create locator and parent to joint3
    loc3 = mc.spaceLocator()[0]
    mc.parent(loc3, joint3, relative=True)
    mc.parent(loc3, world=True)
    #create locator and parent to joint3
    loc4 = mc.spaceLocator()[0]
    mc.parent(loc4, joint4, relative=True)
    mc.parent(loc4, world=True)

    loc1Pos = mc.getAttr('{loc1}.translate'.format(loc1=loc1))[0]
    loc2Pos = mc.getAttr('{loc2}.translate'.format(loc2=loc2))[0]
    loc3Pos = mc.getAttr('{loc3}.translate'.format(loc3=loc3))[0]
    loc4Pos = mc.getAttr('{loc4}.translate'.format(loc4=loc4))[0]
    vertPosList = [loc1Pos, loc2Pos, loc3Pos, loc4Pos]
    # create teh poly face
    mesh = mc.polyCreateFacet(point = vertPosList, name = 'ray')
    #delete locators
```

```
    mc.delete(loc1, loc2, loc3, loc4)
    #bind mesh to joints
    if bind == True:
        print 'yes'
        mc.select(joint1, joint3, mesh)
        mc.SmoothBindSkin()
    #parent to rayGeo group
    mc.parent(meshRay1, 'rayGeo')

def distanceBetween(joint1, joint2, name):
    name = name
    dis = mc.shadingNode('distanceBetween', asUtility=True, name=name)
    mc.connectAttr((joint1 +'.worldMatrix'), (dis + '.inMatrix1'))
    mc.connectAttr((joint2 +'.worldMatrix'), (dis + '.inMatrix2'))
    mc.connectAttr((joint1 +'.rotatePivotTranslate'), (dis + '.point1'))
    mc.connectAttr((joint2 +'.rotatePivotTranslate'), (dis + '.point2'))
    return dis

def createJoints(direction = 1):
    xDis= direction * 3
    zDis=0
    firstJointPos = (0,4,0)
    firstJoint = mc.joint(position=firstJointPos)
    secondJointPos = (xDis,0,zDis)
    secondJoint = mc.joint(position=secondJointPos)
    #parent under JOINTS grp
    mc.parent(firstJoint, 'JOINTS')
    #edit joint orientation
    mc.joint(firstJoint, edit=True,  orientJoint ='xyz')
    #create locator below first joint
    locatorGround = (mc.spaceLocator(position=(0,0,0)))[0]
    #parent under groundLOCATORS grp
    mc.parent(locatorGround, 'groundLOCATORS')

    #distance X (firstJoint to locator)
    disX = mc.shadingNode('distanceBetween', asUtility=True,
name='distanceX')

mc.connectAttr(('{firstJoint}.worldMatrix'.format(firstJoint=firstJoint))
, ('{disX}.inMatrix1'.format(disX=disX)))

mc.connectAttr(('{locatorGround}.worldMatrix'.format(locatorGround=locato
rGround)), ('{disX}.inMatrix2'.format(disX=disX)))

mc.connectAttr(('{firstJoint}.rotatePivotTranslate'.format(firstJoint=fir
stJoint)), ('{disX}.point1'.format(disX=disX)))

mc.connectAttr(('{locatorGround}.rotatePivotTranslate'.format(locatorGrou
nd=locatorGround)), ('{disX}.point2'.format(disX=disX)))
    #get distance Y (locator to secondJoint)
    disY = mc.shadingNode('distanceBetween', asUtility=True,
name='distanceY')

mc.connectAttr(('{secondJoint}.worldMatrix'.format(secondJoint=secondJoin
t)), ('{disY}.inMatrix1'.format(disY=disY)))

mc.connectAttr(('{locatorGround}.worldMatrix'.format(locatorGround=locato
rGround)), ('{disY}.inMatrix2'.format(disY=disY)))
```

```
mc.connectAttr(('{secondJoint}.rotatePivotTranslate'.format(secondJoint=s
econdJoint)), ('{disY}.point1'.format(disY=disY)))

mc.connectAttr(('{locatorGround}.rotatePivotTranslate'.format(locatorGrou
nd=locatorGround)), ('{disY}.point2'.format(disY=disY)))

    #create multDivide for square of X and Y
    squareMultDiv = mc.shadingNode('multiplyDivide', asUtility=True,
name='square')
    mc.connectAttr('{disX}.distance'.format(disX=disX),
'{squareMultDiv}.input1.input1X'.format(squareMultDiv=squareMultDiv))
    mc.connectAttr('{disX}.distance'.format(disX=disX),
'{squareMultDiv}.input2.input2X'.format(squareMultDiv=squareMultDiv))
    mc.connectAttr('{disY}.distance'.format(disY=disY),
'{squareMultDiv}.input1.input1Y'.format(squareMultDiv=squareMultDiv))
    mc.connectAttr('{disY}.distance'.format(disY=disY),
'{squareMultDiv}.input2.input2Y'.format(squareMultDiv=squareMultDiv))
    # add the square of X and the square of Y
    addDoubLin = mc.shadingNode('addDoubleLinear', asUtility=True,
name='add')

mc.connectAttr('{squareMultDiv}.outputX'.format(squareMultDiv=squareMultD
iv), '{addDoubLin}.input1'.format(addDoubLin=addDoubLin))

mc.connectAttr('{squareMultDiv}.outputY'.format(squareMultDiv=squareMultD
iv), '{addDoubLin}.input2'.format(addDoubLin=addDoubLin))
    #get the square root of Xsquared + Ysquared
    squrRt = mc.shadingNode('multiplyDivide', asUtility=True,
name='squrRoot')
    mc.connectAttr('{addDoubLin}.output'.format(addDoubLin=addDoubLin),
'{squrRt}.input1.input1X'.format(squrRt=squrRt))
    mc.setAttr('{squrRt}.operation'.format(squrRt=squrRt), 3)
    mc.setAttr('{squrRt}.input2X'.format(squrRt=squrRt), 0.5)
    #divide the square root by 5
    divideBy5 = mc.shadingNode('multiplyDivide', asUtility=True,
name='divideBy5')
    mc.connectAttr('{squrRt}.outputX'.format(squrRt=squrRt),
'{divideBy5}.input1.input1X'.format(divideBy5=divideBy5))
    mc.setAttr('{divideBy5}.operation'.format(divideBy5=divideBy5), 2)
    mc.setAttr('{divideBy5}.input2X'.format(divideBy5=divideBy5), 5)
    # how to get rid of cycleCheck warning?????

    #connect result to firstJoint scaleX
    mc.connectAttr('{divideBy5}.outputX'.format(divideBy5=divideBy5),
'{firstJoint}.scaleX'.format(firstJoint=firstJoint))
    #constrain locator to joint position X and Z
    mc.pointConstraint( firstJoint ,locatorGround, maintainOffset=True,
skip='y' )

    return firstJoint, secondJoint

def createShadowCntrl(direction=1):
    direction = direction
    shadowCntrl = mc.polyCone(height=7, subdivisionsX=5, radius=2.5,
name='lightSource', constructionHistory=False)[0]
```

```
mc.move(3.5,'{shadowCntrl}.scalePivot'.format(shadowCntrl=shadowCntrl),
moveY=True)

mc.move(3.5,'{shadowCntrl}.rotatePivot'.format(shadowCntrl=shadowCntrl),
moveY=True)
    mc.xform(shadowCntrl, translation = (0,25,0))

mc.setAttr("{shadowCntrl}.overrideEnabled".format(shadowCntrl=shadowCntrl
), 1)

mc.setAttr("{shadowCntrl}.overrideShading".format(shadowCntrl=shadowCntrl
), 0)

mc.setAttr("{shadowCntrl}.overrideColor".format(shadowCntrl=shadowCntrl),
17)
    rotation = direction * 30.33
    mc.setAttr("{shadowCntrl}.rotateZ".format(shadowCntrl=shadowCntrl),
rotation)
    return shadowCntrl

def createPoly(joint1, joint2, bind=True):

    joint1 = joint1
    joint4 = joint2
    joint2 = mc.listRelatives(joint1, children=True, type='joint')
    joint3 = mc.listRelatives(joint4, children=True, type='joint')
    orderedJoints = [joint1,joint2,joint3,joint4]
    print orderedJoints

    #create locator and parent to joint1
    loc1 = mc.spaceLocator()[0]
    mc.parent(loc1, joint1, relative=True)
    mc.parent(loc1, world=True)
    #create locator and parent to joint2
    loc2 = mc.spaceLocator()[0]
    mc.parent(loc2, joint2, relative=True)
    mc.parent(loc2, world=True)
    #create locator and parent to joint3
    loc3 = mc.spaceLocator()[0]
    mc.parent(loc3, joint3, relative=True)
    mc.parent(loc3, world=True)
    #create locator and parent to joint3
    loc4 = mc.spaceLocator()[0]
    mc.parent(loc4, joint4, relative=True)
    mc.parent(loc4, world=True)

    loc1Pos = mc.getAttr('{loc1}.translate'.format(loc1=loc1))[0]
    loc2Pos = mc.getAttr('{loc2}.translate'.format(loc2=loc2))[0]
    loc3Pos = mc.getAttr('{loc3}.translate'.format(loc3=loc3))[0]
    loc4Pos = mc.getAttr('{loc4}.translate'.format(loc4=loc4))[0]
    vertPosList = [loc1Pos, loc2Pos, loc3Pos, loc4Pos]
    # create the poly face
    mesh = mc.polyCreateFacet(point = vertPosList, name = 'ray')
    #delete locators
    mc.delete(loc1, loc2, loc3, loc4)
    #bind mesh to joints
    if bind == True:
```

```
        print 'yes'
        mc.select(joint1, joint4, mesh)
        mc.SmoothBindSkin()

    return mesh

def createLocators(vertList, direction=1):

    #create Light controller
    if mc.objExists("lightSource") == True:
        cntrl = "lightSource"
        print 'lightSource already exists'
    else:
        cntrl = createShadowCntrl(direction=direction)
    #create group for model locators if it doesnt exist
    if mc.objExists("modelLOCATORS") == True:
        print 'modelLOCATORS group already exists'
    else:
        mc.group(empty=True, name = 'modelLOCATORS')
        mc.setAttr("modelLOCATORS.visibility", 0)
    #create group for joints if it doesnt exist
    if mc.objExists("JOINTS") == True:
        print 'JOINTS group already exists'
    else:
        mc.group(empty=True, name = 'JOINTS')
        # mc.setAttr("JOINTS.visibility", 0)
    #create group for ground locators if it doesnt exist
    if mc.objExists("groundLOCATORS") == True:
        print 'groundLOCATORS group already exists'
    else:
        mc.group(empty=True, name = 'groundLOCATORS')
        mc.setAttr("groundLOCATORS.visibility", 0)

    #create group for shadow and ray Geo if it doesnt exist
    if mc.objExists("groundShadowGeo") == True:
        print 'groundShadowGeo group already exists'
    else:
        mc.group(empty=True, name = 'groundShadowGeo')

    if mc.objExists("rayGeo") == True:
        print 'rayGeo group already exists'
    else:
        mc.group(empty=True, name = 'rayGeo')

    #get list of selected vertexes from main procedure
    vertList = vertList

    #make lists for joint locations
    secondJntPosList = []
    firstJntList = []
    #put a locator on each vertex and constrain it
    for vert in vertList:
        vertexPosition = mc.xform(vert,query=True, worldSpace=True,
translation=True)
        locator = mc.spaceLocator()
        mc.xform(locator, worldSpace = True, translation =
vertexPosition)
        #create variables for target and object; just because...
```

```
        target = str(vert)
        obj = str(locator[0])
        #select the target then the object and constrain the locator
        mc.select(target)
        mc.select(obj, tgl=True)
        mc.PointOnPolyConstraint()
        #parent the locator under LOCATOR grp
        mc.parent(locator, 'modelLOCATORS')

        # create joints
        joints = createJoints(direction=direction)
        firstJoint = joints[0]
        secondJoint = joints[1]
        #constrain first joint to locator rivetted to mesh
        mc.pointConstraint(locator ,firstJoint)

        #constrain  firstJoint rotation to controller
        mc.orientConstraint(cntrl,  firstJoint, maintainOffset=True,
weight=1)

        # get second joint location and add to list
        joint2Loc = mc.spaceLocator()[0]
        mc.parent(joint2Loc, secondJoint, relative=True)
        mc.parent(joint2Loc, world=True)
        secondJointPos =
(mc.getAttr('{joint2Loc}.translate'.format(joint2Loc=joint2Loc)))[0]
        print 'secondJointPos is
{secondJointPos}'.format(secondJointPos=secondJointPos)
        secondJntPosList.append(secondJointPos)
        #delete the locator
        mc.delete(joint2Loc)

        # add first Joint to the list
        firstJntList.append(firstJoint)

    print 'firstJntList is
{firstJntList}'.format(firstJntList=firstJntList)
    joint1, joint2, joint3, joint4 = firstJntList
    print 'secondJntPosList is
{secondJntPosList}'.format(secondJntPosList=secondJntPosList)
    pos1, pos2, pos3, pos4 = secondJntPosList
    orderedList = [pos1, pos2, pos4, pos3]
    ### create and bind ground shadow poly
    meshGround = mc.polyCreateFacet(point = orderedList, name =
'shadow')[0]
    mc.select(joint1, joint2, joint3, joint4, meshGround)
    mc.SmoothBindSkin()
    #create side geo
    meshRay1 = createPoly(joint1, joint2)[0]
    meshRay2 = createPoly(joint3, joint4)[0]
    #parent geo to groups
    mc.parent(meshGround, 'groundShadowGeo')
    mc.parent(meshRay1, 'rayGeo')
    mc.parent(meshRay2, 'rayGeo')


def shadowPlayMain(direction=1):
    #track selection order
```

```
    # mc.selectPref(trackSelectionOrder=True)
    #get list of selected vertexes
    vertList = mc.ls(flatten = True, orderedSelection = True)
    print vertList
    createLocators(vertList=vertList, direction = direction)

shadowPlayMain(direction=-1)


########################################## from shelf button2
import maya.cmds as mc

'''
from SHELF button TWO -->
select vertices and the script will create a locator,
position it at the location of each vertex,
then point on poly constrain it to the vertex
'''

def distanceBetween(joint1, joint2, name):
    name = name
    dis = mc.shadingNode('distanceBetween', asUtility=True, name=name)
    mc.connectAttr((joint1 +'.worldMatrix'), (dis + '.inMatrix1'))
    mc.connectAttr((joint2 +'.worldMatrix'), (dis + '.inMatrix2'))
    mc.connectAttr((joint1 +'.rotatePivotTranslate'), (dis + '.point1'))
    mc.connectAttr((joint2 +'.rotatePivotTranslate'), (dis + '.point2'))
    return dis

def createJoints(direction = 1):
    xDis=3 * direction
    zDis=0
    firstJointPos = (0,4,0)
    firstJoint = mc.joint(position=firstJointPos)
    secondJointPos = (xDis,0,zDis)
    secondJoint = mc.joint(position=secondJointPos)
    #parent under JOINTS grp
    mc.parent(firstJoint, 'JOINTS')
    #edit joint orientation
    mc.joint(firstJoint, edit=True,  orientJoint ='xyz')
    #create locator below first joint
    locatorGround = (mc.spaceLocator(position=(0,0,0)))[0]
    #parent under groundLOCATORS grp
    mc.parent(locatorGround, 'groundLOCATORS')

    #distance X (firstJoint to locator)
    disX = mc.shadingNode('distanceBetween', asUtility=True,
name='distanceX')

mc.connectAttr(('{firstJoint}.worldMatrix'.format(firstJoint=firstJoint))
, ('{disX}.inMatrix1'.format(disX=disX)))

mc.connectAttr(('{locatorGround}.worldMatrix'.format(locatorGround=locato
rGround)), ('{disX}.inMatrix2'.format(disX=disX)))

mc.connectAttr(('{firstJoint}.rotatePivotTranslate'.format(firstJoint=fir
stJoint)), ('{disX}.point1'.format(disX=disX)))
```

```
mc.connectAttr(('{locatorGround}.rotatePivotTranslate'.format(locatorGrou
nd=locatorGround)), ('{disX}.point2'.format(disX=disX)))
    #get distance Y (locator to secondJoint)
    disY = mc.shadingNode('distanceBetween', asUtility=True,
name='distanceY')

mc.connectAttr(('{secondJoint}.worldMatrix'.format(secondJoint=secondJoin
t)), ('{disY}.inMatrix1'.format(disY=disY)))

mc.connectAttr(('{locatorGround}.worldMatrix'.format(locatorGround=locato
rGround)), ('{disY}.inMatrix2'.format(disY=disY)))

mc.connectAttr(('{secondJoint}.rotatePivotTranslate'.format(secondJoint=s
econdJoint)), ('{disY}.point1'.format(disY=disY)))

mc.connectAttr(('{locatorGround}.rotatePivotTranslate'.format(locatorGrou
nd=locatorGround)), ('{disY}.point2'.format(disY=disY)))

    #create multDivide for square of X and Y
    squareMultDiv = mc.shadingNode('multiplyDivide', asUtility=True,
name='square')
    mc.connectAttr('{disX}.distance'.format(disX=disX),
'{squareMultDiv}.input1.input1X'.format(squareMultDiv=squareMultDiv))
    mc.connectAttr('{disX}.distance'.format(disX=disX),
'{squareMultDiv}.input2.input2X'.format(squareMultDiv=squareMultDiv))
    mc.connectAttr('{disY}.distance'.format(disY=disY),
'{squareMultDiv}.input1.input1Y'.format(squareMultDiv=squareMultDiv))
    mc.connectAttr('{disY}.distance'.format(disY=disY),
'{squareMultDiv}.input2.input2Y'.format(squareMultDiv=squareMultDiv))
    # add the square of X and the square of Y
    addDoubLin = mc.shadingNode('addDoubleLinear', asUtility=True,
name='add')

mc.connectAttr('{squareMultDiv}.outputX'.format(squareMultDiv=squareMultD
iv), '{addDoubLin}.input1'.format(addDoubLin=addDoubLin))

mc.connectAttr('{squareMultDiv}.outputY'.format(squareMultDiv=squareMultD
iv), '{addDoubLin}.input2'.format(addDoubLin=addDoubLin))
    #get the square root of Xsquared + Ysquared
    squrRt = mc.shadingNode('multiplyDivide', asUtility=True,
name='squrRoot')
    mc.connectAttr('{addDoubLin}.output'.format(addDoubLin=addDoubLin),
'{squrRt}.input1.input1X'.format(squrRt=squrRt))
    mc.setAttr('{squrRt}.operation'.format(squrRt=squrRt), 3)
    mc.setAttr('{squrRt}.input2X'.format(squrRt=squrRt), 0.5)
    #divide the square root by 5
    divideBy5 = mc.shadingNode('multiplyDivide', asUtility=True,
name='divideBy5')
    mc.connectAttr('{squrRt}.outputX'.format(squrRt=squrRt),
'{divideBy5}.input1.input1X'.format(divideBy5=divideBy5))
    mc.setAttr('{divideBy5}.operation'.format(divideBy5=divideBy5), 2)
    mc.setAttr('{divideBy5}.input2X'.format(divideBy5=divideBy5), 5)
    # how to get rid of cycleCheck warning?????

    #connect result to firstJoint scaleX
    mc.connectAttr('{divideBy5}.outputX'.format(divideBy5=divideBy5),
'{firstJoint}.scaleX'.format(firstJoint=firstJoint))
```

```
    #constrain locator to joint position X and Z
    mc.pointConstraint( firstJoint ,locatorGround, maintainOffset=True,
skip='y' )

    return firstJoint, secondJoint

def createShadowCntrl(direction=1):
    shadowCntrl = mc.polyCone(height=7, subdivisionsX=5, radius=2.5,
name='lightSource', constructionHistory=False)[0]

mc.move(3.5,'{shadowCntrl}.scalePivot'.format(shadowCntrl=shadowCntrl),
moveY=True)

mc.move(3.5,'{shadowCntrl}.rotatePivot'.format(shadowCntrl=shadowCntrl),
moveY=True)
    mc.xform(shadowCntrl, translation = (0,25,0))

mc.setAttr("{shadowCntrl}.overrideEnabled".format(shadowCntrl=shadowCntrl
), 1)

mc.setAttr("{shadowCntrl}.overrideShading".format(shadowCntrl=shadowCntrl
), 0)

mc.setAttr("{shadowCntrl}.overrideColor".format(shadowCntrl=shadowCntrl),
17)
    rotation = 30.33 * direction
    mc.setAttr("{shadowCntrl}.rotateZ".format(shadowCntrl=shadowCntrl),
rotation)
    return shadowCntrl

def createLocators(direction=1):
    #get list of selected vertexes
    vertList = mc.ls(selection = True, flatten = True)
    #create Light controller
    if mc.objExists("lightSource") == True:
        cntrl = "lightSource"
        print 'lightSource already exists'
    else:
        cntrl = createShadowCntrl(direction=direction)

    #create group for model locators if it doesnt exist
    if mc.objExists("modelLOCATORS") == True:
        print 'modelLOCATORS group already exists'
    else:
        mc.group(empty=True, name = 'modelLOCATORS')
        mc.setAttr("modelLOCATORS.visibility", 0)
    #create group for joints if it doesnt exist
    if mc.objExists("JOINTS") == True:
        print 'JOINTS group already exists'
    else:
        mc.group(empty=True, name = 'JOINTS')
        # mc.setAttr("JOINTS.visibility", 0)
    #create group for ground locators if it doesnt exist
    if mc.objExists("groundLOCATORS") == True:
        print 'groundLOCATORS group already exists'
    else:
        mc.group(empty=True, name = 'groundLOCATORS')
        mc.setAttr("groundLOCATORS.visibility", 0)
```

```
    #put a locator on each vertex and constrain it
    for vert in vertList:
        vertexPosition = mc.xform(vert,query=True, worldSpace=True,
translation=True)
        locator = mc.spaceLocator()
        mc.xform(locator, worldSpace = True, translation =
vertexPosition)
        #create variables for target and object; just because...
        target = str(vert)
        obj = str(locator[0])
        #select the target then the object and constrain the locator
        mc.select(target)
        mc.select(obj, tgl=True)
        mc.PointOnPolyConstraint()
        #parent the locator under LOCATOR grp
        mc.parent(locator, 'modelLOCATORS')

        # create joints
        joints = createJoints(direction=direction)
        firstJoint = joints[0]
        #constrain first joint to locator rivetted to mesh
        mc.pointConstraint(locator ,firstJoint)

        #contrain  firstJoint rotation to controller
        mc.orientConstraint(cntrl,  firstJoint, maintainOffset=True,
weight=1)


createLocators(direction=-1)


############################################################
import maya.cmds as mc

'''
from button THREE -->
create and bind ground shadow poly from anyNumber of selected joints
'''

def groundPolySelectedJoints(bind=True):
    '''
    select joints to make a poly face between them
    '''
    if mc.objExists("groundShadowGeo") == True:
        print 'groundShadowGeo group already exists'
    else:
        mc.group(empty=True, name = 'groundShadowGeo')

    jointList = mc.ls(orderedSelection = True)
    if len(jointList) == 4:
        joint1, joint2, joint3, joint4 = jointList
    else:
        joint1, joint2, joint3 = jointList

    parentJointList = []
    for joint in jointList:
        parentJoint = mc.listRelatives(joint, parent=True, type='joint')
```

```
        parentJointList.append(parentJoint)
    if len(parentJointList) == 4:
        parentJoint1, parentJoint2, parentJoint3, parentJoint4 =
parentJointList
    else:
        parentJoint1, parentJoint2, parentJoint3 = parentJointList

    #create locator and parent to joint1
    loc1 = mc.spaceLocator()[0]
    mc.parent(loc1, joint1, relative=True)
    mc.parent(loc1, world=True)
    #create locator and parent to joint2
    loc2 = mc.spaceLocator()[0]
    mc.parent(loc2, joint2, relative=True)
    mc.parent(loc2, world=True)
    #create locator and parent to joint3
    loc3 = mc.spaceLocator()[0]
    mc.parent(loc3, joint3, relative=True)
    mc.parent(loc3, world=True)
    if len(jointList) == 4:
        #create locator and parent to joint4
        loc4 = mc.spaceLocator()[0]
        mc.parent(loc4, joint4, relative=True)
        mc.parent(loc4, world=True)

    loc1Pos = mc.getAttr('{loc1}.translate'.format(loc1=loc1))[0]
    loc2Pos = mc.getAttr('{loc2}.translate'.format(loc2=loc2))[0]
    loc3Pos = mc.getAttr('{loc3}.translate'.format(loc3=loc3))[0]
    if len(parentJointList) == 4:
        loc4Pos = mc.getAttr('{loc4}.translate'.format(loc4=loc4))[0]
        vertPosList = [loc1Pos, loc2Pos, loc3Pos, loc4Pos]
    else:
        vertPosList = [loc1Pos, loc2Pos, loc3Pos]
    # create the poly face
    mesh = mc.polyCreateFacet(point = vertPosList, name =
'shadowExtraBit')[0]
    #delete locators
    mc.delete(loc1, loc2, loc3)
    if len(jointList) == 4:
        mc.delete(loc4)
    #bind mesh to joints
    if bind == True:
        if len(jointList) == 4:
            mc.select(parentJoint1, parentJoint2, parentJoint3,
parentJoint4, mesh)
        else:
            mc.select(parentJoint1, parentJoint2, parentJoint3, mesh)
        mc.SmoothBindSkin()
    #parent to rayGeo group
    mc.parent(mesh, 'groundShadowGeo')

groundPolySelectedJoints()


#################################################################
import maya.cmds as mc

'''
```

```
from button FOUR -->
'''

def createPolySelectedJoints(bind=True):
    '''
    select two joints to make a poly face between them
    it create locators and then deletes them
    NB. make sure that bind method is set to closest distance
    '''
    parentJointList = mc.ls(selection = True, dependencyNodes=True)
    print parentJointList

    if mc.objExists("rayGeo") == True:
        print 'rayGeo group already exists'
    else:
        mc.group(empty=True, name = 'rayGeo')


    joint1 = parentJointList[0]
    joint4 = parentJointList[1]
    # print joint1
    # print joint4

    joint2 = mc.listRelatives(joint1, children=True, type='joint')
    joint3 = mc.listRelatives(joint4, children=True, type='joint')
    orderedJoints = [joint1,joint2,joint3,joint4]
    print orderedJoints

    #create locator and parent to joint1
    loc1 = mc.spaceLocator()[0]
    mc.parent(loc1, joint1, relative=True)
    mc.parent(loc1, world=True)
    #create locator and parent to joint2
    loc2 = mc.spaceLocator()[0]
    mc.parent(loc2, joint2, relative=True)
    mc.parent(loc2, world=True)
    #create locator and parent to joint3
    loc3 = mc.spaceLocator()[0]
    mc.parent(loc3, joint3, relative=True)
    mc.parent(loc3, world=True)
    #create locator and parent to joint3
    loc4 = mc.spaceLocator()[0]
    mc.parent(loc4, joint4, relative=True)
    mc.parent(loc4, world=True)

    loc1Pos = mc.getAttr('{loc1}.translate'.format(loc1=loc1))[0]
    loc2Pos = mc.getAttr('{loc2}.translate'.format(loc2=loc2))[0]
    loc3Pos = mc.getAttr('{loc3}.translate'.format(loc3=loc3))[0]
    loc4Pos = mc.getAttr('{loc4}.translate'.format(loc4=loc4))[0]
    vertPosList = [loc1Pos, loc2Pos, loc3Pos, loc4Pos]
    # create teh poly face
    mesh = mc.polyCreateFacet(point = vertPosList, name = 'ray')
    #delete locators
    mc.delete(loc1, loc2, loc3, loc4)
    #bind mesh to joints
    if bind == True:
        print 'yes'
        mc.select(joint1, joint4, mesh)
```

```
        mc.SmoothBindSkin()
    #parent to rayGeo group
    mc.parent(mesh, 'rayGeo')

    #turn off cycle check if its on
    cycleStatus = mc.cycleCheck(query = True, evaluation= True)
    if cycleStatus == True:
        mc.cycleCheck(e= False)

createPolySelectedJoints()


# ######################################################################
'''
from cycle check button 5
'''
mc.cycleCheck(e= False)
```