**단계 1**

import maya.cmds as cmds

def gui():

    if cmds.window('win', q=1, ex=1):

        cmds.deleteUI('win', window=True)

    cmds.window('win', title='Generator')

    cmds.columnLayout(adjustableColumn=True)

    cmds.button(label='hello')

    cmds.showWindow('win')

gui()

→→→→→

1. 커스텀 윈도우 창을 만드는 스크립트 (=UNDO 같은 UI)
2. columnLayout(): 세로로 아이템을 배치하는 컬럼 레이아웃 함수
3. adjustableColumn=True: 레이아웃이 창의 너비로 조절되도록 하는 함수 (=윈도우 사이즈에 맞춤)
4. hello 버튼이 있는 윈도우 창을 띄움

**단계 2**

import maya.cmds as cmds

def gui():

    if cmds.window('win', q=1, ex=1):

        cmds.deleteUI('win', window=True)

    cmds.window('win', title='Generator')

    cmds.columnLayout(adjustableColumn=True)

    cmds.button(label='hello')

    cmds.floatSliderGrp ('test', label='Test\_F', field=1, minValue=0.1, maxValue=10.0, fieldMinValue=0.1, value=10)

    cmds.showWindow('win')

gui()

→→→→→

1. hello버튼과 Test\_F 슬라이더가 같이 있는 창을 띄움

아직은 그 둘에 아무런 함수도 들어가 있지 않아서 기능 없음.

**단계 3**

import maya.cmds as cmds

from functools import partial

def gui():

    if cmds.window('win', q=1, ex=1):

        cmds.deleteUI('win', window=True)

    cmds.window('win', title='Generator')

    cmds.columnLayout(adjustableColumn=True)

    def generate\_poly(cube\_size, \*arguments):

        cmds.polyCube(width=1.0, height=1.0, depth=1.0)[0]

    cmds.button(label='hello', command=partial(generate\_poly, ))

    cmds.floatSliderGrp('test', label='Test\_F', field=1, minValue=0.0, maxValue=10.0, fieldMinValue=0.0, value=10)

    cmds.showWindow('win')

gui()

→→→→→

1. from functools import partial <- 이걸 왜 임포트 하느냐. \*을 쓰기 위해서 ->\*args = \*arguments : 가변 인자

⑴ add(1, 2)

⑵ add(1, 4, 5)

⑶ add(1, 5, 10, 13, 19)

이렇게 인자가 몇개 들어올지 알 수 없는 인자를 가변인자라고 함. 변할 수 있는 인자라는 뜻

**단계 4**

씬 최적화, 혹은 정리 하는 인터페이스??를... 언젠가 만들자.. 라는 생각에서 시작

++추후에 하이퍼 쉐이드에서 del unused node 기능도 추가하자

(del\_LockNode, deleteUnknown, keyDelete, Sub\_Lock, animOptiA, Min\_Find을 포함)

import maya.cmds as cmds

from functools import partial

def Scene\_Cleaner\_Gui():

    if cmds.window('win', q=1, ex=1):

        cmds.deleteUI('win', window=True)

    cmds.window('win', title='Scene Cleaner')

    cmds.columnLayout(adjustableColumn=True)

    def del\_LockNode(Delete\_Bool = True):

        sel = cmds.ls(' TurtleDefaultBakeLayer')

        lock =cmds.lockNode(sel , l =0)

        cmds.delete(sel)

    cmds.button(label='del\_LockNode', command=partial(del\_LockNode, ))

    cmds.showWindow('win')

Scene\_Cleaner\_Gui()

1. del\_LockNode 추가

import maya.cmds as cmds

from functools import partial

def Scene\_Cleaner\_Gui():

    if cmds.window('win', q=1, ex=1):

        cmds.deleteUI('win', window=True)

    cmds.window('win', title='Scene Cleaner')

    cmds.columnLayout(adjustableColumn=True)

#1.del\_LockNode Function add

    def del\_LockNode(Delete\_Bool = True):

        sel = cmds.ls(' TurtleDefaultBakeLayer')

        lock =cmds.lockNode(sel , l =0)

        cmds.delete(sel)

 #1.del\_LockNode button add

    cmds.button(label='Del LockNode', command=partial(del\_LockNode, ))

#2.key\_Delete Function add

    def key\_Delete(Delete\_Bool = True, \*arguments):

        All =[]

        lst\_PO = cmds.ls(type = 'animCurveTL')

        lst\_RO = cmds.ls(type = 'animCurveTA')

        lst\_ETF = cmds.ls(type = 'animCurveTU')

        All.extend(lst\_PO)

        All.extend(lst\_RO)

        All.extend(lst\_ETF)

        if Delete\_Bool == True:

            cmds.delete(All)

        else:

            pass

 #2.key\_Delete button add

    cmds.button(label='key Delete', command=partial(key\_Delete, True))

    cmds.showWindow('win')

Scene\_Cleaner\_Gui()

2. keyDelete 추가

import maya.cmds as cmds

from functools import partial

def Scene\_Cleaner\_Gui():

    if cmds.window('win', q=1, ex=1):

        cmds.deleteUI('win', window=True)

    cmds.window('win', title='Scene Cleaner')

    cmds.columnLayout(adjustableColumn=True)

#1.del\_LockNode Function add

    def del\_LockNode(Delete\_Bool = True):

        sel = cmds.ls(' TurtleDefaultBakeLayer')

        lock =cmds.lockNode(sel , l =0)

        cmds.delete(sel)

 #1.del\_LockNode button add

    cmds.button(label='Del LockNode', command=partial(del\_LockNode, ))

#2.key\_Delete Function add

    def key\_Delete(Delete\_Bool = True, \*arguments):

        All =[]

        lst\_PO = cmds.ls(type = 'animCurveTL')

        lst\_RO = cmds.ls(type = 'animCurveTA')

        lst\_ETF = cmds.ls(type = 'animCurveTU')

        All.extend(lst\_PO)

        All.extend(lst\_RO)

        All.extend(lst\_ETF)

        if Delete\_Bool == True:

            cmds.delete(All)

        else:

            pass

 #2.key\_Delete button add

    cmds.button(label='key Delete', command=partial(key\_Delete, True))

#3.deleteUnknown Function add

    def deleteUnknown(Delete\_Bool = True, \*arguments):

        lst = []

        sel01 = cmds.ls('vray\*')

        sel02 = cmds.ls(type="unknown")

        sel03 = cmds.ls( ' TurtleDefaultBakeLayer')

        lst.extend(sel01)

        lst.extend(sel02)

        lst.extend(sel03)

        lst\_unlock = [ cmds.lockNode(x, l =0) for x in lst]

        cmds.delete(lst)

#3.key\_Delete button add

    cmds.button(label='delete Unknown', command=partial(deleteUnknown, True))

    cmds.showWindow('win')

Scene\_Cleaner\_Gui()

3. deleteUnknown 추가

import maya.cmds as cmds

from functools import partial

import maya.mel as mel

def Scene\_Cleaner\_Gui():

    if cmds.window('win', q=1, ex=1):

        cmds.deleteUI('win', window=True)

    cmds.window('win', title='Scene Cleaner')

    cmds.columnLayout(adjustableColumn=True)

#\_\_\_\_\_del\_LockNode Function add\_\_\_\_\_

    def del\_LockNode(Delete\_Bool = True):

        sel = cmds.ls(' TurtleDefaultBakeLayer')

        lock =cmds.lockNode(sel , l =0)

        cmds.delete(sel)

#\_\_\_\_\_del\_LockNode button add\_\_\_\_\_

    cmds.button(label='TurtleDefaultBakeLayer Delete', command=partial(del\_LockNode, ))

#\_\_\_\_\_key\_Delete Function add\_\_\_\_\_

    def key\_Delete(Delete\_Bool = True, \*arguments):

        All =[]

        lst\_PO = cmds.ls(type = 'animCurveTL')

        lst\_RO = cmds.ls(type = 'animCurveTA')

        lst\_ETF = cmds.ls(type = 'animCurveTU')

        All.extend(lst\_PO)

        All.extend(lst\_RO)

        All.extend(lst\_ETF)

        if Delete\_Bool == True:

            cmds.delete(All)

        else:

            pass

#\_\_\_\_\_key\_Delete button add\_\_\_\_\_

    cmds.button(label='key Delete', command=partial(key\_Delete, True))

#\_\_\_\_\_deleteUnknown Function add\_\_\_\_\_

    def deleteUnknown(Delete\_Bool = True, \*arguments):

        lst = []

        sel01 = cmds.ls('vray\*')

        sel02 = cmds.ls(type="unknown")

        sel03 = cmds.ls( ' TurtleDefaultBakeLayer')

        lst.extend(sel01)

        lst.extend(sel02)

        lst.extend(sel03)

        lst\_unlock = [ cmds.lockNode(x, l =0) for x in lst]

        cmds.delete(lst)

#\_\_\_\_\_deleteUnknown button add\_\_\_\_\_

    cmds.button(label='delete Unknown', command=partial(deleteUnknown, True))

#\_\_\_\_\_Sub\_Lock Function add\_\_\_\_\_

    def Sub\_lock(Check):

        pre\_lst = cmds.ls(type='RedshiftMeshParameters')

        lst = [x for x in pre\_lst if '\_sub' in x]

        if Check:

            lock = True

        else:

            lock = False

        for obj in lst:

            cmds.lockNode(obj, lock=lock)

    def deleteUnknown():

        lst = []

        sel01 = cmds.ls('vray\*')

        sel02 = cmds.ls(type="unknown")

        sel03 = cmds.ls( ' TurtleDefaultBakeLayer')

        lst.extend(sel01)

        lst.extend(sel02)

        lst.extend(sel03)

        lst\_unlock = [ cmds.lockNode(x, l =0) for x in lst]

        cmds.delete(lst)

    class SDPwin:

        def create(self):

            cmds.select(cl=1)

            selA = cmds.ls('\_Ctrl', '\_Con', '\*\_C', type='transform')

            resultA = []

            resultB = []

            resultC = []

            resultE = []

            for x in range(len(selA)):

                trs = ['translateX', 'translateY', 'translateZ', 'rotateX', 'rotateY', 'rotateZ']

                for y in range(len(trs)):

                    try:

                        if cmds.getAttr('%s.%s' % (selA[x], trs[y]), lock=1):

                            resultC.append(selA[x] + '.' + trs[y])

                            continue

                        ga = cmds.getAttr('%s.%s' % (selA[x], trs[y]))

                        if (y <= 5):

                            if (ga != 0.0):

                                if 'e' in str(ga):

                                    cmds.setAttr('%s.%s' % (selA[x], trs[y]), 0)

                                    resultA.append(selA[x])

                                else:

                                    resultB.append(selA[x] + '.' + trs[y])

                                    pass

                        else:

                            if (ga != 1.0):

                                if 'e' in str(ga):

                                    cmds.setAttr('%s.%s' % (selA[x], trs[y]), 0)

                                    resultA.append(selA[x])

                                else:

                                    resultB.append(selA[x] + '.' + trs[y])

                                    pass

                    except:

                        pass

            print('---0 list---   ' + ', '.join(resultA))

            print('---0 or Move list---   ' + ', '.join(resultB))

            print('---lock list---   ' + ', '.join(resultC))

#\_\_\_\_\_Sub\_Lock button add\_\_\_\_\_

    def button\_callback(\*args):

        if Sub\_lock(True):

            SDPwin().create()

    cmds.columnLayout(adjustableColumn=True)

    cmds.button(label='Sub lock', command=button\_callback)

#\_\_\_\_\_Display Window\_\_\_\_\_

    cmds.showWindow('win')

Scene\_Cleaner\_Gui()

4. Sub\_Lock 추가 (클래스는 함수 밖으로 빼줘야 한다.)

import maya.cmds as cmds

from functools import partial

import maya.mel as mel

class SDPwin02:

    def create\_MinFind(self, Delete\_Bool=True, \*arguments):

        cmds.select(cl=1)

        selA = cmds.ls( '\_Ctrl', '\_Con', '\*\_C', type='transform' )

        resultA = []

        resultC = []

        resultE = []

        print '\n\n    print -> Ctrl List\n\n'

        for x in range(len(selA)):

            #print selA[x]

            trs = [ 'translateX', 'translateY', 'translateZ', 'rotateX', 'rotateY', 'rotateZ', 'scaleX', 'scaleY', 'scaleZ' ]

            for y in range(len(trs)):

                ga = cmds.getAttr('%s.%s'%(selA[x],trs[y]))

                if(y<=5):

                    if(ga!=0.0):

                        #if(ga>=0.1):

                        print '%s.%s    %s'%(selA[x],trs[y],ga)

                        resultA.append(selA[x])

                else:

                    if(ga!=1.0):

                        #if(ga>=1.1):

                        print '%s.%s    %s'%(selA[x],trs[y],ga)

                        resultA.append(selA[x])

        for x in range(len(selA)):

            trs = [ 'rotatePivotX', 'rotatePivotY', 'rotatePivotZ', 'scalePivotX', 'scalePivotY', 'scalePivotZ' ]

            for y in range(len(trs)):

                ga = cmds.getAttr('%s.%s'%(selA[x],trs[y]))

                if(ga!=0.0):

                    #if(ga>=0.1):

                    print '%s.%s    %s'%(selA[x],trs[y],ga)

                    resultC.append(selA[x])

            selA = cmds.ls( 'Mod\_Grp' )

        if selA:

            selB = cmds.listRelatives( selA, ad=1, type='transform')

            for x in range(len(selB)):

            #print selB[x]

                trs = [ 'scaleX', 'scaleY', 'scaleZ' ]

                for y in range(len(trs)):

                    ga = cmds.getAttr('%s.%s'%(selB[x],trs[y]))

                    if(ga!=1.0):

                        print '%s.%s    %s'%(selB[x],trs[y],ga)

                        resultE.append(selB[x])

        else:

            print '\n\n    Check!    "Mod\_Grp"\n\n'

        resultBB = set(resultA)

        resultDD = set(resultC)

        resultFF = set(resultE)

        resultB = list(resultBB)

        resultD = list(resultDD)

        resultF = list(resultFF)

        corName = 'correction\_SET'

        pivName = 'pivot\_SET'

        modName = 'modScale\_SET'

        selCP = [ corName, pivName, modName ]

        selBD = [ resultB, resultD, resultF ]

        for x in range(len(selCP)):

            #print len(selBD[x])

            print selBD[x]

            selE = cmds.ls( selCP[x] )

            if (len(selE)!=0):# type

                cmds.delete( selCP[x] )

            if(len(selBD[x])!=0):# list

                cmds.sets( selBD[x], n=selCP[x] )

def Scene\_Cleaner\_Gui():

    if cmds.window('win', q=1, ex=1):

        cmds.deleteUI('win', window=True)

    cmds.window('win', title='Scene Cleaner')

    cmds.columnLayout(adjustableColumn=True)

#\_\_\_\_\_del\_LockNode Function add\_\_\_\_\_

    def del\_LockNode(Delete\_Bool = True):

        sel = cmds.ls(' TurtleDefaultBakeLayer')

        lock =cmds.lockNode(sel , l =0)

        cmds.delete(sel)

#\_\_\_\_\_del\_LockNode button add\_\_\_\_\_

    cmds.button(label='TurtleDefaultBakeLayer Delete', command=partial(del\_LockNode, ))

#\_\_\_\_\_key\_Delete Function add\_\_\_\_\_

    def key\_Delete(Delete\_Bool = True, \*arguments):

        All =[]

        lst\_PO = cmds.ls(type = 'animCurveTL')

        lst\_RO = cmds.ls(type = 'animCurveTA')

        lst\_ETF = cmds.ls(type = 'animCurveTU')

        All.extend(lst\_PO)

        All.extend(lst\_RO)

        All.extend(lst\_ETF)

        if Delete\_Bool == True:

            cmds.delete(All)

        else:

            pass

#\_\_\_\_\_key\_Delete button add\_\_\_\_\_

    cmds.button(label='key Delete', command=partial(key\_Delete, True))

#\_\_\_\_\_deleteUnknown Function add\_\_\_\_\_

    def deleteUnknown(Delete\_Bool = True, \*arguments):

        lst = []

        sel01 = cmds.ls('vray\*')

        sel02 = cmds.ls(type="unknown")

        sel03 = cmds.ls( ' TurtleDefaultBakeLayer')

        lst.extend(sel01)

        lst.extend(sel02)

        lst.extend(sel03)

        lst\_unlock = [ cmds.lockNode(x, l =0) for x in lst]

        cmds.delete(lst)

#\_\_\_\_\_deleteUnknown button add\_\_\_\_\_

    cmds.button(label='delete Unknown', command=partial(deleteUnknown, True))

#\_\_\_\_\_Sub\_Lock Function add\_\_\_\_\_

    def Sub\_lock(Check):

        pre\_lst = cmds.ls(type='RedshiftMeshParameters')

        lst = [x for x in pre\_lst if '\_sub' in x]

        if Check:

            lock = True

        else:

            lock = False

        for obj in lst:

            cmds.lockNode(obj, lock=lock)

    def deleteUnknown():

        lst = []

        sel01 = cmds.ls('vray\*')

        sel02 = cmds.ls(type="unknown")

        sel03 = cmds.ls( ' TurtleDefaultBakeLayer')

        lst.extend(sel01)

        lst.extend(sel02)

        lst.extend(sel03)

        lst\_unlock = [ cmds.lockNode(x, l =0) for x in lst]

        cmds.delete(lst)

    class SDPwin:

        def create(self):

            cmds.select(cl=1)

            selA = cmds.ls('\_Ctrl', '\_Con', '\*\_C', type='transform')

            resultA = []

            resultB = []

            resultC = []

            resultE = []

            for x in range(len(selA)):

                trs = ['translateX', 'translateY', 'translateZ', 'rotateX', 'rotateY', 'rotateZ']

                for y in range(len(trs)):

                    try:

                        if cmds.getAttr('%s.%s' % (selA[x], trs[y]), lock=1):

                            resultC.append(selA[x] + '.' + trs[y])

                            continue

                        ga = cmds.getAttr('%s.%s' % (selA[x], trs[y]))

                        if (y <= 5):

                            if (ga != 0.0):

                                if 'e' in str(ga):

                                    cmds.setAttr('%s.%s' % (selA[x], trs[y]), 0)

                                    resultA.append(selA[x])

                                else:

                                    resultB.append(selA[x] + '.' + trs[y])

                                    pass

                        else:

                            if (ga != 1.0):

                                if 'e' in str(ga):

                                    cmds.setAttr('%s.%s' % (selA[x], trs[y]), 0)

                                    resultA.append(selA[x])

                                else:

                                    resultB.append(selA[x] + '.' + trs[y])

                                    pass

                    except:

                        pass

            print('---0 list---   ' + ', '.join(resultA))

            print('---0 or Move list---   ' + ', '.join(resultB))

            print('---lock list---   ' + ', '.join(resultC))

#\_\_\_\_\_Sub\_Lock button add\_\_\_\_\_

    def button\_callback(\*args):

        if Sub\_lock(True):

            SDPwin().create()

    cmds.columnLayout(adjustableColumn=True)

    cmds.button(label='Sub lock', command=button\_callback)

#\_\_\_\_\_Min\_Find Function add\_\_\_\_\_

#\_\_\_\_\_Min\_Find button add\_\_\_\_\_

    cmds.button(label='Min Find', command=partial(SDPwin02().create\_MinFind, True))

#\_\_\_\_\_Display Window\_\_\_\_\_

    cmds.showWindow('win')

Scene\_Cleaner\_Gui()

5. Min\_Find 추가 (마찬가지로 클래스는 밖으로 빼줘야 함. 그리고 이름은 당연히 다르게 설정.)

import maya.cmds as cmds

from functools import partial

import maya.mel as mel

#\_\_\_\_\_0\_Del Function add\_\_\_\_\_

#khs

class SDPwin\_Del:

    def create\_MinDel(self, Delete\_Bool=True, \*arguments):

        cmds.select(cl=1)

        selA = cmds.ls('\_Ctrl', '\_Con', '\*\_C', type='transform')

        resultA = []

        resultB = []

        resultC = []

        resultE = []

        for x in range(len(selA)):

            trs = ['translateX', 'translateY', 'translateZ', 'rotateX', 'rotateY', 'rotateZ']

            for y in range(len(trs)):

                if cmds.getAttr('%s.%s' % (selA[x], trs[y]), lock=1):

                    resultC.append(selA[x] + '.' + trs[y])

                    continue

                ga = cmds.getAttr('%s.%s' % (selA[x], trs[y]))

                if (y <= 5):

                    if (ga != 0.0):

                        if 'e' in str(ga):

                            cmds.setAttr('%s.%s' % (selA[x], trs[y]), 0)

                            resultA.append(selA[x])

                        else:

                            resultB.append(selA[x] + '.' + trs[y])

                            pass

                else:

                    if (ga != 1.0):

                        if 'e' in str(ga):

                            cmds.setAttr('%s.%s' % (selA[x], trs[y]), 0)

                            resultA.append(selA[x])

                        else:

                            resultB.append(selA[x] + '.' + trs[y])

                            pass

        print('---0 list---   ' + ', '.join(resultA))

        print('---0 or Move list---   ' + ', '.join(resultB))

        print('---lock list---   ' + ', '.join(resultC))

#\_\_\_\_\_Min\_Find Function add\_\_\_\_\_

class SDPwin\_Find:

    def create\_MinFind(self, Delete\_Bool=True, \*arguments):

        cmds.select(cl=1)

        selA = cmds.ls( '\_Ctrl', '\_Con', '\*\_C', type='transform' )

        resultA = []

        resultC = []

        resultE = []

        print '\n\n    print -> Ctrl List\n\n'

        for x in range(len(selA)):

            #print selA[x]

            trs = [ 'translateX', 'translateY', 'translateZ', 'rotateX', 'rotateY', 'rotateZ', 'scaleX', 'scaleY', 'scaleZ' ]

            for y in range(len(trs)):

                ga = cmds.getAttr('%s.%s'%(selA[x],trs[y]))

                if(y<=5):

                    if(ga!=0.0):

                        #if(ga>=0.1):

                        print '%s.%s    %s'%(selA[x],trs[y],ga)

                        resultA.append(selA[x])

                else:

                    if(ga!=1.0):

                        #if(ga>=1.1):

                        print '%s.%s    %s'%(selA[x],trs[y],ga)

                        resultA.append(selA[x])

        for x in range(len(selA)):

            trs = [ 'rotatePivotX', 'rotatePivotY', 'rotatePivotZ', 'scalePivotX', 'scalePivotY', 'scalePivotZ' ]

            for y in range(len(trs)):

                ga = cmds.getAttr('%s.%s'%(selA[x],trs[y]))

                if(ga!=0.0):

                    #if(ga>=0.1):

                    print '%s.%s    %s'%(selA[x],trs[y],ga)

                    resultC.append(selA[x])

            selA = cmds.ls( 'Mod\_Grp' )

        if selA:

            selB = cmds.listRelatives( selA, ad=1, type='transform')

            for x in range(len(selB)):

            #print selB[x]

                trs = [ 'scaleX', 'scaleY', 'scaleZ' ]

                for y in range(len(trs)):

                    ga = cmds.getAttr('%s.%s'%(selB[x],trs[y]))

                    if(ga!=1.0):

                        print '%s.%s    %s'%(selB[x],trs[y],ga)

                        resultE.append(selB[x])

        else:

            print '\n\n    Check!    "Mod\_Grp"\n\n'

        resultBB = set(resultA)

        resultDD = set(resultC)

        resultFF = set(resultE)

        resultB = list(resultBB)

        resultD = list(resultDD)

        resultF = list(resultFF)

        corName = 'correction\_SET'

        pivName = 'pivot\_SET'

        modName = 'modScale\_SET'

        selCP = [ corName, pivName, modName ]

        selBD = [ resultB, resultD, resultF ]

        for x in range(len(selCP)):

            #print len(selBD[x])

            print selBD[x]

            selE = cmds.ls( selCP[x] )

            if (len(selE)!=0):# type

                cmds.delete( selCP[x] )

            if(len(selBD[x])!=0):# list

                cmds.sets( selBD[x], n=selCP[x] )

#\_\_\_\_\_create Gui\_\_\_\_\_

def Scene\_Cleaner\_Gui():

    if cmds.window('win', q=1, ex=1):

        cmds.deleteUI('win', window=True)

    cmds.window('win', title='Scene Cleaner')

    cmds.columnLayout(adjustableColumn=True)

#\_\_\_\_\_del\_LockNode Function add\_\_\_\_\_

    def del\_LockNode(Delete\_Bool = True):

        sel = cmds.ls(' TurtleDefaultBakeLayer')

        lock =cmds.lockNode(sel , l =0)

        cmds.delete(sel)

#\_\_\_\_\_del\_LockNode button add\_\_\_\_\_

    cmds.button(label='Turtle Delete', command=partial(del\_LockNode, ))

#\_\_\_\_\_key\_Delete Function add\_\_\_\_\_

    def key\_Delete(Delete\_Bool = True, \*arguments):

        All =[]

        lst\_PO = cmds.ls(type = 'animCurveTL')

        lst\_RO = cmds.ls(type = 'animCurveTA')

        lst\_ETF = cmds.ls(type = 'animCurveTU')

        All.extend(lst\_PO)

        All.extend(lst\_RO)

        All.extend(lst\_ETF)

        if Delete\_Bool == True:

            cmds.delete(All)

        else:

            pass

#\_\_\_\_\_key\_Delete button add\_\_\_\_\_

    cmds.button(label='key Delete', command=partial(key\_Delete, True))

#\_\_\_\_\_deleteUnknown Function add\_\_\_\_\_

    def deleteUnknown(Delete\_Bool = True, \*arguments):

        lst = []

        sel01 = cmds.ls('vray\*')

        sel02 = cmds.ls(type="unknown")

        sel03 = cmds.ls( ' TurtleDefaultBakeLayer')

        lst.extend(sel01)

        lst.extend(sel02)

        lst.extend(sel03)

        lst\_unlock = [ cmds.lockNode(x, l =0) for x in lst]

        cmds.delete(lst)

#\_\_\_\_\_deleteUnknown button add\_\_\_\_\_

    cmds.button(label='delete Unknown', command=partial(deleteUnknown, True))

#\_\_\_\_\_Sub\_Lock Function add\_\_\_\_\_

    def Sub\_lock(Check):

        pre\_lst = cmds.ls(type='RedshiftMeshParameters')

        lst = [x for x in pre\_lst if '\_sub' in x]

        if Check:

            lock = True

        else:

            lock = False

        for obj in lst:

            cmds.lockNode(obj, lock=lock)

    def deleteUnknown():

        lst = []

        sel01 = cmds.ls('vray\*')

        sel02 = cmds.ls(type="unknown")

        sel03 = cmds.ls( ' TurtleDefaultBakeLayer')

        lst.extend(sel01)

        lst.extend(sel02)

        lst.extend(sel03)

        lst\_unlock = [ cmds.lockNode(x, l =0) for x in lst]

        cmds.delete(lst)

    class SDPwin:

        def create(self):

            cmds.select(cl=1)

            selA = cmds.ls('\_Ctrl', '\_Con', '\*\_C', type='transform')

            resultA = []

            resultB = []

            resultC = []

            resultE = []

            for x in range(len(selA)):

                trs = ['translateX', 'translateY', 'translateZ', 'rotateX', 'rotateY', 'rotateZ']

                for y in range(len(trs)):

                    try:

                        if cmds.getAttr('%s.%s' % (selA[x], trs[y]), lock=1):

                            resultC.append(selA[x] + '.' + trs[y])

                            continue

                        ga = cmds.getAttr('%s.%s' % (selA[x], trs[y]))

                        if (y <= 5):

                            if (ga != 0.0):

                                if 'e' in str(ga):

                                    cmds.setAttr('%s.%s' % (selA[x], trs[y]), 0)

                                    resultA.append(selA[x])

                                else:

                                    resultB.append(selA[x] + '.' + trs[y])

                                    pass

                        else:

                            if (ga != 1.0):

                                if 'e' in str(ga):

                                    cmds.setAttr('%s.%s' % (selA[x], trs[y]), 0)

                                    resultA.append(selA[x])

                                else:

                                    resultB.append(selA[x] + '.' + trs[y])

                                    pass

                    except:

                        pass

            print('---0 list---   ' + ', '.join(resultA))

            print('---0 or Move list---   ' + ', '.join(resultB))

            print('---lock list---   ' + ', '.join(resultC))

#\_\_\_\_\_Sub\_Lock button add\_\_\_\_\_

    def button\_callback(\*args):

        if Sub\_lock(True):

            SDPwin().create()

    cmds.columnLayout(adjustableColumn=True)

    cmds.button(label='Sub lock', command=button\_callback)

#\_\_\_\_\_Min\_Find button add\_\_\_\_\_

    cmds.button(label='Min Find', command=partial(SDPwin\_Find().create\_MinFind, True))

#\_\_\_\_\_Min Del button add\_\_\_\_\_

    cmds.button(label='Min Del', command=partial(SDPwin\_Del().create\_MinDel, True))

#\_\_\_\_\_Display Window\_\_\_\_\_

    cmds.showWindow('win')

Scene\_Cleaner\_Gui()

6. animOptiAsms는 제외. 스크립트 다듬기

**단계 5**

인터페이스 정렬 방법에 대해서...

#\_\_\_\_\_design interface\_\_\_\_\_

    cmds.window('win', e=1, widthHeight= 400, 140))

1. 지정된 사이즈로 호출 되게 하기. 262열에 위 구문 추가. 실행 시 사이즈 200\*200으로 지정.

cmds.columnLayout(adjustableColumn=True)

2. 116열에 이 구문이 있기 때문에 지정된 크기로 호출 되더라도 후에 크기 조절 가능.

++계속 진행