

Python-Cubit[®] Enhancement Scripts: 16.18

Neal Grieb/Trevor Hensley
thensle@sandia.gov
(505) 844-3304
Sandia National Laboratories
Org. 9376

August 19, 2024

1 DOE funding statement



”Sandia National Laboratories is a multimission laboratory managed and operated by National Technology Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA0003525.”

2 Introduction

The Python-CUBIT[®] enhancement code base is intended to be used as an extension to already existing CUBIT[®] functionality. It provides the user with a number of functionalities that are either currently outside the realm of the python functions which CUBIT[®] supplies internally (such as vector math), or that are comprised of commonly used combinations of already existing python functionalities (such as removing a full round from a slot cut).

The foreseen style of use for many of these scripts is to utilize volume names and geometric data such as surface area, surface type, etc. as a way to filter out geometries, and provide a powerful id-less method. These filters combined with a number of already existing python functionalities such as the set() operator and zip() function can be used to operate on many geometries at a single time without a need for the user to manually select them or use their ids. Please refer to the example given in the documents examples section for a demonstration of the work flow.

3 Getting Started

In order to use the functions described below you will need CUBIT[®] (preferably 15.5 or higher) and the python scripting package. CUBIT[®] 15.7 and higher contains a snapshot of the script packages located in the CUBIT[®] bin directory under python2 or 3 → Lib → site-packages. Note that scripts here will take precedence over anything loaded after start up (i.e. duplicate function names will load from the built in CUBIT[®] location). This can cause issues for updating and testing of modified scripts in CUBIT[®] 15.7. CUBIT[®] 15.8 and higher provides a directory override feature in the GUI (see Figure 2.2). Manual overrides can be done by overwriting of the base files if the user has significant permissions (simply replace the file in the above location). The scripts are specifically designed such that only the base python package is used (numpy and other package functionality was added in CUBIT[®] 15.8+). You may gather the Python-CUBIT[®] Enhancement scripting package using gitlab either via direct zip download, or by using the git functionality if on a Linux system. The scripts can then be loaded by using either starting a python journaling file with a header, or by using a CUBIT[®] initialization file on startup (search initialization files in the CUBIT[®] help documentation for more details). The gitlab repository can be accessed here with a valid account. An example of header lines in a journal file would be:

In CUBIT[®] 15.8 and up, there is a GUI command panel for easily loading scripting libraries. The Python-CUBIT[™]Enhancement libraries are already included in CUBIT[®] builds above 15.7, but alternative paths can be specified for other libraries, local copies, or updated scripts. In order to do this, simply select the python version and provide the path to the main scripting folder (This should match the python version). Restart your CUBIT[®] session to apply the modifications. See Figure 1 below for more details.

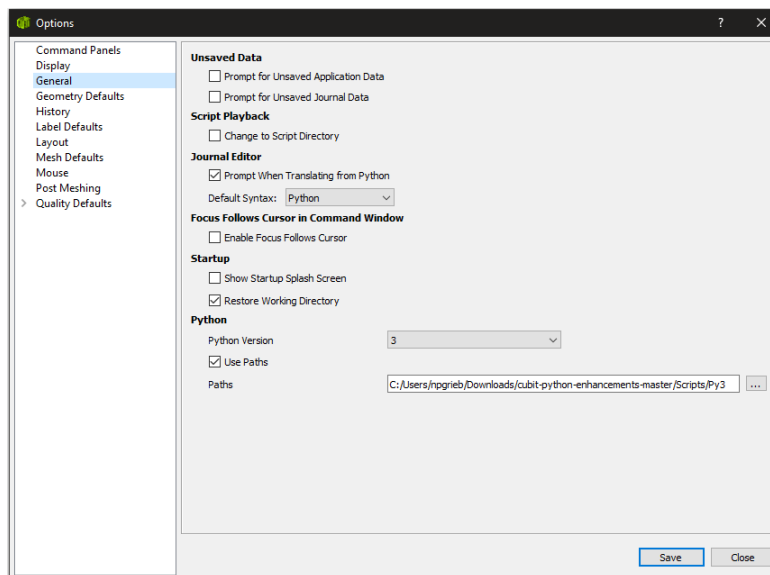


Figure 1: Specify Default Local Python Paths in CUBIT[®] 15.8+

*** NOTE: Versions of CUBIT[®] older than CUBIT[®] 15.8 will not be able to use a small number of functions due to internal python script updates. ***

for CUBIT[®] 15.7 and up, simply load the default modules using the script tab, in the CUBIT[®] command line window using the below python commands:

```
import PyCubed_Main
from PyCubed_Main import *
from cubit import *
```

for CUBIT[®] 15.6 and lower you will need to load the full path to the scripting library by appending the library to the system scope as follows:

```
#LOAD THE SCRIPTS, ADD THE PATH THEY ARE IN
#!python
import sys, os
#ADD THE SCRIPT PATH TO THE RELATIVE PATH
#NOTE THAT FOR WINDOWS FILE SYSTEMS \CANNOT BE USED BECAUSE IT IS A SPECAIAL CHARACTER IN PYTHON
#IF YOU WOULD LIKE TO COPY AND PASTE THE PATH, SINGLE \MUST BE MODIFIED TO \\
scriptPath = 'C:\\Users\\jdoe\\cubit-python-enhancements-master\\Py2\\'
sys.path.insert(0, scriptPath)
import PyCubed_Main
from PyCubed_Main import *
from cubit import *

#NOW DO SOMTHING ELSE IN THE JOURNAL FILE
cmd('create brick x 10')
#SELECT THE CURVES WITH A LENGTH EQUAL TO TEN IN ALL VOLUMES
s1 = selByLength('in vol all', 'EQ', 10.0)
...
```

The main requirement for loading the modules is adding the location path of the files to python (`sys.path.insert()` or `sys.path.append()`) after this, importing the main module and all of its functions allows all of the functions to be directly called via the CUBIT[®] scripting interface. In order to enable the scripting interface go to Tools → Options → Layout, and tick the show script box. You should now be able to either copy and paste the top portion of the above script into the scripting command panel area (after changing the `scriptPath` variable to match where you have placed the scripts) without errors. Once loaded you should be able to access all of the other functions from the scripting tab by simply calling the functions and providing the needed inputs, as done

in the second half of the above script.

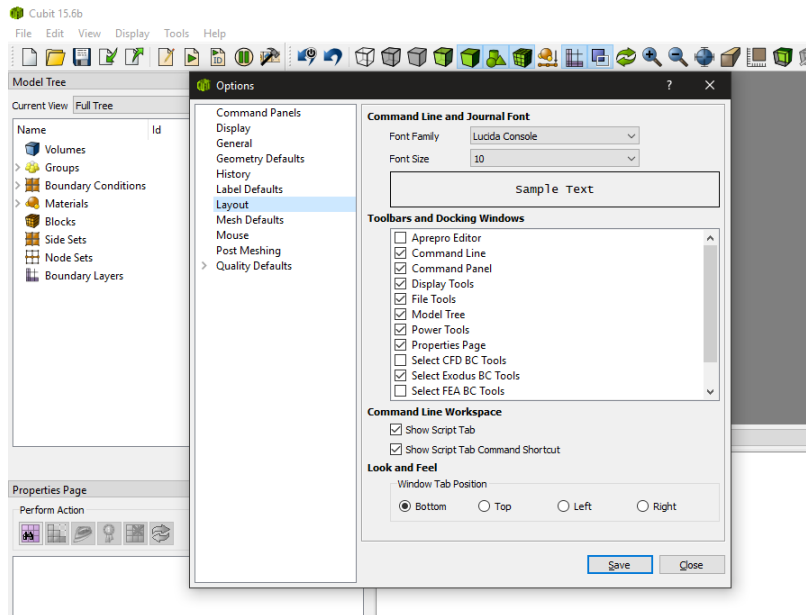


Figure 2: Accessing the CUBIT[®] Python Scripting Tab

The most convenient way to load the Python scripting modules in CUBIT[®] 15.6 and lower is to use an initialization file (see initialization files in the CUBIT[®] user manuals for more details). Different operating systems have different ways of employing these methods.

On Windows systems:

- Create a Python file containing the loading header shown above.
- Right click the CUBIT[®] desktop icon → Properties, and modify the target value in the Target field to include the Python file to initialize. For example the initialization file below is named PyCubed_AutoLoad.py, and a default working directory has been set (not a required action).

```
"C:\Program Files\Cubit 15.X\bin\claro.exe" -workingdir "H:\Documents\Analysis" "H:\Desktop\PyCubed_AutoLoad.py"
```

- Apply the changes (Note you will need to have elevated privileges for this on the Windows system.).

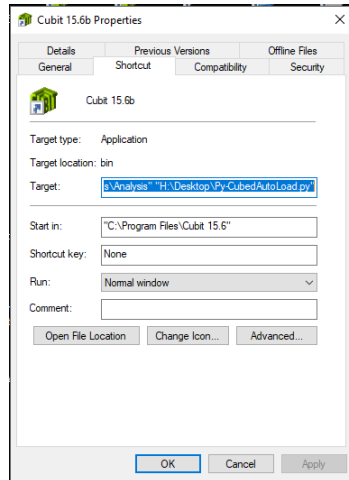


Figure 3: Setting Up the Windows Initialization File

On Linux systems:

- The simplest way to do this on Linux systems is to simply alias Cubit to Cubit PyCubed_AutoLoad.py.
- In your home (~/) directory edit your .bashrc (or other shell startup file) and add the line below.

```
alias cubit="cubit PyCubed_AutoLoad.py"
```

- Then open a terminal in the same directory and run the command **exec bash** to refresh the shell aliases.

4 Code Structure

The scripts are separated into a number of different files, each providing a different base functionality. These files are:

python only tools -

tools that perform python related functionality only (typecasting, dictionary restructuring, etc.)

vector math -

functions for basic vector math (vector addition, subtraction, dot products, etc.)

data gathering -

tools which return large amounts of data or coordinates, but which do not specifically select geometry entities

geometry selection -

tools which are focused on the selection of geometry entities (generally these will highlight the geometry items in question for user interaction)

mesh manipulation -

tools for modifying the mesh entities such as nodes, edges, or hexes (smoothing, removal, mesh definition, etc.)

geometry manipulation -

tools focused on geometry manipulation (webcutting, tweaking, regularizing, etc.)

Within the scripts there exists a basic hierarchy. At the base are the PyFuncs and VectorMath modules. These provide basic math and Python functionality that allows the user to compute desired quantities and manipulate Python data structures in order to better suit Cubit's needs. Most of the functionality in these modules is not at all dependent on the CUBIT[®] library (general purpose).

On the next level we have the GetData and SelectFuncs modules. These modules provide more refined inquiries about CUBIT[®] geometry entities and contain multiple calls to the Cubit command structure. The functions in the GetData module focus primarily on larger amounts of data, such as geometric or mesh entity size and location and will generally return these values. The functions in the SelectFuncs module focus entirely on returning Cubit entity ids, they are intended to assist the user in making batch geometry and mesh selections and provide a visual indication of the selection (currently using Cubit's highlight function). The functions in these modules depend on the VectorMath, PyFuncs, and CUBIT[®] routines to work.

On the last level we have the most complex and targeted modules: the GeomManip and MeshMods modules. These modules as their name implies focus on mesh manipulation and mesh modification. They may or may not return data (generally they move or tweak Cubit entities), and are fairly dependent on all of the other modules to function.

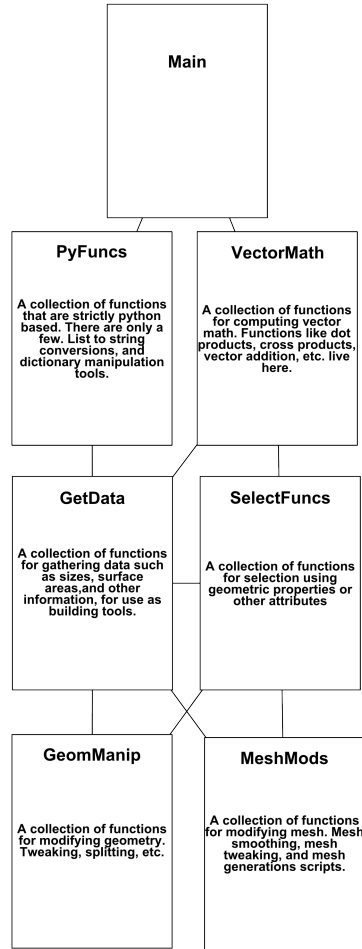


Figure 4: Script Hierarchy

5 All Functions

Currently there are 228 functions defined in the code base. They are:

```

face= list
surf= list
ACISfromNG
alignVertSplits
autoGenSidesets
autoGroupByName
autoPairSidesets
autoPopVelodyneMatIDs
autoRemovePlanarGaps
autoSpiderSidesets
autoSplitPoorHexes
blockByGroups
buildVelodyneMatLib
cadStyleAlign
cmdLooper
collapseTiers
  
```

combineBlocks
combineSidesets
compressBlocks
compressSidesets
convertVol2Beam
copyAlign
copyCombineCurves
copyMoveReplace
crossProd
deltas
dict2CSV
dictInv
differential
dirSurfSplit
dotProd
entity2BB
exoFromNG
exportBlockMats
exportCubitMatsCSV
extrComp
findDups
fireRay
gen2DRotModel
genAccelVInc
genAMRTracerVInc
genCylBB
genEnclosureInp
genOffsetVol
genPolygonImprint
genPrestoBlockMats
genSalinasBlockMats
genSlotModel
genVelodyneBlockMats
getAlignedCurves
getAngle
getArclenFromPos
getBaseName
getBaseNameDict
getBBPairs
getBBPerms
getC2CMidpoint
getConnSets
getCPProj
getCurveChainVertOrder
getCurveNorm
getCurveSegmentPoints
getCurveSizes
getDist
getDistance
getEdgeAngle
getEdgeLoopOrders
getEdgeNorm
getElemNbrs
getElemSliceMassProperties

getExtraVerts
getFaceNorm
getGeomEntityCounts
getGroupVols
getHexBSphere
getHexWithNodes
getIslandNodes
getMassProperties
getMassPropertiesExo
getMinEdgeLengths
getMinThruTh
getNNCurvePair
getNNSurfPair
getNodeBlockMap
getNodeIslandChain
getNodeNbrs
getNodeNorm
getPolylineCenter
getPoorElemVols
getSchemeless
getSurfaceLoops
getSurfaceSizes
getSweepColumnCubit
getSweepColumnPython
getTotalLength
getTotalSurfaceArea
getTotalVolume
getUDMesh
getUMCurves
getUVSamplePoints
getVertNorm
getVolBlocks
getVolGroups
getVolSidesets
getVolSurfIP
getVolumeSizes
groupConMesh
groupLinkedFlatExo
imprintOverride
isClosedLoop
l2s
l2tl
lb
lineIntersect2D
listLengSort
loadCubitMatsCSV
mapSidesets
mean
meshBasedSideset
moveNodesInNorm
moveRadially
n2cImprint
ngFromExo
nodeAvgSS

nodeAvgVS
nodeLoop2PointSpline
nodeLoopLocAvg
nodes2Surface
nodes2Surfs
nodeset2Nodeset
norm
normAvg
offsetNodes
onionizeMesh
orderEnclosures
planeCutWithCurve
planeDir
pointcloudFromCSV
pointLineInt3D
printBlockPics
printNameAndGroupVols
printNamesAndVols
proj2DThickness
recursiveTetCollapse
remFullRounds
removeAndSmooth
removeExtraVerts
renameVols
rolodexList
scalarConv
scalarMult
scaleMatsByLength
selBlendSurfs
selBoxBound
selBoxBoundExo
selByArea
selByCType
selByLength
selBySSC
selBySType
selByVolume
selConMesh
selCurveChain
selCurveVerts
selEdgeLineEnds
selEdgeLoop
selElemByNode
selExInTeriorSurfs
selHexInVol
selLinkedFlat
selLinkedFlatExo
selMergedVols
selMore
selMoreRecursive
selMultiloopSurfs
selNormSurf
selPolygonSurfs
selProjectedNodes

selProjFaceElemNorm
selProjNodeNorm
selSphericalBound
selSphericalBoundExo
selVertAt
selVertCurves
setStatGeomSize
shiftLinkingSurfSkew
shiftNodeSkew
showMaterialColors
showOverlapping
showParametricLabels
showSweeps
sidesetFromNodes
sign
skinCurveChain
slowSubtraction
smearVerts
snapToMidplane
sortCurveLoops
splintize
splintizeLocs
splitBlock
splitBoltGeom
splitRingSkew
splitSkew
splitSurfSkew
splitSurfVN
squeezeMesh
stdDev
sumSquare
sweepVolsInDir
testEnclosures
tierNormals
trapInt
trimAllLinear
trimTierPlanes
tweak3PointPlane
tweakTangent
vectAbs
vectAdd
vectAvg
vectMult
vectNorm
vectSub
vertsFromLocs
virtualTwin
vols2SameBlock
webcutWithCurveNorm
webcutWithCurveSandT

6 PyCubed_PyFuncs Overview

l2s (goto python line: 10) :

CONVERTS A LIST TO A STRING, AN ALL TOO USED FUNCTIONALITY IN PYTHON

INPUTS

- PYLIST - A PYTHON LIST
- JOINSTR - A STRING TO JOIN LIST ARGS

OUTPUTS

- A STRING WITH LIST ITEMS JOINED TOGETHER BY THE JOINING STRING

dict2CSV (goto python line: 21) :

PRINT DICTIONARY TO A CSV FILE

INPUTS

- PYDICTIONARY - THE DICTIONARY TO USE WHEN WRITING THE CSV
- CSVFILENAME - THE FILENAME (AND/OR PATH) TO USE TO WRITE THE CSV

OUTPUTS

- WRITES A CSV FILE TO THE CSV FILENAME

rolodexList (goto python line: 41) :

CYCLE THROUGH OR WRAP A LIST AS IF IT WAS CIRCULAR

INPUTS

- L - THE LIST TO CYCLE THROUGH
- N - THE NUMBER OF PLACES TO CYCLE

OUTPUTS

- REBUILDS THE LIST L, NO NEED TO CREATE A NEW VARIABLE

l2t1 (goto python line: 55) :

TAKES A LIST AND RETURNS A TUPLE LIST USING THE INTERVAL GIVEN; I.E. entityList='123, 240, 520, 637', interval = 2, GIVES [(123, 240), (520, 637)]; FOR USE WITH THE CMDLOOPER FUNCTION AND THE CUBIT ECHO FUNCTIONALITY

INPUTS

- A STRING LIST OF ITEMS WITH WHITESPACE SEPERATORS
- THE INTERVAL TO PAIR BY

OUTPUTS

- A TUPLE LIST PAIRED DOWN BY THE INTERVAL

lb (goto python line: 70) :

TAKES A LIST OF LISTS AND PREFORMS THE PYTHON SET BOOLEAN OPERATION SPECIFIED

INPUTS

- LoL - A LIST OF LISTS TO PREFORM THE OPERATIONS ON, THE SAME OPERATION OCCURS ON ALL GIVEN

OUTPUTS

- OPERATION - THE BOOLEAN OPERATION TO BE PERFORMED ON THE SET I-INTERSECT, U-UNION, D-DIFFERENCE

findDups (goto python line: 96) :

FINDS AND RETURNS DUPLICATE VALUES IN A LIST WITHOUT A COUNT

INPUTS

- NUMLIST - A LIST OF SINGLE VALUES

OUTPUTS

- DUPLICATED - A LIST OF THE VALUES DUPLICATED

dictInv (goto python line: 112) :

INVERT THE GIVEN DICTIONARY FOR BOOK KEEPING PURPOSES

INPUTS

- REGDICT - A DICTIONARY ITEM

OPTIONAL

- NONE

OUTPUTS

- INVDICT - THE INPUT DICTIONARY INVERTED

listLengSort (goto python line: 129) :

SORT A LIST OF LISTS USING THE LENGTH OF INTERNAL LISTS

INPUTS

- A LIST OF LISTS

OUTPUTS

- A LIST OF LISTS SORTED BY THE NUMBER OF INTERNAL LIST ELEMENTS

7 PyCubed_VectorMath Overview

norm (goto python line: 20) :

NORMALIZE A SET OF NUMBERS $\text{SQRT}(\text{SUM}(A^{**2}+B^{**2}+\dots))$

INPUTS

- A LIST OF FLOATING POINT NUMBERS

OUTPUTS

- THE L2 NORM AS A FLOAT

scalarMult (goto python line: 30) :

SCALAR MULTIPLICATION OF A VECTOR

INPUTS

- A FLOATING POINT NUMBER, A LIST OF FLOATING POINT NUMBERS

OUTPUTS

- A LIST OF FLOATING POINT NUMBERS

vectAdd (goto python line: 40) :

VECTOR ADDITION - ADD TWO VECTORS OF THE SAME LENGTH

INPUTS

- TWO LISTS OF FLOATS, A AND B

OUTPUTS

- A SINGLE LIST OF FLOATS A+B

vectSub (goto python line: 52) :

VECTOR SUBTRACTION - SUBTRACT TWO VECTORS OF THE SAME LENGTH

INPUTS

- TWO LISTS OF FLOATS, A AND B

OUTPUTS

- A SINGLE LIST OF FLOATS A-B

scalarConv (goto python line: 64) :

CREATE A BASIS VECTOR OF LENGTH DIM FROM A SCALAR NUMBER (I.E. 5.0 TO [0.0, 5.0, 0.0] FOR DIR 1)

INPUTS

- SCALAR - A SCALAR FLOATING POINT NUMBER
- DIR - AN INTEGER LOCATION FOR THE SCALAR, THE DIMENSION OF THE VECTOR (INTEGER)

OPTIONAL

- DIM - THE DIMENSION OF THE TARGET VECTOR (DEFAULTED TO 3)

OUTPUTS

- A LIST OF FLOATING POINT NUMBERS, WHICH IS ALL ZEROS EXCEPT FOR THE FLOOR PLACED IN THE DIR LOCATION

extrComp (goto python line: 77) :

ZERO OUT ALL OTHER COMPONENTS OF A VECTOR LEAVING THE DESIRED COMPONENTS VALUE (I.E. [2.0, 3.0, 4.0] TO [2.0, 0.0, 0.0] FOR DIR 0)

INPUTS

- VECTOR - A PYTHON LIST OF SOME LENGTH
- DIR - THE POSIION OF THE COMPONENT DESIRED

OUTPUTS

- A PYTHON LIST WITH ALL ITEMS ZEROED EXCEPT THE DESIRED COMPONENTS

dotProd (goto python line: 94) :

TAKE THE DOT PRODUCT OF TWO 3 DIMENSIONAL VECTORS

INPUTS

- TWO LISTS OF N FLOATING POINT NUMBERS

OUTPUTS

- A SCALAR FLOATING POINT NUMBER, THE DOT PRODUCT

crossProd (goto python line: 107) :

TAKE THE CROSS PRODUCT OF TWO 3 DIMENSIONAL VECTORS

INPUTS

- TWO LISTS OF 3 FLOATING POINT NUMBERS

OUTPUTS

- A LIST OF 3 FLOATING POINT NUMBERS, THE CROSS PRODUCT

getAngle (goto python line: 123) :

GET THE ANGLE BETWEEN THE TWO VECTORS OR SURFACE NORMALS

INPUTS

- 2 SURFACE IDS (INTEGERS) OR TWO 3D VECTOR LISTS (FLOAT LISTS)

OUTPUTS

- THE ANGLE IN DEGREES BETWEEN THE TWO VECTORS (FLOAT)

getDist (goto python line: 155) :

NON-GEOCENTRIC GET DISTANCE; RETURN THE DISTANCE BETWEEN TWO 3D VECTORS

INPUTS

- TWO PYTHON LISTS OF LENGTH 3

OUTPUTS

- THE MAGNITUDE OF THE DISTANCE AND THE VECTOR DISTANCE

vectMult (goto python line: 166) :

MULTIPLY COMPONENTS IN A VECTOR $A_i * B_i$

INPUTS

- TWO PYTHON LISTS OF LENGTH N

OUTPUTS

- A VECTOR OF LENGTH N

vectAvg (goto python line: 179) :

AVERAGE N NUMBER OF 3D VECTORS

INPUTS

- A LIST OF 3D VECTOR LISTS TO BE AVERAGED

OUTPUTS

- THE AVERAGE POINT DEFINED BY A 3D VECTOR LIST [X, Y, Z]

normAvg (goto python line: 197) :

GET THE AVERAGE NORMAL VECTOR FOR THE SET OF SURFACES, MAKE GUESSES BASED ON OPPOSING SETS

INPUTS

- NLS - A LIST OF THE NORMAL DIRECTIONS $[[n1,n2,n3], [n1,n2,n3]]$
- CPS - A LIST OF CENTER POINT LOCATIONS FOR EACH NORMAL $[[x,y,z],[x,y,z]]$

OUTPUTS

- NAVG - THE AVERAGE NORMAL DIRECTION VECTOR AS A LIST OF

vectAbs (goto python line: 227) :

RETURN ALL POSITIVE VALUES FOR A GIVEN VECTOR, I.E. $[|x|, |y|, |z|]$

INPUTS

- TWO PYTHON LISTS OF LENGTH N

OUTPUTS

- A PYTHON LIST OF LENGTH N

sign (goto python line: 239) :

RETURN THE SIGN OF THE PASSED ARGUMENT AS A MULTIPLIER

INPUTS

- A FLOAT OR INTEGER NUMBER

OUTPUTS

- EITHER A 1 OR -1 BASED ON THE SIGN OF THE GIVEN INPUT

planeDir (goto python line: 253) :

GET THE PLANAR DIRECTION USING 3 POINTS

INPUTS

- A PYTHON LIST CONTAINING THE COORDINATES OF A POINT (3 FLOAT VALUES)
- A PYTHON LIST CONTAINING THE COORDINATES OF A POINT (3 FLOAT VALUES)
- A PYTHON LIST CONTAINING THE COORDINATES OF A POINT (3 FLOAT VALUES)

OUTPUTS

- A PYTHON LIST CONTAINING ONE OF THE TWO UNIT NORMAL DIRECTIONS POSSIBLE

vectNorm (goto python line: 268) :

NORMALIZE THE VECTOR TO A UNIT VECTOR $A/|A|$

INPUTS

- A VECTOR AS A LIST

OUTPUTS

- THE NORMALIZED UNIT VECTOR AS A LIST BASED ON THE INPUT

mean (goto python line: 281) :

GET THE MEAN VALUE OF A 1D LIST

INPUTS

- A LIST OF FLOATS OR INTEGERS

OUTPUTS

- THE MEAN OF THE LIST (AS A FLOAT)

sumSquare (goto python line: 295) :

RETURN THE SUM SQUARED OF THE DATA

INPUTS

- A LIST OF FLOATS OR INTEGERS

OUTPUTS

- THE SUM SQUARED OF THE DATA AS A FLOAT

stdDev (goto python line: 306) :

RETURN THE STANDARD DEVIATION OF A LIST OF FLOATS

INPUTS

- A LIST OF FLOATS OR INTEGERS

OPTIONAL

- THE STANDARD DEVIATION DEGREE OF FREEDOM
- 0 OR 1 GENERALLY

OUTPUTS

- THE STANDARD DEVIATION

deltas (goto python line: 324) :

GET THE DIFFERENCE BETWEEN NEIGHBORING VALUES IN A 1D ARRAY

INPUTS

- A LIST OF FLOATING POINT OR INTEGER VALUES (OF LENGTH N)

OUTPUTS

- A LIST OF FLOATING POINT VALUES LENGTH N-1

differential (goto python line: 337) :

GET THE DISCRETE DERIVATIVE OF THE FUNCTION GIVEN A TUPLE LIST [(X1,Y1), (XN,YN)]

INPUTS

- A LIST OF TUPLES CONTAINING 2 FLOAT VALUES

OUTPUTS

- THE DERIVATIVE AS A LIST OF

trapInt (goto python line: 351) :

INTEGRATE A FUNCTION GIVEN A TUPLE LIST [(X1,Y1), (XN,YN)] USING THE DISCRETE TRAPEZOIDAL METHOD

INPUTS

- A LIST OF TUPLES CONTAINING 2 FLOAT VALUES

OUTPUTS

- THE INTEGRATED VALUE OF THE FUNCTION USING THE DISCRETE TRAPEZOIDAL METHOD

pointLineInt3D (goto python line: 366) :

PROJECT A POINT ONTO A LINE IN 3 DIMENSIONAL SPACE GIVEN A LINE CENTER POINT AND NORMAL, AND THE POINTS LOCATION

INPUTS

- LINEORIGIN - THE ORIGIN OF THE LINE AS A 3D VECTOR OR LIST OF FLOATS
- LINENORM - THE NORMAL VECTOR AS A 3D VECTOR OR LIST OF FLOATS
- POINT - THE POINT TO BE PROJECTED TO THE LINE

OUTPUTS

- NPP - THE NEAREST POINT PROJECTED ON THE LINE

lineIntersect2D (goto python line: 377) :

FIND THE INTERSECTION POINTS OF THE TWO LINES GIVEN

INPUTS

- 2 LISTS OF TUPLES OF LENGTH 2 (2D POINT LISTS)

OUTPUTS

- THE INTERSECTION POINT OF THE TWO LINES AS A TUPLE

proj2DThickness (goto python line: 405) :

PROJECT ANOTHER LINE, FROM A LINE; MAKING A PARALLEL THICKNESS CONSTRAIN *** ASSUMES THAT WE ARE USING ONLY X AND Y COMPONENTS ***

INPUTS

- 2 VERT IDS THAT FORM THE LINE (REMOVES THE Z COMP), THE OFFSET THICKNESS, AND A FLIP IF NEEDED
- OUTPUTS
- THE PROJECTED POINTS FROM THE ORIGINAL

8 PyCubed_GetData Overview

getBaseName (goto python line: 32) :

REMOVE THE AT NOTATION FROM THE NAME

@C OR @@@@@@B

INPUTS

- THE ENTITY NAME AS A STRING WITH THE CUBIT INSTANCE NOTATION

OUTPUTS

- THE CUBIT NAME WITHOUT THE @ NOTATION, THIS PROVIDES A GENERIC NAME REFERENCE FOR ALL INSTANCES

getBaseNameDict (goto python line: 45) :

GET A DICTIONARY OF VOLUMES PER EACH UNIQUE BASE NAME

INPUTS

- NONE

OUTPUTS

- A DICTIONARY OF BASE NAMES, CONTAINING A LIST OF VOLUME IDS WITH THAT BASE NAME

getCurveNorm (goto python line: 62) :

GET NORMAL DIRECTION OF STRIGHT LINES OR CURVES IN A VERT TO VERT MANNER

INPUTS

- EITHER 2 VERTEX IDS AS INTEGERS OR A CURVE ID AS AN INTEGER

OUTPUTS

- A UNIT NORMAL DIRECTION CREATED BY THE TWO VERTICIES

getEdgeNorm (goto python line: 86) :

GET THE NORMAL DIRECTION OF AN EDGE

INPUTS

- AN EDGE ID AS AN INTEGER

OUTPUTS

- THE EDGE UNIT NORMAL DIRECTION
- THE IDS OF THE TWO CHILD NODES

getFaceNorm (goto python line: 103) :

GET THE FACE NORM OF A SKINNED SHEET ELEMENT

INPUTS

- FACEID - THE FACE ID AS AN INTEGER

OPTIONAL

- ETYPE - THE ELEMENT TYPE TO CONSIDER FACE/QUAD OR TRI (DEFAULT IS FACE)

OUTPUTS

- FACENORM - THE FACE NORMAL AS A UNIT VECTOR (LIST)
- FACE_CENTER - THE FACE CENTER AS A LIST OF FLOATS
- FACE_NODES - A LIST OF THE NODES IN THE FACE

getNodeNorm (goto python line: 120) :

GET THE NODE NORMAL (I.E. THE AVERAGE VALUE BASED ON SHARED FACES)

INPUTS

- NODEID - THE NODE ID AS A CUBIT STRING OR INTEGER

OUTPUTS

- NNORM - THE NORMAL VECTOR AS A LIST OF FLOAT VALUES

getEdgeAngle (goto python line: 141) :

GET THE EDGE BREAK ANGLE FOR THE EDGE GIVEN

INPUTS

- EDGEID - THE EDGE ID AS AN INTEGER

OUTPUTS

- EDGEANGLE - THE EDGE ANGLE AS A FLOAT IN DEGREES
- EDGETANGENT - THE TANGENT DIRECTION VECTOR

getVertNorm (goto python line: 175) :

GET THE VERTEX NORMAL (I.E. THE AVERAGE VALUE BASED ON SHARED SURFACE NORMS)

INPUTS

- VERTID - THE NODE ID AS AN INTEGER OR STRING

OUTPUTS

- VERTNORM - THE VERTEX NORMAL

getEdgeLoopOrders (goto python line: 189) :

GET EDGE LOOP AND EDGE NODE ORDER

INPUTS

- EL - A LIST OF CONNECTED EDGE IDS

OUTPUTS

- ORDERED_EDGES - A LIST OF THE ORDERED NODES IN THE SUPPLIED EDGES AS A LIST OF IDS
- ORDERED_NODES - A LIST OF ORDERED NODES IN THE SUPPLIED EDGES AS A LIST OF NODE IDS

getAlignedCurves (goto python line: 247) :

GET CURVES ALIGNED TO THE GIVEN ALIGNMENT DIRECTION

INPUTS

- CURVELIST - A LIST OF CURVES AS A CUBIT LIST STRING
- ALIGNEDDIR - A DIRECTION TO CHECK AGAINST AS A VECTOR LIST [X,Y,Z]

OPTIONAL

- TOL - AN ANGLE TOLERANCE TO DETERMINE THE ALIGNMENT IN DEGREES (DEFAULT IS 25 DEGREES)

OUTPUTS

- ACS - THE ALIGNED CURVE IDS

getExtraVerts (goto python line: 261) :

FIND VERTEX SPLITS THAT DONT CONTRIBUTE TO THE GEOMETRY AND RETURN THEM

INPUTS

- VERTEXLIST - A VERTEX LIST TO REGULARIZE

OUTPUTS

- EXTRAVERTS - THE VERTS THAT DONT HAVE A T-JUNCTION

getConnSets (goto python line: 276) :

RETURN A LIST OF LIST CONTAINING SURFACES THAT ARE CONNECTED ****PROBABLY FASTER USING AN ADVANCING FRONT****

INPUTS

- SURFLIST - A CUBIT LIST STRING OF SURFACES THAT ARE TO BE SORTED

OUTPUTS

- CONNSETS - A LIST OF LISTS CONTAINING CONNECTED SETS

getCurveSizes (goto python line: 319) :

QUERY ALL CURVE SIZES FOR THE VOLUMES IN THE VOLUME LIST

INPUTS

- A LIST OF VOLUME IDS AS A STRING

OUTPUTS

- A PYTHON DICTIONARY WITH VOLUME IDS AS KEYS, WHICH CONTAINS A TUPLE LIST OF CURVE IDS AND CURVE LENGTHS

getSurfaceSizes (goto python line: 335) :

QUERY ALL SURFACE AREAS FOR THE VOLUMES IN THE VOLUME LIST

INPUTS

- A LIST OF VOLUME IDS AS A STRING

OUTPUTS

- A PYTHON DICTIONARY WITH VOLUME IDS AS KEYS, WHICH CONTAINS A TUPLE LIST OF SURFACE IDS AND SURFACE AREAS

getVolumeSizes (goto python line: 351) :

QUERY THE VOLUME FOR THE VOLUMES IN THE VOLUME LIST

INPUTS

- A LIST OF VOLUME IDS AS A STRING

OUTPUTS

- A TUPLE LIST OF VOLUME IDS AND VOLUMES

getTotalSurfaceArea (goto python line: 364) :

GET THE TOTAL SURFACE AREA FOR THE PROVIDED VOLUMES

INPUTS

- VOLLIST - A LIST OF VOLUMES CONTAINING THE DESIRED SURFACES

OUTPUTS

- TSA, SADICT - THE TOTAL SURFACE AREA AS A FLOAT VALUE AND THE TOTAL SURFACE AREA DICTIONARY

getTotalVolume (goto python line: 380) :

GET THE TOAL VOLUME OF ALL BODIES IN THE FILE

INPUTS

- NONE

OUTPUTS

- THE TOTAL VOLUME OF ALL ITEMS

getGeomEntityCounts (goto python line: 396) :

GET THE TOTAL NUMBER OF GEOMETRIC ENTITIES IN THE MODEL

INPUTS

- NONE

OUTPUTS

- A LIST OF GEOMETRY ITEM COUNTS IN THIS ORDER [VOLUMES, SURFACES, CURVES, VERTS]

getMinEdgeLengths (goto python line: 409) :

TRY AND GATHER THE MINIMUM EDGE DISTANCE PER ELEMENT, PER BLOCK ****MIGHT FAIL FOR HIGHER ORDER ELEMENTS DUE TO DISTORTION****

INPUTS

- BLOCKIDLIST - A CUBIT LIST STRING DENOTING THE BLOCKS TO BE INVESTIGATED

OUTPUTS

- BLOCKELEMLIST - A DICTIONARY WITH BLOCK IDS AS KEYS, CONTAINING A LIST OF TUPLES WITH (GLOBAL ELEMENT IDS, SMALLEST EDGE LENGTH)

getPoorElemVols (goto python line: 477) :

GATHER THE QUALITY OF EACH HEX IN EACH VOLUME IN THE LIST AND OUTPUT A LIST OF THE VOLUMES WITH POOR SCALED JACOBIANS

INPUTS

- VOLLIST - A CUBIT LIST STRING OF VOLUMES
- OPTIONAL
- SJM - SCALED JACOBIAN MINIMUM REQUIREMENT TO USE
- OUTPUTS
- QPV - QUALITIES PER VOLUME, THE QUALITY OF ALL HEXES IN THE VOLUME
 - BADVOLS - THE VOLUMES THAT HAVE ELEMENTS WITH A SCALED JACOBIAN OF LESS THAN THE GIVEN VALUE

fireRay (goto python line: 498) :

FIRE A RAY FROM THE GIVEN LOCATION IN THE GIVEN NORMAL DIRECTION

INPUTS

- RAYCENTER - A POINT OF ORIGIN FOR THE RAY
- RAYNORMAL - A NORMAL POINT FOR THE RAY
- OBJECTCENTERS - A LIST OF OBJECT CENTER POINTS OR POINTS OF INTEREST TO MEASURE TO
- OBJECTIDS - A LIST OF CORRESPONDING OBJECT IDS IN THE SAME ORDER AS THE GIVEN OBJECT CENTERS
- DIST TOL - A DISTANCE FROM THE RAY (BEAM THICKNESS)

OUTPUTS

- NEARFIRE - A LIST OF THE ENTITY IDS THAT ARE WITHIN THE DISTTOL FROM THE RAY

getMinThruTh (goto python line: 516) :

TRY AND FIND THE MINIMUM THROUGH THICKNESS COUNT PER BLOCK ** VERY SLOW FOR TET MESHES **

INPUTS

- BLOCKIDLIST - THE BLOCKS TO INVESTIGATE

OPTIONAL

- MESHTOL - A MULTIPLIER OF THE MESH SIZE TO SEARCH INSIDE OF FOR TET MESHES (0.3 FOR NORMAL TET STRUCTURE, 0.5 FOR HTET STRUCTURE)

OUTPUTS

- BMINS - A LIST OF TUPLES CONTAINING THE BLOCK ID AND THE MIN THROUGH THICKNESS FOUND IN THAT BLOCK

getSweepColumnPython (goto python line: 609) :

RETURN THE SWEEP COLUMN GIVEN THE FACEID ** ONLY WORKS FOR HEXES **

INPUTS

- FACEID - A FACE ID AS AN INTEGER

OUTPUTS

- HEXLIST - A LIST OF THE HEX IDS IN THE SWEEP COLUMN

getSweepColumnCubit (goto python line: 633) :

RETURN THE SWEEP COLUMN GIVEN THE FACEID ** ONLY WORKS FOR HEXES **

INPUTS

- FACEID - A FACE ID AS AN INTEGER

OUTPUTS

- HEXLIST - A LIST OF THE HEX IDS IN THE SWEEP COLUMN

getHexWithNodes (goto python line: 647) :

RETURN THE HEX(ES) CONTAINING ALL AND ONLY THE NODES PROVIDED, NONE IS RETURNED IF THERE ARE NO MATCHES

INPUTS

- NODEIDS - A CUBIT LIST STRING OF NODE IDS

OUTPUTS

- HEXID - THE HEX CONTAINING THE NODES GIVEN

getElemNbrs (goto python line: 668) :

RETURN ELEMENTS SHARING A NODE

INPUTS

- ETYPE - THE TYPE OF EXODUS ENTITY

OUTPUTS

- NBRS - THE NEIGHBORING ELEMENTS EXCLUDING THE ORIGINAL ID

getNodeNbrs (goto python line: 679) :

RETURNS A LIST OF THE NODES IMMEDIATELY CONNECTED TO THE NODE PROVIDED

INPUTS

- NODEID - A NODE ID

OUTPUTS

- NBRS - THE NODES MOST IMMEDIATELY CONNECTED TO THE PROVIDED NODE, WITHOUT THE ORIGINAL IN THE LIST

getIslandNodes (goto python line: 690) :

GET A SINGLE ISLAND BASED ON NODE CONNECTIVITY WITH RESPECT TO A SELECTION

INPUTS

- POOLNODES - A CUBIT LIST STRING OF NODES TO SEARCH OUTPUTS
- ISLAND - A LIST OF NODE IDS WHICH CREATE AN ISLAND

getNodeIslandChain (goto python line: 714) :

GET ISLAND CHAINS BASED ON THE NODE SELECTION

INPUTS

- POOLNODES - A CUBIT LIST STRING OF NODES TO SEARCH

OUTPUTS

- ISLANDS - A LIST OF LISTS CONTAINING NODE IDS FOR EACH ISLAND

onionizeMesh (goto python line: 731) :

GET A LEVEL-SET LIKE LAYER COUNT OF THE MESH FOR A GIVEN VOLUME

INPUTS

- ETYPE - THE ELEMENT TYPE
- VOLID - THE MESHED VOLUMES ID

OUTPUTS

- LAYERDICT - A DICTIONARY OF THE MESH ELEMENTS IN THE VOLUME DENOTING THE LEVELS OF DEPTH OF THE GIVEN ELEMENT ID

getPolylineCenter (goto python line: 772) :

GETS THE POLYLINE CENTER (LENGTH AND POSITION WEIGHTED AVERAGES) FROM A LIST OF CURVES; USEFUL FOR ALIGNMENTS

INPUTS

- A LIST OF CURVES AS A STRING

OUTPUTS

- THE CENTER POINT OF THE PROVIDED CURVES

getTotalLength (goto python line: 799) :

GET THE TOTAL LENGTH OF THE CURVE LIST

INPUTS

- THE CURVE ID LIST STRING AS A CUBIT STRING

OUTPUTS

- THE TOTAL LENGTH OF THE CURVES IN THE GIVEN STRING AS A FLOAT

getUMCurves (goto python line: 811) :

ATTEMPT TO MERGE VOLUMES, AND GET A LIST OF THE UNMERGED CURVE PAIRS FOUND WHEN HAVING TROUBLE FULLY MERGING GIVEN VOLUMES.

INPUTS

- VLIST - A LIST OF VOLUMES AS A STRING

OPTIONAL

- UNMERGE - A BOOLEAN FLAG THAT ALLOWS THE USER TO KEEP THE MESH FROM UNMERGING

OUTPUTS

- A LIST OF THE CURVES THAT ARE UNMERGED (PROBLEMATIC)

getUDMesh (goto python line: 839) :

GET THE HARDCODED MESHING PARAMETERS FROM THE MESH. PRINT THESE PARAMETERS OUT USING CUBIT STYLE COMMANDS FOR REUSE/REAPPLICATION ON OTHER MESHES

INPUTS

- NONE

OUTPUTS

- A LIST PRINTED IN THE CUBIT WINDOW OF THE USER DEFINED PARAMETERS USED FOR MESHING THESE. INCLUDE SET VOLUME SIZES, CURVE INTERVALS, ETC.

sortCurveLoops (goto python line: 874) :

GIVEN A CURVE LOOP LIST SORT THEM WRT TOTAL LENGTH

INPUTS

- A LIST OF LISTS CONTAINING CURVE IDS

OUTPUTS

- THE SAME LIST SORTED FROM LEAST TO GREATEST

getCurveChainVertOrder (goto python line: 886) :

GET THE VERT ORDER OF THE CURVE CHAIN, GIVEN AS A TEXT LIST

INPUTS

- A CURVE CHAIN LIST (USUALLY FROM CURVE CHAIN SELECTION OR SURFACE LOOP SELECTION FUNCTIONS, AS A CUBIT STRING)

OPTIONAL

- A STARTING VERTEX POINT AS AN INTEGER ID
- A BOOLEAN FLAG FOR ORDER REVERSAL

OUTPUTS

- A LIST OF VERTEX IDS IN A SEQUENTIAL ORDER DOWN THE CHAIN

getSurfaceLoops (goto python line: 966) :

GATHER THE LOOPS CONTAINED IN A SURFACE

INPUTS

- A SURFACE ID AS AN INTEGER

OUTPUTS

- A LIST OF LISTS (CURVE IDS), CONTAINING ALL CLOSED CURVE LOOPS IN THE SURFACE

getC2CMidpoint (goto python line: 1002) :

GET THE MIDPOINT PROJECTION BETWEEN THE SPECIFIED CURVES PROJECTING THE SEGMENTATION POINTS

INPUTS

- CURVE 1 ID - THE CURVE TO BE SEGMENTED
- CURVE 2 ID - THE CURVE TO BE PROJECTED ONTO
- NSEGS - THE NUMBER OF SEGMENTATION POINTS TO PROJECT

OUTPUTS

- A LIST OF POINTS AN AVERAGE DISTANCE BETWEEN THE TWO CURVES

getVolSurfIP (goto python line: 1019) :

FIND INTERSECTION POINTS OF VOLUMES WRT A VOLUMES SURFACE; INTENDED FOR SLOT MODEL USE, BUT POTENTIALLY USEFUL FOR OTHER TIES PROVIDE A VOLUME LIST FOR INTERSECTION CALC, VOLUME OF INTEREST AND SURFACE OF INTEREST

INPUTS

- A LIST OF VOLUMES AS A STRING (FOR INTERSECTION CALCULATION) *** THIS LIST SHOULD INCLUDE THE VOLUME OF INTEREST AS WELL AS THE VOLUMES INTERSECTING IT ***
- A VOLUME ID AS AN INTEGER DENOTING THE VOLUME OF INTERESTS CUBIT ID
- A SURFACE ID AS AN INTEGER DENOTING THE SURFACE OF INTERESTS CUBIT ID

OPTIONAL

- A PROJECTION TOLERANCE USED TO DETERMINE IF THE ITEM SHOULD BE PROJECTED

OUTPUTS

- A LIST OF THE INTERSECTION POINTS GIVEN AS LISTS

getCurveSegmentPoints (goto python line: 1056) :

SWEEP THE U SPACE OF A GIVEN CURVE USING A CONSTANT SPACING SCHEME GIVEN N SEGMENTS URMIN AND URMAY CAN BE USED TO SUBSEGMENT BETWEEN THE GIVEN U PARAMS *** NOTE THERE IS NO IN-BOUNDS CHECK *** ; RETURNS THE PHYSICAL POINT LOCATION LIST AND ASSOCIATED U PARAMETER LIST

INPUTS

- THE CURVE ID AS AN INTEGER
- THE DESIRED NUMBER OF SEGMENTS (REMEMBER THE FIRST AND LAST POINT ARE THE END VERTS)

OPTIONAL

- A MINIMUM U POSITION TO START FROM IF DESIRED
- A MAXIMUM U POSITION TO END AT IF DESIRED

OUTPUTS

- A LIST OF THE POINTS IN THE CURVE SEGMENT IN PHYSICAL SPACE
- A LIST OF THE POINTS IN THE CURVE SEGMENT IN U SPACE

getUVSamplePoints (goto python line: 1081) :

GATHER A GRID OF UV SAMPLE POINTS FROM A SURFACE

INPUTS

- SID - THE SURFACE ID AS AN INTEGER OR CUBIT STRING ITEM

OPTIONAL

- INTERVALS - THE SIZE OF THE NUMBER OF GRID INTERVALS TO USE IN BOTH U AND V (AN EVEN GRID SPACING; 15 BY DEFAULT)
- TRIM - OPTION TO TRIM THE UVS TO MATCH THE PHYSICALLY TRIMMED SURFACE (FALSE BY DEFAULT)

OUTPUTS

- PLGRIDPOINTS - THE PHYSICAL LOCATIONS OF EACH UV POINT AS A LIST OF LISTS; [X, Y, Z] COORDINATE LIST
- UVGRIDPOINTS - THE UV LOCATION OF EACH POINT AS A LIST OF LISTS; [U,V] COORDINATE LIST
- TRIMMEDGRID - THE TRIMMED GRID IF TRIM IS TRUE

getArcLenFromPos (goto python line: 1113) :

GET THE LENGTH BETWEEN VECTOR POSITIONS IF THEY ARE ON THE SAME CURVE

INPUTS

- THE CURVE ID TO CHECK AS AN INTEGER
- THE FIRST LOCATION AS A LIST OF FLOATS (X,Y,Z)
- THE SECOND LOCATION AS A LIST OF FLOATS (X,Y,Z)

OUTPUTS

- THE ARCLENGTH BETWEEN THE TWO POINTS

getBBPerms (goto python line: 1127) :

GET THE BOUNDING BOX CORNER POINTS AS 8 [X, Y, Z] LISTS

INPUTS

- THE TYPE OF GEOMETRY TO BOUND SURFACE OR VOLUME
- THE ID OF THE GEOMETRY ENTITY AS AN INTEGER

OUTPUTS

- A LIST OF THE 8 BOUNDING BOX POINTS AS LISTS PLUS THE CENTER POINT

getBBPairs (goto python line: 1150) :

GET THE GEOMETRY BOUNDING BOX AS A SERIES OF 3 MIN MAX PAIRS, INTENDED TO WORK EASILY WITH SELBOXBOUND CALLS (BOUNDING BOX IS GLOBAL)

INPUTS

- GEOMTYPE - THE TYPE OF GEOMETRY BEING USED TO CALCULATE THE (CURVE, SURFACE, VOLUME)

- IDLIST - THE ID LIST AS A CUBIT LIST STRING, MULTIPLE ITEMS WILL RESULT IN A COMBINED BOUNDING BOX OUTPUTS

- XS, YS, ZS, D - THE X, Y, AND Z BOUNDS AS A MIN MAX PAIR [XMIN, XMAX], [YMIN, YMAX], [ZMIN, ZMAX] AND THE TOTAL DIAGONAL AS A FLOAT

getCProj (goto python line: 1166) :

GET A DIRECT CURVE PROJECTION POINT BASED ON CURVE TANGENT AND POINT POSITION

RETURNS THE POINT LOCATION *** CAREFUL THIS IS RECURSIVE ***

INPUTS

- CURVE ID - THE CURVE ID AS AN INTEGER
- PROJPOINT - THE POINT TO PROJECT TO AS A LIST

OPTIONAL

- UMIN - A MIN U PARAMETER START POINT IF DESIRED
- UMAX - A MAX U PARAMETER START POINT IF DESIRED
- NS - NUMBER OF SEGMENTS FOR BISECTION (MORE NEEDED FOR FULL LOOPS?)
- TOL - THE ANGULAR TOLERANCE TO THE PROJECTED POINT (THE SCRIPT ATTEMPTS TO MAKE A 90 DEGREE PROJECTION FROM THE CURVE TANGENT)

OUTPUTS

- THE LOCATION OF THE POINT PROJECTED ONTO THE SELECTED CURVE AS A LIST OF FLOATS (3D VECTOR)

getSchemeless (goto python line: 1216) :

GET VOLUMES THAT CURRENTLY HAVE NO AUTOScheme ASSIGNED THIS GENERALLY MEANS THEY ARE COMPLEX OR MULTISWEEP VOLUMES

INPUTS

- A LIST OF VOLUME IDS AS A STRING

OUTPUTS

- A LIST OF VOLUME IDS WITH NO SET SCHEME

getGroupVols (goto python line: 1232) :

RETURNS THE VOLUMES IN A GROUP LIST

INPUTS

- A LIST OF GROUP IDS AS A CUBIT STRING

OPTIONAL

- THE TYPE OF OUTPUT DESIRED (NAME OR ID, DEFAULT IS ID)

OUTPUTS

- A DICTIONARY WITH THE GROUP IDS AS A KEY, WITH A LIST OF CONTAINED VOLUMES

getVolGroups (goto python line: 1250) :

RETURNS A LIST OF GROUPS THAT CONTAIN THE LIST OF VOLUMES GIVEN OUTPUT AS NAME OR ID

INPUTS

- A VOLUME LIST AS A CUBIT STRING

OPTIONAL

- THE TYPE OF OUTPUT DESIRED (NAME OR ID, DEFAULT IS ID)

OUTPUTS

- A DICTIONARY WITH THE VOLUME IDS AS A KEY, WITH A LIST OF MEMBER GROUPS

getVolBlocks (goto python line: 1273) :

RETURNS A DICTIONARY OF VOLUME IDS WITH THEIR RESPECTIVE BLOCKS

INPUTS

- NONE

OUTPUTS

- A PYTHON DICTIONARY WITH THE FEA MODEL VOLUMES AS THE KEY, LISTING THEIR RESPECTIVE BLOCKS

getVolSidesets (goto python line: 1288) :

GRAB SIDESSETS ASSOCIATED WITH THE SELECTED VOLUMES

INPUTS

- VOLLIST - A CUBIT LIST STRING OF VOLUME IDS

OUTPUTS

- SSIV - SIDESSETS IN THE VOLUMES

surf= list (goto python line: 1295) :

- DOCUMENTATION NOT YET AVAILABLE...

face= list (goto python line: 1296) :

- DOCUMENTATION NOT YET AVAILABLE...

getHexBSphere (goto python line: 1301) :

GET A HEX ELEMENT'S BOUNDING SPHERE. RETURNS THE HEX CENTER POINT AND THE MAXIMUM NODE-NODE DISTANCE (I.E HEX DIAGONAL) AS A DIAMETER

INPUTS

- THE HEX ID AS AN INTEGER

OUTPUTS

- THE HEXES DIAGONAL DISTANCE AND ITS CENTER POINT

getNNCurvePair (goto python line: 1314) :

GET THE NEAREST NEIGHBORING CURVE PAIRS, BARRING CURVES ARE ACTUALLY JOINED

INPUTS

- CURVELIST1 - A CUBIT LIST STRING OF THE CURVES IN THE FIRST CURVE CHAIN
- CURVELIST2 - A CUBIT LIST STRING OF THE CURVES IN THE SECOND CURVE CHAIN

OPTIONAL

- TOL - A SEARCH TOLERANCE
- MATCHANGLE - MATCH THE ANGLES WITHIN A TOLERANCE
- MANGLE - THE ANGLE TOLERANCE TO MATCH IN ORDER TO PAIR THE CURVES

OUTPUTS

- NNCPAIRS - A TUPLE LIST CONTAINING THE NEAREST NEIGHBORING CURVES AS A TUPLE LIST

getNNSurfPair (goto python line: 1363) :

GATHER THE NEAREST NEIGHBORING SURFACES USING TWO LISTS OF SURFACES, LIST 1 TO LIST 2

INPUTS

- SURFLIST1 - A LIST OF SURFACES AS A CUBIT LIST STRING
- SURFLIST2 - A LIST OF SURFACES TO PAIR AS A CUBIT LIST STRING

OPTIONAL

- TOL - A TOLERANCE TO SEARCH WITHIN, IF THE VALUE IS BELOW THIS TOLERANCE KEEP IT

- MINTOL - A MINIMUM TOLERANCE, IF THE TOLERANCE IS BELOW THIS TOSS IT OUT (THIS SHOULD BE WITHIN THE MERGE TOLERNACE)
- ANGLETOL - A TOLERANCE TO FILTER FACE ANGLE IN PAIR CALCULATIONS
- DISTCALC - MODIFY THE DISTANCE CALCULATION FROM CENTER TO CENTER TO CENTER TO NEAREST PROJECTED POINT TO NORMAL PROJECTION (CENTER, NORMAL, PROJECTED)

OUTPUTS

- NNSURFS - A LIST OF THE NEAREST NEIGHBOR SURFACE PAIRS *** THIS CAN DOUBLE UP ON SURFACES ***

getNodeBlockMap (goto python line: 1413) :

GET A BLOCK MAPPING PER NODE (MAPPINGS WITH LENGTHS GREATER THAN 1 ARE INTERFACES)

INPUTS

- BLOCKLIST - THE LIST OF BLOCKS TO MAP TO AS A CUBIT LIST STRING

OUTPUTS

- MAPDICT - THE MAPPING DICTIONARY WITH THE NODE AS THE KEYS, BLOCKS ARE GIVEN USING A LIST (MULTIPLES EXIST ON MERGERS)

showOverlapping (goto python line: 1438) :

CHECKS FOR OVERLAP AND DRAWS ONLY PARTS OVERLAPPING IN THE LIST OF VOLUMES PROVIDED, IF NOT PROVIDED ALL ASSUMED

INPUTS

- A LIST OF VOLUMES AS A CUBIT STRING (IF NONE PROVIDED ALL IS ASSUMED)

OPTIONAL

- A BOOLEAN TO TOGGLE DRAWING ON OR OFF (DRAWS BY DEFAULT (TRUE))
- A LIST OF VOLUMES TO EXCLUDE FROM THE INTERSECTION OF THE MAIN LIST AS A CUBIT LIST STRING

OUTPUTS

- A LIST OF THE OVERLAPPING GEOMETRIES, A PRINTED COMMAND LIST TO DRAW THEM INDIVIDUALLY

showSweeps (goto python line: 1479) :

GET A VISUAL HANDLE ON THE MESH SWEEPING SCHEME THAT CUBIT IS USING

HIGHLIGHT SOURCE SURFACES IN GREEN AND TARGET SURFACES IN RED, MAKE ALL LINKING SURFACES YELLOW

INPUTS

- VOLUME LIST

OUTPUTS

- NONE, A COLOR CODEDED VIS OF THE VOLUMES LISTED

printNamesAndVols (goto python line: 1505) :

PRINT NAME AND VOLUME OF ALL VOLUMES

INPUTS

- NONE

OUTPUTS

- PRINTS TO THE CUBIT CONSOLE

printNameAndGroupVols (goto python line: 1519) :

GET THE TOTAL VOLUME OF VOLUMES BY GROUP

INPUTS

- NONE

OUTPUTS

- PRINTS TO THE CUBIT CONSOLE

printBlockPics (goto python line: 1534) :

GENERATES SNAPSHOTS OF ALL BLOCKS FROM A SPECIFIED DEFAULT VIEW, WILL GENERATE PICTURE WITH GIVEN QUALITY METRIC IF THE METRIC IS SPECIFIED (CURRENTLY SETUP FOR FREE MESH GEOMS)

INPUTS

- THE PATH NAME, INCLUDING THE BASE NAME OF THE IMAGE TO BE GENERATED
- THE NAME OF THE VIEW TO USE (TOP, BOTTOM, LEFT, RIGHT, FRONT, BACK, ISO)

OPTIONAL

- A QUALITY METRIC TO USE WHEN TAKING A SNAPSHOT (I.E. SCALED JACOBIAN); IF NONE, THEN JUST SHOW THE MESH...
- FREEMESH - FLAG TO INDICATE THE ELEMENTS IN THE BLOCK HAVE NO OWNING VOLUME

OUTPUTS

- A SERIES OF SNAPSHOTS OF EACH MESH BLOCK

showParametricLabels (goto python line: 1575) :

USE THE CUBIT LABEL FUNCTIONALITY IN ORDER TO GENERATE A CURVE LENGTH DISPLAY *** WARNING THIS WILL RENAME THE CURVES IN THE GIVEN SURFACES ***

INPUTS

- A SURFACE ID LIST STRING

OUTPUTS

- TURNS ON PARAMETRIC LABELS, TO TURN OFF TYPE "LABEL OFF"

showMaterialColors (goto python line: 1590) :

COLORS BLOCKS BASED ON THE ASSIGNED MATERIAL, BLACK INDICATES THE BLOCK IS NOT ASSIGNED

INPUTS

- NONE

OUTPUTS

- NONE

renameVols (goto python line: 1614) :

APPEND OR PREFIXES NAMES OF VOLS WITH suf/pre; REPLACES repl with replw STRING; IF LEFT BLANK NOTHING WILL HAPPEN
BLANK = '' *** CUBIT HAS ISSUES GROUPING ITEMS WHEN THE VOLUME NAMES CONTAIN SPACES OR START WITH A NUMBER ***

INPUTS

- NONE REQUIRED, DOES NOTHING IF NOTHING IS SUPPLIED

OPTIONAL

- TOTAL RENAME STRING, TO WIPE THE EXISTING NAME CLEAN IF DESIRED, FROM THE SELECTED VOLUMES
- PREFIX STRING, TO APPEND A PREFIX TO THE SELECTED VOLUME NAMES
- SUFFIX STRING, TO APPEND A SUFFIX TO THE SELECTED VOLUME NAMES
- REPLACE STRING, REPLACE ANY MATCHING STRING WITH THE REPLACE WITH STRING (*BE CAREFUL, THIS IS PARTIAL STRING MATCHING*)
- REPLACE WITH STRING, THE REPLACEMENT STRING
- A LIST OF VOLUMES AS A CUBIT LIST STRING (ALL IS THE DEFAULT IF NONE IS SUPPLIED)

autoGroupByName (goto python line: 1647) :

AUTOMATICALLY GROUPS ALL VOLUMES BY NAME GIVES GROUPED ENTITIES THE SAME COLOR

INPUTS

- NONE

OPTIONAL

- A BOOLEAN FLAG FOR COLORING LIKE NAMED VOLUMES THE SAME COLOR

OUTPUTS

- NONE, GROUPED VOLUMES BASED ON BASENAME

getDistance (goto python line: 1703) :

RETURN THE DISTANCE BETWEEN TWO CUBIT OBJECTS

INPUTS

- GEOMETRY OBJECT TYPE 1
- GEOMETRY OBJECT ID 1
- GEOMETRY OBJECT TYPE 2
- GEOMETRY OBJECT ID 2

OPTIONAL

- METHOD - THE METHOD FOR PROJECTION (DEFAULTS TO CENTER OF THE CUBIT OBJECT)

OUTPUTS

- THE MAGNITUDE OF THE DISTANCE AND THE VECTOR DISTANCE

getMassProperties (goto python line: 1832) :

CALCULATE THE TOTAL MASS PROPERTIES OF AN OBJECT GIVEN THAT A MATERIAL AND DENSITY HAVE BEEN ASSIGNED TO THE BLOCK OF THE VOLUMES

INPUTS

- VOLLIST - A CUBIT STRING LIST OF VOLUMES TO CALCULATE THE MASS PROPERTIES FROM

OPTIONAL

- SCALEFACTOR - SCALE THE VOLUME BY THE FACTOR GIVEN

OUTPUTS

- THE TOTAL MASS PROPERTIES OF THE VOLUME LIST AS A LIST - [MASS, CGX, CGY, CGZ]
- MassPropsReport.csv - A MASS PROPERTIES REPORT IN THE CUBIT WORKING DIRECTORY

getMassPropertiesExo (goto python line: 1871) :

CALCULATE THE TOTAL MASS PROPERTIES OF AN OBJECT GIVEN THAT A MATERIAL AND DENSITY HAVE BEEN ASSIGNED TO THE BLOCK OF THE VOLUMES

INPUTS

- BLOCKLIST - A CUBIT LIST STRING OF BLOCKS TO CALCULATE THE MASS PROPERTIES FROM

OPTIONAL

- SCALEFACTOR - SCALE THE VOLUME BY THE FACTOR GIVEN

OUTPUTS

- THE TOTAL MASS PROPERTIES OF THE VOLUME LIST AS A DICTIONARY - [MASS, CGX, CGY, CGZ]

getElemSliceMassProperties (goto python line: 1921) :

GET THE TOTAL VOLUME AND MASS PROPERTIES OF THE GIVEN ELEMENTS ON A TOTAL AND PER BLOCK BASIS *** NOTE THIS REQUIRES BLOCK MATERIAL ASSIGNMENTS ***

INPUTS

- ELEMLIST - A CUBIT LIST STRING OF ELEMENT IDS

OPTIONAL

- ETYPE - THE TYPE OF ELEMENTS GIVEN IN THE ELEMENT LIST (DEFAULT IS HEX)

OUTPUTS

- THE TOTAL MASS, TOTAL VOLUME, AND PER BLOCK VOLUME AND MASS CAPTURED

9 PyCubed_SelectFuncs Overview

selMore (goto python line: 28) :

SELECT ADJACENT FACES TO THE ORIGINAL SELECTION, REMOVE THOSE IN THE REMOVAL CATEGORY

INPUTS

- ORIGINAL - A LIST OF CUBIT IDS AS A STRING (DEFAULTS TO THE CURRENT SELECTION)

OPTIONAL

- REMFACES - A LIST OF CUBIT IDS AS A STRING FOR DESELECTION (DEFAULTS TO NONE)
- SPREADTYPE - PICK HOW TO SELECT ADJOINING GEOMETRY (VERTEX OR CURVE; DEFAULT IS CURVE)

OUTPUTS

- MOREORLESS - THE LIST OF THE DESIRED ADJACENT SURFACES AND THE ORIGINAL SURFACES IF NOT OMMITED

selMoreRecursive (goto python line: 53) :

A RECURSIVE VERSION OF SELECT MORE WHICH STOPS AT THE SELECTED STOPPING SURFACES

INPUTS

- SEEDIDS - THE SURFACES TO USE AS A SEED SURFACES AS A CUBIT LIST STRING

OPTIONAL

- STOPIDS - THE SURFACES TO USE TO STOP THE SPREAD OF THE SELECTION AS A CUBIT LIST STRING

OUTPUTS

- SURFLIST - THE SURFACES THAT ARE SELECTED

selLinkedFlat (goto python line: 74) :

TRYING TO MIMIC BLENDER'S SELECT LINKED FLAT FACES ROUTINE

INPUTS

- A SEED SURFACE ID AS AN INTEGER
- THE ANGLE THAT DETERMINES IF THE SELECTION ADVANCES

OPTIONAL

- THE VOLUMES TO COMPUTE ON, THIS IS REQUIRED IF MORE THAN ONE VOLUME EXISTS

OUTPUTS

- THE LIST OF THE SURFACE IDS THAT ARE SMOOTH ENOUGH
- A PYTHON DICTIONARY GIVING THE EDGE SHARPNESS ANGLE FOR EACH EDGE

selLinkedFlatExo (goto python line: 138) :

SELECT THE LINKED FLAT FACES OR TRIS ASSOCIATED WITH A SEED FACE

INPUTS

- SEEDFACE - A SEED FACE FOR THE ADVANCING FRONT

OPTIONAL

- ANGLE - A BREAK ANGLE FOR THE FRONT (DEFAULT IS 15.0 DEGREES)
- SEEDTYPE - THE TYPE OF ELEMENT SELECTED (DEFAULT IS QUAD)

OUTPUTS

- CONFACES - A LIST OF THE CONNECTED FACE IDS BASED ON THE SEED FACE AND GIVEN ANGLE

groupLinkedFlatExo (goto python line: 182) :

GROUP ALL SELECTED MESH ELEMENTS WITH LINKED FLAT FACES BASED ON THE BREAK ANGLE SPECIFIED

INPUTS

- ANGLE - THE BREAK ANGLE TO USE IN DEGREES AS A FLOAT VALUE

OUTPUTS

- CONGROUPS - THE LIST CONTAINING THE CONNECTED FACE GROUPS AS GROUPS AS LISTS OF FACE IDS
- CUBIT GROUPS NAMED FlatFaces_X SHOWING THE CONNECTED REGIONS OF THE MESH

selNormSurf (goto python line: 231) :

DESELECTS OR SELECTS FACES BASED ON NORMAL DIR GIVE, TOL PROVIDES A TOLERANCE TO THE NORMAL VECTOR

INPUTS

- A NORMAL DIRECTION UNIT VECTOR AS A LIST
- AN ANGLE TOLERANCE AS A FLOAT IN RADIANS
- A LIST OF SURFACES AS A CUBIT ID STRING

OUTPUTS

- THE LIST OF SURFACE IDS WHICH ARE NORMAL TO THE GIVEN DIRECTION, WITHIN TOLERANCE

selBySType (goto python line: 268) :

FILTER BY SURFACE TYPE

PLANE, CONE, SPLINE, TORUS

INPUTS

- A LIST OF CUBIT SURFACE IDS (AS A STRING)
- THE TYPE OF SURFACE TO GATHER - PLANE, CONE, SPLINE, TORUS

OUTPUTS

- A LIST OF THE SURFACES THAT MEET THE

selByVolume (goto python line: 288) :

RETURNS VOL IDS BY DOING A VOLUMETRIC CALCULATION, SPECIFY CRITERION crit TO MEET- LT, GT, EQ, LE, GE, NE (OLD FORTRAN)

INPUTS

- A LIST OF VOLUME IDS AS A CUBIT STRING (GETS PARSED)
- THE CRITERIA TO SOLVE FOR, GREATER THAN , LESS THAN, ETC AS A STRING 'GT', 'LT', 'EQ', 'LE', 'GE'
- THE VALUE FOR COMPARISON

OPTIONAL

- THE TOLERANCE USED FOR EQUAL TO

OUTPUTS

- A LIST OF VOLUMES THAT PASS THE FILTER

selByArea (goto python line: 325) :

RETURNS SURF IDS BY DOING A VOLUMETRIC CALCULATION, SPECIFY CRITERION CRIT TO MEET- LT, GT, EQ, LE, GE, NE (OLD FORTRAN)

INPUTS

- A LIST OF SURFACE IDS AS A CUBIT STRING (GETS PARSED)
- THE CRITERIA TO SOLVE FOR, GREATER THAN , LESS THAN, ETC AS A STRING 'GT', 'LT', 'EQ', 'LE', 'GE'
- THE VALUE FOR COMPARISON

OPTIONAL

- THE TOLERANCE USED FOR EQUAL TO

OUTPUTS

- A LIST OF SURFACES THAT PASS THE FILTER

selByLength (goto python line: 360) :

RETURNS CURVE IDS BY DOING A LENGTH CALCULATION, SPECIFY CRITERION crit TO MEET - LT, GT, EQ, LE, GE, NE (OLD FORTRAN)

INPUTS

- A LIST OF CURVE IDS AS A CUBIT STRING (GETS PARSED)
- THE CRITERIA TO SOLVE FOR, GREATER THAN , LESS THAN, ETC AS A STRING 'GT', 'LT', 'EQ', 'LE', 'GE'
- THE VALUE FOR COMPARISON

OPTIONAL

- THE TOLERANCE USED FOR EQUAL TO

OUTPUTS

- A LIST OF CURVES THAT PASS THE FILTER

selByCType (goto python line: 395) :

RETURNS CURVE IDS BY TYPE - LINE, ARC, SPLINE

INPUTS

- A LIST OF CURVE IDS AS A CUBIT STRING (GETS PARSED)
- THE TYPE OF CURVE(S) TO GATHER

OUTPUTS

- A LIST OF CURVES THAT ARE OF THE TYPE SOUGHT
- A LIST OF ALL CURVES PARSED, AND THEIR TYPES

selBySSC (goto python line: 413) :

SELECTS SURFACES USING THE SELECTED SURFACE CHILDREN; GATHER SURFACES WITH N NUMBER OF CURVES SELECTED GOOD TO PAIR WITH POLYGONAL AND CURVE LENGTH SELECTION

INPUTS

- A LIST OF CURVE IDS AS A CUBIT STRING

OPTIONAL

- THE NUMBER OF CURVES OF THE PARENT SURFACE THAT INDICATE THE SURFACE NEEDS SELECTION AS AN INTEGER
- A BOOLEAN TO TURN OFF EXACT COUNT, AND USE ANYTHING GREATER THAN AS WELL

OUTPUTS

- A LIST OF THE SURFACES THAT CONTAIN N SELECTED CURVES

selPolygonSurfs (goto python line: 441) :

SELECT SURFACES WITH N CHILD CURVES

INPUTS

- A LIST OF SURFACES AS A CUBIT ID STRING LIST

OUTPUTS

- THE SURFACES IN THAT LIST CONTAINING THAT NUMBER OF CURVES

selMergedVols (goto python line: 459) :

RETURN THE VOLUMES MERGED TO THE SELECTION

INPUTS

- A SEED VOLUME ID AS AN INTEGER OR CUBIT LIST STRING

OUTPUTS

- THE VOLUMES THAT ARE MERGED TO THE SEED VOLUME AS A PYTHON LIST OF CUBIT INTEGER IDS

selVertCurves (goto python line: 480) :

SELECT ONLY THE CURVE WHICH CONTAINS THE TWO VERTS

INPUTS

- 2 VERTEX IDS

OUTPUTS

- THE CURVE SEGMENT CREATED BY THESE POINTS

selCurveVerts (goto python line: 495) :

SELECT THE VERTS ASSOCIATED WITH THE PROVIDED CURVE

INPUTS

- THE CURVE ID AS A CUBIT STRING OR INTEGER

OUTPUTS

- THE VERTICIES OR ENDPOINTS OF THE CURVE ID

selMultiloopSurfs (goto python line: 508) :

RETURNS THE SURFACES WHICH CONTAIN MULTIPLE CURVE LOOPS, I.E. THOSE WITH THROUGH HOLES AND UNSPLIT CONICAL SURFACES

INPUTS

- SURFLIST - A SURFACE LIST AS A CUBIT LIST STRING

OUTPUTS

- MLSURFS - A LIST OF THE SURFACE IDS WHICH ARE MULTILOOP

selBlendSurfs (goto python line: 526) :

SELECT THE BLEND SURFACES (ROUNDS) WHICH ARE SMALLER THAN THE MAX RAD

INPUTS

- VOLLIST - A LIST OF VOLUMES TO OPERATE ON AS A CUBIT STRING
- MAX_RAD - THE MAXIMUM RADIUS TO SELECT AS A FLOAT (CAPTURES ANYTHING SMALLER)

OPTIONAL

- MIN_RAD - THE MAXIMUM RADIUS TO SELECT AS A FLOAT (DEFAULT IS 0.0)
- CTOL - THE TOLERANCE USED TO CAPTURE NEAR CURVATURES (DEFAULT IS 0.05)
- LSCC - LINKED SURFACE CURVATURE CHECK, CHECK IF THE LINKED SURFACES ARE WITHIN THE CURVATURE BOUNDS NEEDED (MAX-AVG HARDCODED)

OUTPUTS

- THE LIST OF SURFACES THAT ARE ROUNDS/BLENDS BELOW THE SPECIFIED MAXIMUM RADIUS

selCurveChain (goto python line: 596) :

SELECT A CHAIN OF CURVES THAT EXIST WITHIN THE GIVEN ANGLE *** NOTE

THE CHAIN ANGLE IS BASED ONLY ON THE END VERTS OF THE CURVE ***

INPUTS

- A SEED CURVE ID AS AN INTEGER

OPTIONAL

- THE MAXIMUM ANGLE BETWEEN VERT SEGMENTS DESIRED

OUTPUTS

- THE CHAIN OF CURVES THAT PROPAGATES FROM THE SEED CURVE BASED ON THE CUTOFF ANGLE

selExInTeriorSurfs (goto python line: 634) :

GATHER THE EXTERIOR SURFACES GIVEN A LIST OF SURFACES, EXTERIOR SURFACES ARE CONSIDERED THOSE THAT HAVE FREE EDGES IN THE LIST

INPUTS

- SURFLIST - A CUBIT LIST STRING OF SURFACES

OPTIONAL

- NONE

OUTPUTS

- EXT_INT - A LIST OF LISTS CONTAINING EXTERIOR AND INTERIOR SURFACES AS [[EXT_LIST], [INT_LIST]]

selBoxBound (goto python line: 664) :

SELECT AND RETURN GEOMETRY IN THE BOUNDING BOX SUPPLIED

INPUTS

- THE TYPE OF GEOMETRY TO BE RETURNED
- A LIST OF THE ITEMS TO BE LOOKED AT AS A STRING
- 3 LISTS OF LENGTH 2 DENOTING THE X,Y,AND Z [MIN, MAX] *** LEAVE AS NONE IF YOU WISH TO INCLUDE ALL ***

OUTPUTS

- A LIST OF THE GEOMETRY ENTITY IDS WITHIN THE GIVEN CUBE

selBoxBoundExo (goto python line: 714) :

SELECT AND RETURN EXODUS ENTITIES IN THE BOUNDING BOX SUPPLIED

INPUTS

- EXOTYPE - THE TYPE OF EXODUS ENTITIES TO BE RETURNED
- EXOLIST - A LIST OF THE ITEMS TO BE LOOKED AT AS A STRING
- XS, YS, ZS - 3 LISTS OF LENGTH 2 DENOTING THE X, Y, AND Z [MIN, MAX] *** LEAVE AS NONE IF YOU WISH TO INCLUDE ALL ***

OUTPUTS

- A LIST OF THE EXODUS ENTITY IDS WITHIN THE GIVEN CUBE

selSphericalBound (goto python line: 758) :

SELECT AND RETURN THE GEOMETRY WITHIN A SPHERICAL BUBBLE FROM THE SPECIFIED CENTER POINT

INPUTS

- GEOMTYPE - THE TYPE OF GEOMETRY TO CONSIDER AS A STRING
- GEOMLIST - A LIST OF THE GEOMETRY ENTITIES' IDS TO CONSIDER AS A CUBIT STRING
- CENTERPOINT - THE SPECIFIED CENTER POINT AS A LIST [X,Y,Z]
- RADIUS - THE SPHERICAL RADIUS AS A FLOAT

OPTIONAL

- SAMPLES - THE NUMBER OF U OR UV SAMPLES TO TAKE, MORE IS SLOWER BUT MORE ACCURATE

OUTPUTS

- BUBS - A LIST OF THE GEOMETRY ENTITY IDS WITHIN THE GIVEN BUBBLE
- ALL_POINTS - A LIST OF ALL GATHERED POINTS USED IN THE CALCULATION
- GCP - A GLOBAL CENTER POINT BASED ON THE POINTS IN THE SELECTION

selSphericalBoundExo (goto python line: 819) :

SELECT AND RETURN THE EXODUS ENTITY WITHIN A SPHERICAL BUBBLE FROM THE SPECIFIED CENTER POINT, ORDERED WRT SMALLEST RADIUS

INPUTS

- EXOTYPE - THE TYPE OF MESH ENTITY TO CONSIDER AS A STRING
- EXOLIST - A LIST OF THE MESH ENTITIES' IDS TO CONSIDER AS A CUBIT STRING
- CENTERPOINT - THE SPECIFIED CENTER POINT AS A LIST [X,Y,Z]
- RADIUS - THE SPHERICAL RADIUS AS A FLOAT

OUTPUTS

- A LIST OF THE MESH ENTITY IDS WITHIN THE GIVEN BUBBLE

selVertAt (goto python line: 849) :

SELECT VERTS AT THE GIVEN LOCATION

INPUTS

- LOCVECTOR - THE LOCATION NEAR WHERE TO SEARCH FOR A VERTEX AS A PYTHON LIST

OPTIONAL

- VERTLIST - A LIST OF DESIRED ENTITIES TO CHECK AS A CUBIT STRING
- TOL - A TOLERANCE FOR FINDING THE VERTEX

OUTPUTS

- THE VERTEX ID(S) IN THE VECINITY OF THE LOCATION SPECIFIED

selProjectedNodes (goto python line: 868) :

PROJECT THE GIVEN NODES ONTO A SURFACE AND FIND THOSE NODES WITHIN SPECIFIED ANGLE AND/OR DISTANCE TOLERANCES

INPUTS

- SURFIDS - A LIST OF SURFACES PROVIDED AS A CUBIT LIST STRING
- NODEIDS - A LIST OF NODE IDS AS A CUBIT LIST STRING

OPTIONAL

- TOL - A DISTANCE TOLERANCE FOR SELECTING
- ANGLE_TOL - A MAXIMUM ANGLE FROM THE NORMAL OF THE SURFACE ALLOWED

OUTPUTS

- NEARNODES - A LIST OF THE NODES WITHIN TOLERANCE

selElemByNode (goto python line: 912) :

SELECT AN ELEMENT USING A NODELIST OF CHILD NODES AND THE MINIMUM NUMBER OF NODES THAT NEED TO BE PRESENT IN THE NODELIST

INPUTS

- ETYPE - THE ELEMENT TYPE TO SEARCH FOR
- NODELIST - A CUBIT LIST STRING OF NODE IDS TO USE IN THE SEARCH

OPTIONAL

- ELIST - A CUBIT LIST STRING OF ELEMENT IDS OF ETYPE TO SEARCH THROUGH (ALL BY DEFAULT IF NONE ARE SET)
- MINNODECOUNT - THE MINIMUM NODE COUNT TO USE IN THE SEARCH (1 BY DEFAULT)

OUTPUTS

- A LIST OF THE ELEMENT IDS THAT MATCH THE QUERY

selProjNodeNorm (goto python line: 935) :

PROJECT NODES FROM ONE VOLUME TO ANOTHER USING A NORMAL DIRECTION (ORTHOGRAPHIC PROJECTION)

INPUTS

- NODELIST1 - A CUBIT LIST STRING OF NODES OF MAIN VOLUME TO PROJECT
- NODELIST2 - A CUBIT LIST STRING OF NODES OF THE VOLUME TO BE PROJECTED TO
- NORMALDIR - NORMAL DIRECTION FOR PROJECTION AS A LIST [X, Y, Z]

OPTIONAL

- NORMALTOL - ANGLE TOLERANCE FOR THE NORMAL PROJECTION IN DEGREES

OUTPUTS

- A NODELIST OF THE NODES MEETING THE GIVEN CRITERION

selProjFaceElemNorm (goto python line: 964) :

PROJECT FACE CENTERS AND NORMALS TO INTERSECT ELEMENT CENTERS GIVEN A NORMAL DIRECTION

INPUTS

- FACETYPE - THE TYPE OF FACE QUAD/FACE OR TRI (DEFAULT IS FACE)
- FACELIST - A CUBIT LIST STRING OF FACE IDS TO USE FOR PROJECTION
- ELEMTYPE - THE TYPE OF ELEMENT TO SEARCH HEX OR TET (DEFAULT IS HEX)
- ELEMLIST - A CUBIT LIST STRING OF ELEMENT IDS TO USE FOR PROJECTION
- NORMDIR - A NORMAL DIRECTION TO PROJECT THE FACES IN

OPTIONAL

- NORMTOL - A TOLERANCE BETWEEN THE FACE CENTER AND THE ELEMENT CENTER WRT THE PROJECTED NORMAL (DEFAULT IS 1.0 DEGREE)
- MASSINT - A BOOLEAN SWITCH TO CALCULATE THE MASS INTERSECTION (DEFAULT IS FALSE)

OUTPUTS

- AN ELEMENT LIST OF THE ELEMENTS MEETING THE GIVEN CRITERION, THE VOLUME INTERSECTED

selConMesh (goto python line: 999) :

SELECT ALL MESH NODES CONNECTED TO THE SEED NODE

INPUTS

- A SEED NODE ID AS AN INTEGER

OUTPUTS

- ALL THE SUBSEQUENT NODE IDS THAT ARE CONNECTED TO THE SEED NODE

groupConMesh (goto python line: 1025) :

GROUP ALL SELECTED MESH ELEMENTS WITH CONNECTED NODES IN THE GIVEN BLOCK

INPUTS

- NONE - THIS IS AN ADVANCING FRONT ALGORITHM, TIMES FOR LARGER MESH ENTITIES MIGHT BECOME LARGE

OUTPUTS

- GROUPS NAMES CONGROUP_X SHOWING THE CONNECTED REGIONS OF THE MESH

selHexInVol (goto python line: 1050) :

GET THE HEX ITEMS THAT OVERLAP WITH UNMESHED VOLUMES IN THE MODEL

INPUTS

- NONE REQUIRED

OPTIONAL

- VOLUMELIST - A CUBIT LIST STRING OF VOLUME IDS (DEFAULTS TO ALL UNMESHED VOLUMES IF NOT SPECIFIED)
- HEXLIST - A CUBIT LIST STRING OF HEX IDS (DEFAULTS TO ALL IF NOT SPECIFIED)
- CHECKTYPE - A PROJECTION METHOD FOR FINDING THE OVERLAP
- LOOSE TYPE USES ONLY THE CENTER POINT, TIGHT USES HEX NODES AND A COLLISION NODE COUNT
- ONC - A NUMBER OF NODES TO CONSIDER AN ELEMENT FULLY COLLIDING DURING TIGHT BOUNDING SELECTION, THIS ALLOWS FOR BETTER CONTROL OF THE SELECTION

OUTPUTS

- COLLIDING - A LIST OF THE HEX IDS THAT ARE CONTAINED INSIDE THE UNMESHED VOLUMES

selEdgeLineEnds (goto python line: 1090) :

GATHER THE ENDPOINTS OF A CONTINUOUS EDGE LINE *** THIS ASSUMES AN ACTUAL EDGE LINE IS SELECTED AND WILL RETURN MULTIPLE PAIRS IF NOT CONTINUOUS ***

INPUTS

- THE EDGE LONE LIST AS A LIST OF EDGE IDS

OUTPUTS

- THE START AND END NODES AS A LIST (MULTIPLE PAIRS IF THE EDGE LIST PROVIDED HAS GAPS)

selEdgeLoop (goto python line: 1108) :

SELECT A CONTINUOUS LOOP OF EDGES BASED ON EDGE NORMALS, FOR THE OWNING VOLUME *** PICKING EDGES IN THE MIDDLE OF A LOOP SPEEDS THIS UP ***

INPUTS

- A SEED EDGE ID TO START THE LOOP (INTEGER OR CUBIT STRING)

OPTIONAL

- A MAXIMUM DEVIATION ANGLE FOR CORNERING IN DEGREES AS A FLOAT

OUTPUTS

- A LIST OF THE EDGE IDS IN THE LOOP (THE EDGES WILL BE SELECTED)

10 PyCubed_MeshMods Overview

compressBlocks (goto python line: 26) :

COMPRESS BLOCK IDS

INPUTS

- NONE

OUTPUTS

- NONE, RESTRUCTURED BLOCK IDS

compressSidesets (goto python line: 50) :

COMPRESS SIDASET IDS

INPUTS

- NONE

OUTPUTS

- NONE, RESTRUCTURED SIDASET IDS

blockByGroups (goto python line: 74) :

GENERATES BLOCKS BASED ON GROUPS (BEST USED WITH AUTOGROUPBYNAME FUNCTION)

INPUTS

- NONE

OPTIONAL

- AUTOGROUPED BOOLEAN (REMOVED THE G PREFIX WHEN USING THE AUTOGROUP BY NAME FUNCTION)

OUTPUTS

- NONE, BLOCKS WHICH REPRESENT GROUPS

combineBlocks (goto python line: 95) :

COMBINE BLOCK BA WITH BLOCK BB, RETAIN BLOCK BA, REMOVE BLOCK BB IF REMOVE IS TRUE

INPUTS

- BLOCK ID A AS AN INTEGER
 - BLOCK ID B AS AN INTEGER
- OPTIONAL
- REMOVE BLOCK B (FALSE AS DEFAULT), IF TRUE REMOVES THE SECOND BLOCK AFTER COMBINATION

splitBlock (goto python line: 113) :

SPLIT A BLOCK OR BLOCKS, COMBINING THE VOLUMES, SURFACES, CURVES, EDGES, FACES, OR ELEMS INTO A NEW BLOCK

INPUTS

- ENTITYTYPE - THE TYPE OF ENTITY TO SEPARATE INTO THE NEW BLOCK
- ENTITYLIST - A LIST OF ENTITIES TO SEPARATE FROM THEIR CURRENT BLOCKS AS A CUBIT LIST STRING

OUTPUTS

- THE NEW BLOCK ID FORMED

combineSidesets (goto python line: 130) :

COMBINE SIDESETS MERGING WITH THE SIDESSET PAIR WITH THE LOWEST ID

INPUTS

- SIDESSETLIST - A LIST OF SIDESETS TO COMBINE

OUTPUTS

- ZIPPEDALL - A LIST OF THE CONSOLIDATED ITEMS

vols2SameBlock (goto python line: 162) :

MOVE SELECTED VOLUMES TO THE SAME BLOCK MOVES INTO THE FIRST SELECTED ITEMS BLOCK

INPUTS

- A LIST OF VOLUMES AS A CUBIT LIST STRING TO COMBINE INTO A SINGLE BLOCK ** THIS WILL MOVE ALL SELECTED VOLUMES INTO THE BLOCK COMMON TO THE FIRST SELECTED VOLUME ID**

OUTPUTS

- NONE, VOLUMES MOVED INTO DIFFERENT BLOCKS

sweepVolsInDir (goto python line: 180) :

AUTOMATICALLY APPLY SWEEP PARAMETERS TO ALL VOLUMES GIVEN, IN ORDER TO SWEEP THEM ALL IN THE NORMAL DIRECTION GIVEN THIS METHOD WORKS FOR MULTI-SWEEPS

INPUTS

- A NORMAL DIRECTION AS A PYTHON LIST OF FLOATING NUMBERS ([X,Y,Z])
- A DEVIATION TOLERANCE FOR THIS NORMAL DIRECTION AS A FLOAT IN RADIANS

OPTIONAL

- A LIST OF VOLUME IDS AS A CUBIT STRING

OUTPUTS

- NONE, APPLIES THE SWEEP SCHEME IN THE DIRECTION

nodes2Surface (goto python line: 244) :

MOVE NODES TO THE SELECTED SURFACE (LESS POWERFUL THAN NODES2 SURFS, BUT WORKS ON FREE MESH AS WELL)

INPUTS

- SURFACE - A SURFACE TO MOVE NODES TO
- NODELIST - A LIST OF NODES AS A CUBIT LIST STRING TO MOVE

OPTIONAL

- SMOOTH SURFS - A LIST OF SURFACES TO SMOOTH AFTER THE NODES ARE MOVED *** CAREFUL HERE, THIS CAN WARP MESH SUBSTANTIALLY ***
- USE NORMAL - THIS CHECKS THE NORMAL DIRECTION OF THE NODE MOVEMENT TO THE SPECIFIED SURFACE, IF NOT WITHIN THE TOLERANCE THE NODE IS NOT MOVED
- TOL - THE ANGLE TOLERANCE FOR DETERMINING IF THE MOVEMENT TO THE SURFACE SHOULD BE ALLOWED

OUTPUTS

- NONE, NODES ARE MOVED TO THE DESIRED SURFACE

nodes2Surfs (goto python line: 283) :

MOVE NODES IN A SURFACE LIST BY PROJECTING THEM TO THE CLOSEST POINT ON ANOTHER SURFACE LIST

INPUTS

- A SURFACE LIST TO PROJECT TO AS A CUBIT LIST STRING
- A LIST OF MESHED SURFACES AS A CUBIT LIST STRING, FOR NODAL CORRELATION *** THE NODES WILL BE PROJECTED OFF OF THE GEOMETRY TO THE GIVEN GEOMETRY ***

OPTIONAL

- THE TOLERANCE FOR PROJECTION AS A FLOAT (WILL NOT PROJECT PAST THE TOLERANCE CUTOFF)

OUTPUTS

- NONE, MOVEMENT OF NODES

offsetNodes (goto python line: 328) :

PUSH SURFACE NODES IN THE NORMAL DIRECTION BY THE OFFSET AMMOUNT

INPUTS

- MESHEDSURFS - THE MESHED SURFACES AS A CUBIT LIST STRING
- OFFSET - THE VALUE THAT WE SHOULD TRY AND OFFSET BY

OUTPUTS

- NONE - THE PUSHED NODES

nodeset2Nodeset (goto python line: 384) :

A MANUAL MOVING OF NODES TO BARBARICALLY FORCE MERGE NODES TOGETHER *** THIS ASSUMES A 1
1 CORRELATION OF NODES ***

INPUTS

- A MASTER NODESET
- A SLAVE NODESET (TO BE MOVED TO THE MASTER)

OUTPUTS

- NONE, MOVEMENT OF NODES AS REQUIRED

sidesetFromNodes (goto python line: 413) :

CONVERT A NODESET TO A SIDASET * OMITTING THE FACES WIHTOUT ALL 4 OR 3 NODES SELECTED IN THE SIDASET

INPUTS

- NODELIST - A CUBIT LIST STRING OF THE NODES TO BE USED

OPTIONAL

- NCOUNT - A COUNT OF NODES TO REMOVE FOR CONSIDERATION (I.E. 1 ALLOWS FOR 3 NODES TO COUNT A FACE OR 2 TO COUNT A TRI)

OUTPUTS

- FACELIST - A LIST OF FACES OR TRIS THAT ARE WITHIN THE NODESET GIVEN

moveNodesInNorm (goto python line: 456) :

MOVE NODES IN THE DIRECTION OF A NORMAL VECTOR BY SOME DISTANCE, SUPPLY A NORMAL OR SURFACE, NODE LIST,
AND DISTANCE ** THIS FUNCTION CAN ALSO BE FOUND IN THE CUBIT GUI **

INPUTS

- A NORMAL DIRECTION EITHER AS A PYTHON LIST OF FLOATS OR GIVEN A SURFACE ID
- A NODE LIST AS A CUBIT LIST STRING
- THE MAGNITUDE OF MOVEMENT IN THE PROVIDED NORMAL DIRECTION AS A FLOAT

OUTPUTS

- NONE, MOVED NODES

shiftNodeSkew (goto python line: 484) :

SHIFT NODES BETWEEN CURVES IN ORDER TO MINIMIZE SKEWING SURFACE, THIS WORKS WHEN THE VOLUME IS FULLY MESHED
*** THIS MAY CAUSE PINCH POINTS IF OVERCONSTRAINED BY VERTS ***

INPUTS

- SURFACE - THE SURFACE TO BE OPERATED ON
- CURVE LIST - CURVES WITH SKEWED ELEMENTS TO BE STRAIGHTENED, THIS WILL ATTEMPT TO MAKE THE PROJECTED
EDGE ANGLE NEAREST TO 90 DEGREES FOR THE SELECTED CURVES
- ITERATIONS - THE NUMBER OF SMOOTHING PASSES TO MAKE (BUMPING ONE EDGE OVER MAY KNOCK THE OTHERS
OUT OF ALIGNMENT)

OPTIONAL

- SMOOTH - A BOOLEAN TO TURN ON OR OFF A SMOOTHING ALGORITHM
- SCHEME - THE DESIRED SMOOTHING SCHEME AS A STRING NAME (DEFAULTS TO WINSLOW)

OUTPUTS

- NONE, THE NODES SHIFTED TO AS NEAR TO NORMAL FROM THE SELECTED SURFACE AS POSSIBLE

shiftLinkingSurfSkew (goto python line: 564) :

SHIFT SKEW IN LINKING SURFACES BY CHECKING AND ATTEMPTING TO AVERAGE NODE LOCATIONS IN THROUGH THICKNESS
AREAS

INPUTS

- SURFLIST - A CUBIT LIST STRING OF ORTHOGONAL OR SWEEPED SURFACES THAT HAVE LINKING SURFACE SKEW

OPTIONAL

- W - A DISTANCE MULTIPLIER TO ENSURE THAT WE DONT TANGLE THE MESH (VALUES BELOW 1 SLOW THE NODE
MOVEMENT, VALUES GREATER THAN 1 SPEED IT UP) TAKEOFFTOL - TAKEOFFTOL - A TOLERANCE ON THE TAKEOFF
ANGLE DERIVED FROM THE ORTHOGONAL SURFACE ORIGINALLY GIVEN

OUTPUTS

- NONE, MODIFIED MESH...

nodeAvgSS (goto python line: 635) :

A LOCAL NODAL SMOOTHING SCHEME BASED ON AVERAGE POSITION OF NEIGHBOORING NODES *** THIS ALGORITHM WILL ROUND HARD CORNERS UNLESS BOUNDARY NODES ARE IGNORED ***

INPUTS

- A LIST OF NODES AS A CUBIT LIST STRING

OPTIONAL

- A LIST OF NODES TO IGNORE, THIS CAN BE USEFUL TO KEEP HARD EDGES

OUTPUTS

- NONE, MOVED NODES

nodeAvgVS (goto python line: 673) :

A VOLUMETRIC LOCAL NODAL SMOOTHING SCHEME BASED ON AVERAGE POSITION OF NEIGHBOORING NODES *** THIS ALGORITHM MAY BE DESTRUCTIVE ***

INPUTS

- A LIST OF NODES AS A CUBIT LIST STRING

OPTIONAL

- A LIST OF NODES TO IGNORE, THIS CAN BE USEFUL TO KEEP CURRENT INTERNAL VOLUME MESH

OUTPUTS

- NONE, MOVED NODES

autoSplitPoorHexes (goto python line: 717) :

SPLIT ELEMENTS THAT ARE OF POOR QUALITY DUE TO AN OBTUSE ANGLE VERTEX SPLIT (FOR INSTANCE ON MAPPED CIRCLES)

INPUTS

- SURFID - THE SOURCE OR TARGET SURFACE CONTAINING THE POOR ELEMENT

OPTIONAL

- HEXQUAL - THE QUALITY CUTOFF TO USE IN TERMS OF SCALED JACOBIAN (DEFAULT IS 0.2), ANYTHING LESS THAN THE CUTOFF WILL BE SPLIT

OUTPUTS

- NONE - A SPLIT OF THE HEXES HOPEFULLY RESOLVING THE ISSUES

removeAndSmooth (goto python line: 828) :

REMOVE ELEMENTS GIVEN, SMOOTH THE AREA AS DESIRED, AND ADD ITEMS TO LISTED SIDESSETS (REQUIRES MESH BASED GEOMETRY FOR VOLUME SMOOTHING)

INPUTS

- REMELEMS - A LIST OF LISTS GIVING THE TETS AND HEXES TO REMOVE [[TET_LIST], [HEX_LIST]]

OPTIONAL

- SIDESSETS - A LIST OF SIDESSETS TO ATTEMPT TO CONSOLIDATE WITH THE SKINS OF THE REMOVED ELEMENTS
- SMOOTH - ATTEMPT TO SMOOTH THE REMOVED AREA (DEFAULT IS SET TO FALSE, AS THIS MAY DESTROY ELEMENT QUALITY)
- SMOOTHITERS - THE NUMBER OF SMOOTH ITERATIONS (DEFULTED TO 3)
- SMOOTHSCHEME- THE SCHEME USED FOR SMOOTHING (DEFAULTS TO CONDITION NUMBER)
- DUPFACES - FLAG TO REMOVE TRIS WHICH ARE PRESENT IN THE REMOVAL SURFACE (I.E. DON'T CREATE ANY 2 SIDED SIDESSETS)

OUTPUTS

- A MESH WITH REMOVED ELEMENTS ADDING OR CONSOLIDATING NEAR SIDESSETS, THE NEWLY SKINNED SIDESSET

nodeLoopLocAvg (goto python line: 969) :

THIS FUNCTION POPS ALL NODES IN THE GIVEN EDGE LOOP TO THE AVERAGE POSITION OF THAT EDGE LOOP (I.E. WILL MAKE A FLAT EDGE LOOP WRT THE AVERAGE DIR)

INPUTS

- EDGE ID - AN EDGE ID AS AN INTEGER TO USE FOR AN EDGE LOOP STARTING POINT
- AVGDIR - THE DIRECTION IN WHICH TO AVERAGE AS A NORMAL DIRECTION, LIST [X, Y, Z]

OPTIONAL

- EDGE LOOP ANGLE - AN ANGLE IN DEGREES AT WHICH THE EDGE LOOP SHOULD TERMINATE AS A FLOAT

OUTPUTS

- A TUPLE LIST OF THE NODES AND THIER MOVEMENTS, NODES MOVED TO BE ALIGNED LINEARLY ABOUT THE AVERAGE IN THE NORMAL PROVIDED

nodeLoop2PointSpline (goto python line: 995) :

THIS FUNCTION REARRANGES NODES TO LIE ON A GIVEN LINE/SPLINE IN ORDER TO ELIMINATE SKEW

INPUTS

- NODESPLINE - A SERIES OF NODES WHICH WILL GENERATE A SPLINE FOR THE EDGE LOOP TO FOLLOW
- SE - A SEED EDGE TO START THE EDGE LOOP SEARCH

OPTIONAL

- BREAKANGLE - THE ANGLE IN WHICH TO TERMINATE THE EDGE LOOP IF DESIRED

OUTPUTS

- NONE - SNAP THE NODES IN THE EDGE LOOP TO THE SPLINE GENERATED BY THE NODES GIVEN

squeezeMesh (goto python line: 1022) :

DISTORT A HEX MESH USING A SPECIFIC SWEEP DIRECTION AND PERCENTAGE DISTORTION VALUE ** REQUIRES A CONTINUOUS LOOP IN THE DISTORTION DIRECTION **

INPUTS

- SURFLIST - A LIST OF LINKING SURFACES
- SN - A NORMAL DIRECTION FOR SQUEEZING (GENERALLY THIS SHOULD BE THE SWEEP DIRECTION)
- PERC - A PERCENTAGE OF THE ORIGINAL SURFACE HEIGHT TO SQUEEZE AS A FLOAT (0-1), I.E. A PERCENTAGE OF 0.5 MOVES MESH EDGES TO HALF THE HEIGHT

OPTIONAL

- TOL - AN ANGLE TOLERANCE AS A FLOAT FOR DEVIATION FROM THE SWEEP DIRECTION

OUTPUTS

- NONE, THE MODIFIED MESH

meshBasedSideset (goto python line: 1095) :

CREATE A SIDESSET USING A SURFACE FOR PROJECTION THIS IS USEFUL FOR GEOMETRY ASSOCIATION BETWEEN MESH BASED AND NON-MESH BASED COMPONENTS, OR SIMPLY TRYING TO PROJECT ONTO A LARGER SURFACE

INPUTS

- A LIST OF SURFACES AS A CUBIT ID STRING TO IMPRINT ONTO THE MESHED SURFACE
- THE MESHED SURFACE ID TO IMPRINT UPON AS AN INTEGER OR CUBIT LIST SRING

OPTIONAL

- TOL - THE TOLERANCE TO ADHERE TO FOR EDGE FACES

OUTPUTS

- A LIST OF THE FACES THAT ARE WITHIN THE IMPRINT RANGE

mapSidesets (goto python line: 1128) :

ATTEMPT TO MAP AND COPY SIDESSET NAMES FROM ONE MESH TO ANOTHER THIS ASSUMES GEOMETRY SIMILARITIES ARE PRESENT AND ORIENTATION IS THE SAME OLD AND NEW FILE ARE LISTS OF 2 - ['FILENAME', 'ADDITIONAL CUBIT OPTIONS']

INPUTS

- A LIST OF 2 STRINGS CONTAINING A FILE PATH AND CUBIT OPEN OPTIONS FOR THE TEMPLATE MESH
- A LIST OF 2 STRINGS CONTAINING A FILE PATH AND CUBIT OPEN OPTIONS FOR THE NEW MESH

OPTIONAL

- A DISTANCE TOLERANCE FOR MATCHING THE SIDESSETS IN SPACE

OUTPUTS

- NONE, RENUMBERS THE SIDESSETS OF THE NEW FILE TO MATCH THAT OF THE OLD FILE

orderEnclosures (goto python line: 1189) :

GIVE SOME ORDER TO THE WAY CUBIT PLOPS OUT ENCLOSURE SIDESSETS; ALSO ADDS PROPERTIES TO THESE ENCLOSURES IN THE FORM OF A DESCRIPTION

INPUTS

- NONE - NO DIRECT INPUTS ARE NEEDED; THE DEFAULT IS TO USE ALL SIDESSETS AND ORDER BY LOCATION FROM ZERO

OPTIONAL

- SIDESSETLIST - A LIST OF THE SIDESSET IDS AS A CUBIT LIST STRING YOU WISH TO BE INTERROGATED
- ZP - THE ZERO POINT TO BE USED IF SORTING BY LOCATION
- ORDERTYPE - THE WAY TO SORT THE SIDESSETS, VIA LOCATION OR VIA TOTAL SURFACE AREA [location or area]
- DEFAULTBF - THE DEFAULT BULK FLUID TO USE FOR THE GIVEN ENCLOSURES
- DEFUALTCC - THE DEFAULT BULK FLUID CONVECTION COEFFICIENT TO USE FOR THE GIVEN ENCLOSURES

OUTPUTS

- OEL - THE ENCLOSURE/SIDESSET LIST ORDERED AS DIRECTED, WILL ALSO AUTOMATICALLY RENUMBER THE SIDESSETS IN CUBIT BASED ON THIS LIST

testEnclosures (goto python line: 1238) :

TEST ENCLOSURES BY ATTEMPTING TO VOLUMETRICALLY MESH THEM, ASSUMING FAILED VOLUMETRIC MESHING MEANS WE HAVE ENCLOSURE ISSUES

INPUTS

- NONE

OPTIONAL

- ENCIDS - THE ENCLOSURE SIDESSET IDS (ALL ARE SELECTED BY DEFAULT)

OUTPUTS

- ENCVOLS - THE VOLUMES OF THE SUCCESSFULLY MESHED ENCLOSURES

autoGenSidesets (goto python line: 1266) :

AUTOMATICALLY GENERATE LABELED SIDESSETS USING VOLUME NAMES AND A SIDESSET COUNTER PROVIDED A SURFACE LIST STRING

INPUTS

- A LIST OF SURFACE IDS AS A CUBIT LIST STRING

OPTIONAL

- A BOOLEAN SWITCH TO COMBINE NEIGHBORING SURFACES

OUTPUTS

- NONE, CREATES AND NAMES SIDESSETS USING THE SURFACES SELECTED

autoPairSidesets (goto python line: 1296) :

ATTEMPT TO AUTOMATICALLY PAIR NEAR SIDESSETS

INPUTS

- SSGROUPS - WHAT SIDESSETS TO CONSIDER AS A CUBIT LIST STRING

OPTIONAL

- TOL - THE DISTANCE BETWEEN SIDESSETS THAT IS DEEMED ACCEPTABLE

- NORMTOL - A TOLERANCE TO USE FOR FACE TO FACE NORMAL MATCHING

OUTPUTS

- MINP - THE PERCENTAGE OF FACES THAT NEED TO BE IN RANGE FOR A PAIR (CURRENTLY AT LEAST 75

autoSpiderSidesets (goto python line: 1367) :

AN ATTEMPT TO AUTOMATICALLY SET UP SPIDER ELEMENTS BETWEEN TWO MESHED BODIES, GIVEN THAT THEY ARE JOINED; USES SIDESSET FACE AVERAGE LOCATION AND A SPECIFIED TOLERANCE SO THIS WILL WORK WITH MESH BASED GEOMETRY AS WELL *** NOTE THAT SIDESSETS ARE REQUIRED HERE

REFER TO autoGenSidesets FUNCTION ABOVE FOR QUICK GENERATION ***

INPUTS

- A LIST OF TUPLE PAIRS CONTAINING BLOCKS TO BE JOINED I.E. [(1,2), (1,3)] WILL JOIN 1 TO 2 AND 1 TO 3 IN 2 OPERATIONS

OPTIONAL

- A TOLERANCE FOR FINDING NEAR SIDESSETS

- A LIST OF SIDESSETS TO SKIP AS A LIST

- IF THE MESH IS A TET OR TRI MESH TRI MUST BE TRUE

OUTPUTS

- A DICTIONARY CONTAINING THE MAPPING

genSlotModel (goto python line: 1541) :

GENERATE AN EM SLOT MODEL, GIVEN A SURFACE THE MESH SIZING, AND THE POSSIBLE HARDWARE INTERSECTORS *** ASSUMES 2 CLOSED LOOPS ONLY ***

INPUTS

- SLOTSURF - THE SLOT SURFACE ID (INTEGER)

- HWRAD - THE RADIUS TO EXCLUDE AROUND THE HARDWARE AS A FLOAT NUMBER

- MSIZE - THE DESIRED MESH SIZE (FLOAT)

OPTIONAL

- HWVOLS - THE NAMES OR LIST OF VOLUMES AS A STRING, THAT CONTAINS INTERSECTING HARDWARE

- KEEPSURFS - SURFACES TO EXPLICITLY KEEP DURING INITIAL GEOMETRY MANIPULATION STAGES (DEFAULT IS NONE)

- WEDGESEGS - THE NUMBER OF WEDGE SEGMENTATION POINTS TO USE IF NO HW VOLUMES ARE SPECIFIED (DEFAULT IS 4)

- NCS - THE NUMBER OF SEGMENTS USED FOR NEAREST CURVE PROJECTION (DEFAULT IS 15)

- TOL - A TOLERANCE VALUE FOR ?

OUTPUTS

- THE SPLIT LOCATIONS OF THE INNER AND OUTER CURVE LOOPS

loadCubitMatsCSV (goto python line: 2086) :

GENERATE CUBIT MATERIALS USING A CSV FILE, USE THE EXPORT FUNCTION TO BE ABLE TO READ AND WRITE MATERIALS

INPUTS

- THE CSV FILE AS A PATH STRING TO LOAD WHICH CONTAINS THE MAJOR MATERIALS PROPERTIES CURRENTLY IN CUBIT

OUTPUTS

- NONE - THE MATERIALS SHOULD BE GENERATED AS NEEDED

exportCubitMatsCSV (goto python line: 2117) :

EXPORT THE FEA MATERIALS CREATED IN CUBIT FOR USE IN OTHER MESH MODELS OR GENERAL MATERIALS INFO

INPUTS

- NONE

OUTPUTS

- A CSV FILE NAMED CUBITMATSEXPORT.CSV IN THE CURRENT WORKING DIRECTORY AND A RETURNED MATERIAL PROPERTIES DICTIONARY

exportBlockMats (goto python line: 2137) :

WRITE OUT SOME USEFUL INFORMATION REGARDING BLOCK MATERIAL, SPECIFICALLY ASSIGNED MATERIAL AND MATERIAL YIELD STRENGTH

INPUTS

- NONE

OUTPUTS

- A CSV OUTPUT WITH THE BLOCKS ID, BLOCK MATERIAL NAME, AND BLOCK YIELD STRESS

scaleMatsByLength (goto python line: 2158) :

SCALE THE CURRENT CUBIT MATERIAL PARAMETERS DUE TO LENGTH SCALE CHANGES IN A MODEL (I.E. MATERIALS IN METERS BUT MODEL LENGTH IS IN CM)

INPUTS

- A LENGTH SCALE PARAMETER TO MODIFY THE MATERIALS PROPERTIES (I.E. THE MATERIALS PROPERTIES ARE IN STANDARD SI UNITS, BUT THE PART IS IN INCHES)

OUTPUTS

- NONE, MODIFIED MATERIAL PROPERTIES

genEnclosureInp (goto python line: 2189) :

GENERATE ENCLOSURE INPUTS FROM CUBITS AUTOMATIC ENCLOSURE ROUTINE, USE ORDERENCLOSURE FIRST TO POPULATE ENCLOSURE DESCRIPTIONS AUTOMATICALLY

INPUTS

- NONE - IF NOTHING IS PROVIDED ALL SIDESSETS WILL BE ASSUMED TO BE ENCLOSURES

OPTIONAL

- ENCIDS - A CUBIT LIST STRING OF ENCLOSURE IDS TO USE, DEFAULT IS SET TO ALL SIDESSETS

OUTPUTS

- NONE - WRITES A ENCLOSURES.INP FILE USING THE EXISTING MATERIALS AND BLOCKS INTO THE CURRENT WORKING DIRECTORY

genSalinasBlockMats (goto python line: 2231) :

GENERATE BLOCK MATERIAL ASSIGNMENTS FOR SALINAS USES THE BLOCK ELEMENT TYPE TO INTELLIGENTLY CREATE BLOCK CALLS BASED ON TYPE

INPUTS

- NONE, MATERIALS, ELASTIC PROPERTIES, SPRING STIFFNESSES, ETC. SHOULD BE DEFINED IN CUBIT

OUTPUTS

- NONE, WRITES A SALINAS_MATS.INP FILE USING THE EXISTING MATERIALS AND BLOCKS INTO THE CURRENT WORKING DIRECTORY

genPrestoBlockMats (goto python line: 2314) :

GENERATE BLOCK MATERIAL ASSIGNMENTS FOR PRESTO/ADAGIO USES THE BLOCK ELEMENT TYPE TO INTELLIGENTLY CREATE BLOCK CALLS BASED ON TYPE

INPUTS

- MESHNAME AND MESH TYPE FOR DATABASE INPUTS; MATERIALS AND THICKNESSES SHOULD BE ASSIGNED IN CUBIT

OPTIONAL

- MODEL TYPE - TYPE OF MATERIAL MODEL TO BE USED (ELASTIC, MLEP_FAIL, ETC.)

OUTPUTS

- NONE, WRITES A PRESTO_MATS.INP FILE USING THE EXISTING MATERIALS AND BLOCKS INTO THE CURRENT WORKING DIRECTORY

buildVelodyneMatLib (goto python line: 2396) :

GATHER THE MATERIAL AND EOS DATA FROM THE DAT FILE INCLUDES INPUTS

MATFILENAME - THE FILE TO BE PARSED, SPACES IN TITLES MAY BREAK THIS... OUTPUTS

NONE - OUPUTS ANY OMITTED MATERIALS FROM THE LIBRARY DURING PARSE FOR FURTHER INVESIGATION

genVelodyneBlockMats (goto python line: 2457) :

GENERATE THE MATERIALS INPUT DECK FOR VELODYNE RUNS DIRECTLY FROM CUBIT

INPUTS

- NONE

OPTIONAL

- TARGET - IF TRUE ADD TO THE PART NUMBER TO STICK WITH GENERAL FORMAT

OUTPUTS

- GENERATES A Parts_def.vinc FILE IN THE CURRENT CUBIT DIRECTORY

autoPopVelodyneMatIDs (goto python line: 2571) :

AUTOMATICALLY POPULATE VELODYNE MATERIAL IDS BASED ON MATERIAL ASSIGNED IN CUBIT ** RUN AFTER GENERATING BLOCK MAT ATTRIBUTES **

INPUTS

- NONE - CHECKS BLOCKS FOR ASSIGNED VELODYNE MATERIALS

OUTPUTS

- NONE - POPULATES THE MATID SECTION OF THE BLOCK ATTRIBUTES

genAccelVInc (goto python line: 2588) :

WRITE OUT ACCELEROMETER NODAL POINTS INTO AN INCLUDE FILE BASED ON SPECIAL BLOCK KEYWORDS USING BLOCK(S) NAMED VELODYNE_ACCELS* CONTAINING HEXES TO BE USED AS ACCELEROMETER LOCATION DEFINITIONS *** NOTE YOU MAY NEED TO SET DUPLICATE BLOCKS ON IN CUBIT ***

INPUTS

- DIR - A COMMON NORMAL DIRECTION TO BE SHARED AS A FIRST EDGE ALIGNMENT *** BEST TO PICK THE SWEEP DIRECTION ***

OPTIONAL

- ATOL - AN ANGLE TOLERANCE FOR PICKING THE FIRST EDGE, ELEMENTS WITH HEAVY MISALIGNMENTS WILL BE IGNORED
- DRAW - CREATE A CURVE ELEMENT THAT DEPICTS THE ACCEL ITEM

OUTPUTS

- ACCEL_DEFS.VINC FILE AS A TEXT FILE IN THE CURRENT CUBIT WORKING DIRECTORY

genAMRTracerVInc (goto python line: 2682) :

GENERATE AN AMR TRACER POINT LOCATIONS INPUT FILE USING MESHED VOLUMES, SURFACES, OR LINES WITH THE NAME AMR_TRACER*

INPUTS

- NONE, A MESHED GEOMETRY ITEM WITH THE NAME AMR_TRACER*

OPTIONAL

- SCALE - A LENGTH SCALE IF THE MODEL IS NOT IN REGULAR
- OUTPUTTIME - THE FREQUENCY BETWEEN OUTPUTS

OUTPUTS

- AMR_TRACERS.vinc FILE IN THE CURRENT CUBIT WORKING DIRECTORY

setStatGeomSize (goto python line: 2726) :

SET THE ELEMENT SIZE BASED ON THE AVERAGE CURVE LENGTH PER VOLUME SIZE SET; RETURNS THE SIZING MAP

INPUTS

- A LIST OF VOLUMES TO SIZE

OPTIONAL

- THE TYPE OF MESH TO BE USED, TRI OR HEX

OUTPUTS

- THE SIZING DICTIONARY USED TO SIZE THE VOLUMES

recursiveTetCollapse (goto python line: 2763) :

RECURSIVELY REMOVE BAD TETS WHEN POSSIBLE USING A QUALITY METRIC VALUE AND A COLLAPSE OPTIMIZED METRIC

INPUTS

- VOLLIST - A LIST OF VOLUMES CONTAINING THE TETS TO ASSESS
- QVALUE - THE THRESHOLD VALUE FOR THE QUALITY METRIC
- QMETRIC - THE QUALITY METRIC NAME IN CUBIT SPEAK
- COMPSTR - THE COMPARISON VALUE STRING GE (GREATER THAN OR EQUAL TO) OR LE (LESS THAN OR EQUAL TO)
- CMETRIC - THE METRIC TO ATTEMPT TO OPTIMIZE DURING THE COLLAPSE PROCEEDURE (CAN DIFFER FROM THE BAD TET METRIC IF DESIRED)
- POSTSMOOTHING - POST COLLAPSE SMOOTHING BOOLEAN (DEFAULT IS TRUE OR ON)

OUTPUTS

- NONE, COLLAPSED TETS

exoFromNG (goto python line: 2798) :

BUILD AN EXO FROM THE BASE NG FORMAT, USEFUL FOR GOING BETWEEN BELNDER AND CUBIT FOR SHELL MESHING NEEDS OR DEFORMATIONS

INPUTS

- NGFILEPATH - THE PATH TO THE NG FORMATTED FILE

OUTPUTS

- NONE, CREATES AND EXODUS MESH STRUCTURE IN CUBIT

11 PyCubed_GeomManip Overview

pointcloudFromCSV (goto python line: 25) :

READ X Y Z POINTS FROM A CSV FILE, AND CREATE THEM

INPUTS

- A FILENAME STRING POINTING TO A 3 COLUMN CSV FILE, CONTAINING X, Y AND Z POINTS (NO HEADER SUPPORT YET)

OUTPUTS

- NONE, A VERTEX CLOUD SPECIFIED BY THE CSV FILES POINTS

ACISfromNG (goto python line: 43) :

READ IN A BLENDER MESH (N-GONS SUPPORTED), FAILS IF BLENDER FACES HAVE A DISTORTION VALUE (I.E. ARE NOT PLANAR)

INPUTS

- A FILE NAME STRING POINTING TO AN N-GON FORMATTED ASCII MESH FILE (PLEASE READ MANUAL FOR NG FORMAT INFO)

OUTPUTS

- NONE, ACIS FACE GEOMETRY

ngFromExo (goto python line: 101) :

WRITE A BLENDER SURFACE MESH FROM EXODUS MESH - FOR USE WITH THE NG READ AND WRITE BLENDER TOOLS PROVIDED

INPUTS

- FILENAME - A FILENAME STRING

OPTIONAL

- MULTIBLOCK - A BOOLEAN TO ALLOW FOR MULTIPLE FILE OUTPUTS RATHER THAN A SINGLE ALL ENCOMPASSING FILE
- EXPORTBLOCK - A SUBSET OF BLOCKS TO EXPOST AS A CUBIT STRING LIST (DEFAULTS TO ALL BLOCKS)

OUTPUTS

- NONE, A FILE CONTAINING THE MESH FACES AND NODES IN N-GON FORMAT FOR BLENDER READ-IN

tweakTangent (goto python line: 196) :

A TANGENCY BLUNTING OPERATION WHICH MODIFIES THE TANGENT AREA USING A NON VIRTUAL GEOMETRY APPROACH

INPUTS

- SURFID - THE SURFACE ID TO TWEAK (THE CURVED SURFACE CREATING THE SHARP TANGENT)
- OPTIONAL
- FRACTION - THE FRACTION TO USE TO TRIM THE CURVE TO TANGENT
 - DELTA - A PROJECTION DISTANCE OUT ON THE CURVES ASSOCIATED WITH A VERTEX USED TO CHECK PINCH ANGLES (LARGER VALUES MAY BE NEEDED FOR LARGER SCALE GEOMS)
- OUTPUTS
- NONE, A MODIFIED GEOMETRY

splitBoltGeom (goto python line: 323) :

SPLIT BOLT GEOMETRIES USING A BOLT VOLUME LIST AND OTHER INTERSECTING VOLUME LIST

- INPUTS
- BOLTVOLS - A LIST OF BOLT VOLUME IDS AS A CUBIT STRING FOR SPLITTING
 - INTERVOLS - A LIST OF OTHER VOLS THAT ARE JOINED BY THE BOLTS AS A CUBIT STRING, OMIT THINGS LIKE
- OPTIONAL
- TOL - AN ANGLE TOLERANCE TO CHECK WHEN ALIGNING THE BOLT AND SURFACE NORMALS
 - IGNOREMERGED - A BOOLEAN FLAG THAT WILL IGNORE MERGED SURFACES IF TRUE (DEFAULT IS SET TO TRUE)
- OUTPUTS
- NONE, BOLTS THAT ARE SPLIT AND NAMED APPROPRIATELY

moveRadially (goto python line: 467) :

MOVE GEOMETRY ENTITIES IN A RADIAL MANNER GIVEN THE GEOMETRY, RADIAL CENTER POINT AND NORMAL DIRECTION

- INPUTS
- GEOMLIST - A LIST OF GEOMETRY ENTITIES AS A CUBIT STRING
 - GEOMTYPE - THE TYPE OF GEOMETRY LIST GIVEN AS A STRING
 - CENTERLOC - THE CENTER POINT OF THE RADIAL PATTERN AS A LIST OF FLOATS
 - NORMDIR - THE NORMAL DIRECTION OF THE RADIAL PATTERN AS A LIST OF FLOATS
- OUTPUTS
- RADMOVE - THE MOVEMENT VECTOR OF EACH ITEM IN THE GIVEN LIST AS A TUPLE LIST

removeExtraVerts (goto python line: 494) :

REGULARIZE VERTS IN A LIST BASED ON IF THEY ACTUALLY CONTRIBURE TO A SPLIT OR NOT, KEEPS DELIBERATE SPLITS INTACT

- INPUTS
- VERTEXLIST - A VERTEX LIST TO REGULARIZE
- OUTPUTS
- NONE, A REGULARIZED VOLUME WITH T SPLITS INTACT

trimAllLinear (goto python line: 509) :

EXTEND OR TRIM ALL LINEAR CURVES OF THE SURFACES GIVEN IN ORDER TO SPLIT THEM IN A MEANINGFUL WAY

- INPUTS
- SURFIDS - A CUBIT LIST STRING CONTAINING ALL OF THE DESIRED SURFACES FOR SPLITTING
- OPTIONAL
- NONE
- OUTPUTS
- NONE - MODIFIED GEOMETRY

smearVerts (goto python line: 535) :

IMPRINT VERTS BELONGING TO THE SURFACE ALONG THE NORMAL VECTOR GIVEN

- INPUTS
- SURFLIST - A SURFACE ID LIST AS A CUBIT LIST STRING
 - NORMALVECT - A NORMAL VECTOR AS A LIST OF FLOATS (I.E. [X,Y,Z])
- OPTIONAL
- BORROWED - A CUBIT LIST STRING OF VERTS IN ANOTHER SURFACE TO BORROW (THIS WILL HAVE A GLOBAL EFFECT ON ALL SURFACES)
 - VERTLIST - A LIST OF VERTS IN THE SURFACE(S) TO USE, THIS IS USEFUL FOR REMOVING PROBLEMATIC VERTS
- OUTPUTS
- NONE - A GEOMETRY MODIFIED GIVEN THE SURACES AND NORMAL

dirSurfSplit (goto python line: 564) :

SPLIT SURFACE USING A DIRECTION CURVE DEFINED BY A DIRECTION ** CAREFUL I USE THE FIRST CURVE IN LINE THAT FITS THE BILL **

- INPUTS
- SURFIDS - A CUBIT LIST STRING CONTAINING ALL OF THE DESIRED SURFACES FOR SPLITTING

- NORMDIR - A NORMAL DIRECTION OF THE CURVE AS A LIST OF FLOATS [X,Y,Z]
- OPTIONAL
- ANGLETOL - A TOLERANCE ANGLE IN DEGREES DEFAULTED TO 70 DEGREES
 - NSEGS - THE NUMBER OF SEGMENTS TO SPLIT INTO (DEFAULT IS 2)
- OUTPUTS
- ALLMATCHED - THE CURVES MATCHING THE ASSOCIATED NORMAL DIRECTION AND TOLERANCE
 - NONE - MODIFIED GEOMETRY

copyCombineCurves (goto python line: 587) :

CREATE A SINGLE SPLINE CURVE FROM MULTIPLE ADJOINING SELECTED CURVES ***ASSUMES THAT THIS IS NOT A CLOSED LOOP***

- INPUTS
- CURVELIST = A CUBIT LIST STRING CONTAINING A LIST OF ADJOINING CURVES
- OUTPUTS
- CID - THE CURVE ID OF THE GENERATED SPLINE

tierNormals (goto python line: 618) :

TIER A NORMAL SURFACE LIST IN THE FORM OF NORMAL DIRECTION AND STARTING POINT

- INPUTS
- NORMSURFLIST - A LIST OF SURFACES GIVEN AS A CUBIT LIST STRING
- OPTIONAL
- TOL - A TOLERANCE ON THE LOCATION DEFAULTS TO
 - ANGLETOL - AN ANGLE TOLERANCE TO CONSIDER WHEN GROUPING NORMAL DIRECTIONS
 - GENPLANE - A BOOLEAN VALUE THAT GENERATES A PLANE AT THE TIER
- OUTPUTS
- NTIERS - THE TIERS FOUND AS A TUPLE PAIR LIST OF NORMAL AND CENTER POINT [[[0,1,0], [0,0,0]], [[0,1,0], [2,2,2]]]
 - SURFGROUPS - A LIST OF THE LISTS CONTAINING THE GROUPED SURFACE IDS [[1,2,3,4], [5,6]]
 - PLANEIDS - RETURN THE PLANE IDS IF GENPLANE IS TRUE

collapseTiers (goto python line: 678) :

COLLAPSE THE TWO NEAREST TIERS USING TIERING ALGORITHM AND SNAPTOMIDPLANE *** BE CAREFUL ABOUT PROVIDING ALL SURFACES IN A VOLUME ***

- INPUTS
- SURFLIST - A SURFACE LIST TO TIER
 - TOL - THE SNAP THRESHOLD DISTANCE FROM THE NORMAL
 - ANGLETOL - THE SNAP ANGLE TOLERANCE, DEFAULT SHOULD BE LOW TO AVOID SKEWING GEOMETRY
- OUTPUTS
- SNAPPIES - A LIST OF THE SURFACE TIERS SNAPPED AND THE MODIFIED GEOMETRY

trimTierPlanes (goto python line: 717) :

TRIM ENTITIES WITH LIKE NAMES USING TIER PLANES BASED ON THE GIVEN SURFACE LIST

- INPUTS
- SURFLIST - A CUBIT LIST STRING OF SURFACES TO USE
- OUTPUTS
- NONE, TRIMMED VOLUMES BASED ON THE SURFACE LIST

alignVertSplits (goto python line: 741) :

SPLIT CURVE ENTITIES SUCH THAT LIKE CURVES ALL HAVE VERTEX SPLITS AT THE SAME POINT, THIS SHOULD MITIGATE SWEP SKEW ESPECIALLY IN CYLINDRICAL OBJECTS

- INPUTS
- VOLLIST - VOLUMES TO BE REALIGNED
- OPTIONAL
- SPLITPOINT - A LIST DEFINING [X,Y,Z] COORDINATES TO BE PROJECTED AS A SPLIT POINT (GREAT FOR A FLANGE TYPE SITUATION)
- OUTPUTS
- NONE, A HOPEFULLY REALIGNED SPLIT OF THE OBJECTS SELECTED

isClosedLoop (goto python line: 836) :

RETURNS THE CURVE ID IF IT IS A SINGLE CURVE WHICH IS A CLOSED LOOP (CONTAINS A SINGLE VERTEX)

- INPUTS
- CRVLIST - A LIST OF CURVE IDS AS A CUBIT LIST STRING
- OUTPUTS
- CLOSEDLOOPS - A LIST OF THE LOOPS WHICH ARE SINGLE CLOSED LOOPS

vertsFromLocs (goto python line: 848) :

SIMPLY CREATE VERTS FROM A LOCATIONS LIST

INPUTS

- THE LOCATIONS AS A LIST OF LISTS [[X, Y, Z],[XN, YN, ZN]]

OUTPUTS

- THE VERTEX IDS AS A LIST OF INTEGERS

imprintOverride (goto python line: 862) :

IMPRINTS VOLUMES, BUT ATTEMPTS TO OVERRIDE THE SECOND VOLUMES CONTRIBUTIONS BACK TO THE FIRST VOLUMES

INPUTS

- VOLUME 1 - THE GEOMETRY OF THIS VOLUME WILL ATTEMPT TO OVERRIDE THE SECOND VOLUMES CONTRIBUTIONS
- VOLUME 2 - THE SECOND VOLUME TO IMPRINT THE FIRST VOLUMES TOPOLOGY ONTO

OPTIONAL

- STOL - THE TOLERANCE VALUE FOR THE IMPRINT SURFACE TO SURFACE DISTANCE
- MINSTOL - THE MINIMUM TOLERANCE
- ASTOL - THE ANGLE TOLERANCE IF A NORMAL CALCULATION IS USED
- PROJMETHOD - THE PROJECTION METHOD USED TO CALCULATE THE DISTANCE, CENTER (CENTER-TO-CENTER), NORMAL (CENTER-TO-CENTER WITH AN ANGLE CHECK), PROJECTED (CENTER-TO-NEAREST PROJECTED POINT)

OUTPUTS

- THE IMPRINTED VOLUME IF POSSIBLE, OTHERWISE AN ERROR MESSAGE WILL BE DISPLAYED IF THE SECOND VOLUME CANNOT BE REGULARIZED DUE TO GEOMETRY

virtualTwin (goto python line: 932) :

GENERATE EXACTLY MATCHING GEOMETRIES FOR SURFACES CONTAINING VIRTUAL ENTITIES (SLOT MODELS)

INPUTS

- DOMSURFS - A LIST OF THE SURFACES TO COPY
- SUBSURFS - A SURFACE TO MODIFY TO EXACTLY MATCH THE DOMINANT SURFACES

OUTPUTS

- A LIST OF THE MODIFIED SURFACE IDS

splintize (goto python line: 1006) :

CREATE A SPLINE USING A VERTEX LIST RETURNS THE NEW CURVE ID

INPUTS

- A CUBIT LIST STRING OF VERT IDS

OUTPUTS

- THE SPLINE CURVE ID AS AN INTEGER

splintizeLocs (goto python line: 1019) :

GENERATE A CURVE GIVEN AN ORDERED LIST OF LOCATIONS INSTEAD OF SIMPLY VERTS

INPUTS

- A LIST OF LISTS [[X, Y, Z],[XN, YN, ZN]] FOR GENERATING THE VERT LOCATION OF THE SPLINE

OUTPUTS

- THE CURVE ID OF THE GENERATED SPLINE

genPolygonImprint (goto python line: 1041) :

CREATE A POLYGON AND IMPRINT IT ON A SURFACE USING A SURFACE ID AND A LIST OF LOCATIONS

INPUTS

- SURF ID - THE SURFACE ID OF THE SURFACE THAT THE POINT SHOULD LIE ON, AND THAT THE FINAL SHAPE WILL BE IMPRINTED ON
- NSIDES - THE NUMBER OF SIDES IN THE POLYGON (I.E. 0 CIRCLE, 2 LINE, 3 TRI, 4 SQUARE, 5 PENTAGON, 6 HEX, ETC.)
- SIZE - THE RADIUS USED FOR THE SHAPE PROJECTION
- LOCS - THE LOCATION OF THE CENTERS OF THE POLYGON PROJECTIONS, AS A LIST OF LISTS

OPTIONAL

- ANGLE - THE ROTATION ANGLE AS A FLOAT IN DEGREES
- CP - A CENTER POINT FOR RADIAL PATTERNING
- AUTOROTATE - AUTOMATICALLY ROTATES FOR RADIAL PATTERNING; THIS IS IN ADDITION TO THE ALREADY GIVEN ANGLE
- DISTFACT - A FACTOR FOR INCREASING THE SEARCH TOLERANCE IN ORDER TO RETURN THE IMPRINTED SURFACES

OUTPUTS

- THE IDS OF THE SURFACES GENERATED IF CLOSED, A POLYGON WITH N SIDES IMPRINTED ON THE SURFACE SPECIFIED

convertVol2Beam (goto python line: 1109) :

CONVERT A SINGLE LOOPED VOLUME INTO A BEAM ELEMENT ATTEMPT TO USE THE CENTER LINE OF THE SWEEPED STRUCTURE PROVIDED THE VOLUME AND TERMINATING SURFACES *** ASSUMPTIONS HERE ARE THAT THE VOLUME IS SPLIT AT LEAST ONCE DOWN THE SWEEP DIRECTION AND THAT THE SPLIT RUNS FROM ONE CAP END TO THE OTHER WITHIN THE ANGLE BOUNDS ***

INPUTS

- THE VOLUME ID TO TRANSFORM AS AN INTEGER OR CUBIT LIST STRING
- THE END CAP SURFACES TO USE WHEN TRACING THE PATH AS A CUBIT LIST STRING

OPTIONAL

- THE TRACE ANGLE AS A FLOAT IN DEGREES (DEFAULT IS 30.0 DEGREES)
- THE NUMBER OF SEGMENTS TO USE WHEN PLOTTING FULL THE SPLINE AS AN INTEGER (DEFAULT IS 30)
- THE NUMBER OF SEGMENTS TO SPLIT GEOMETRIC CURVES INTO AS A PRE-PROCESSING STEP (DEFAULT IS 10)

OUTPUTS

- NONE, A SPLINE CURVE THAT SHOULD TRACK THE VOLUMES CURVATURE

splitRingSkew (goto python line: 1187) :

TRANSFER VERTS OF A DOUBLE SIDED CLOSED LOOP SHAPE (I.E. ANNULAR SHAPE) FROM AN ALREADY MESHED INTERNAL OR EXTERNAL FACE/CURVES. THIS IS INTENDED TO PREVENT SKEW IN SUBMAPPED SWEEPS. *** THIS SCRIPT MAY BE SLIGHTLY OUTDATED ***

INPUTS

- A CUBIT LIST STRING CONTAINING A LIST OF SURFACE IDS

OPTIONAL

- AN ANGLE LIMIT AS A LIST OF TWO FLOATS (UPPER AND LOWER BOUND) THIS PREVENTS A SPLIT WITH HIGH SKEW A BOOLEAN FLAG TO IMPRINT THE CURVES OR LEAVE THEM AS FREE

OUTPUTS

- NONE, A SPLIT ANNULAR SURFACE

splitSurfSkew (goto python line: 1266) :

ATTEMPT TO SPLIT A CLOSED LOOP SURFACE THROUGH EXISTING VERTS USING EDGES AND ANGLE LIMITS **NEEDS WORK**

INPUTS

- SURFIDS - A CUBIT STRING LIST OF DESIRED SURFACE IDS

OPTIONAL

- SKEW LIMITS - A LIST OF UPPER AND LOWER BOUND TAKEOFF ANGLES
- IMPRINT - CREATE CURVES BUT DO NOT IMPRINT IF FALSE, IMPRINT THOSE CURVES IF TRUE

OUTPUTS

-

splitSkew (goto python line: 1328) :

SPLITS THE SKEW ON A SURFACE BASED ON VERTEX NORMALS *** SHOULD EVENTUALLY REPLACE SPLITRING SKEW AND SPLITSURFSKEW ***

INPUTS

- SURFID - THE SURFACE ID TO TRY AND SPLIT
- PROJDIST - A PROJECTION DISTANCE FOR SPLIT LINE LENGTH

OPTIONAL

- CROSSTOL - A TOLERANCE FOR CURVE CROSSING CALCULATIONS

OUTPUTS

- NONE, A SPLIT SURFACE THAT CUBIT CAN HOPEFULLY FIGURE OUT

splitSurfVN (goto python line: 1399) :

SPLIT THE SURFACE USING THE VERT NORMS OF THE GIVEN VERTEX LIST

INPUTS

- SURFID - THE SURFACE TO IMPRINT AS AN ID STRING
- VERTIDS - THE VERTS TO USE IN ORDER TO FORMULATE A SPLIT AS A CUBIT LIST STRING
- PROJDIST - THE PROJECTION DISTANCE USED TO GENERATE THE CURVES FOR IMPRINT

OUTPUTS

- NONE, THE SPLIT SURFACE

n2cImprint (goto python line: 1432) :

IMPRINT THE CURVES WITH THE NODES GIVEN THE CURVES, NODES, AND A BOUNDING TOLERANCE TOLERANCE

INPUTS

- A CURVE LIST AS A CUBIT LIST STRING
- A NODELIST AS A CUBIT LIST STRING

OPTIONAL

- THE TOLERANCE FOR IMPRINTING

OUTPUTS

- NONE, A SPLIT CURVE

remFullRounds (goto python line: 1455) :

REMOVE FULLY ROUNDED SURFACES WITH 4 CORNERS SPEEDS UP REMOVAL OF SLOTS ETC...

INPUTS

- SURFLIST - A LIST OF SURFACES TO BE REMOVED AS A CUBIT LIST STRING

OPTIONAL

- OFFSET - OFFSET THE TWEAK BY A PERCENTAGE OF THE RADIUS 0 IS FULL ARC RADIUS, 1 IS ARC CENTER POINT

OUTPUTS

- NONE, THE FLATTENING OF THE FULL ROUND

skinCurveChain (goto python line: 1482) :

SKIN THE CURVE CHAIN PAIRS AND RETURN THE SKIN SURFACES

INPUTS

- CURVELIST1 - A CUBIT LIST STRING OF THE CURVES IN THE FIRST CURVE CHAIN
- CURVELIST2 - A CUBIT LIST STRING OF THE CURVES IN THE SECOND CURVE CHAIN

OPTIONAL

- TOL - A SEARCH TOLERANCE
- MATCHANGLE - MATCH THE ANGLES WITHIN A TOLERANCE
- MANGLE - THE ANGLE TOLERANCE TO MATCH IN ORDER TO PAIR THE CURVES

OUTPUTS

- SKINS - THE SKINNED SURFACE IDS AFTER MATCH

entity2BB (goto python line: 1505) :

INDIVIDUALLY REPLACE THE ETYPE ITEMS IN THE LIST WITH A BOUNDING BOX REPRESENTAION, RENAMES THE HOUSING BOXES TO MATCH THEIR VOLUMES NAME; THIS IS FOR FAST SIMPLIFICATION OF GEOMETRY *** THIS DIFFERS FROM THE MASS BOUDING BOX CUBIT WILL RETURN ***

INPUTS

- ETYPE - THE TYPE OF ENTITY TO BOUND AS A STRING (CURVE, SURFACE, VOLUME)
- ENTITYLIST - A LIST OF ENTITIES TO BOUND AS A CUBIT LIST STRING

OUTPUTS

- BOXES - THE IDS OF THE GENERATED BOXES, BOUNDING BOXES THAT HAVE BEEN RENAMED TO THE ENTITIES THEY BOUND

genCylBB (goto python line: 1529) :

GENERATE A CYLINDRICAL BOUNDING BOX FOR AN OBJECT IF POSSIBLE

INPUTS

- VOLIDS - A LIST OF VOLUME IDS TO TRANSFORM INTO SEPERATE BOUNDING BOX ENTITIES

OPTIONAL

- TOL - A TOLERANCE TO KEEP BETWEEN THE TIGHT BOUNDING BOX NORMALS AND THE CURVE OF INTEREST NORMAL DIRECTION

OUTPUTS

- THE ID(S) OF THE VOLUMES CREATED - VOLUMES OF THE BOUNDED ITEM, WHICH ARE NAMED THE SAME AS THE ORIGINAL PARENT VOLUMES

genOffsetVol (goto python line: 1589) :

GENERATE AN OFFSET VOLUME FOR EACH VOLUME SELECTED **THIS MAY FAIL FOR COMPLEX ITEMS WITH MANY ROUNDS**

INPUTS

- VOLIDS - A LIST OF VOLUMES TO ATTEMPT TO OFFSET
- OFFSETDIST - THE DISTANCE OFFSET FROM THE ORIGINAL SURFACES, NEGATIVE VALUES RESULT IN INTERNAL GEOMETRY
- OFFSETNAME - A NAME TO USE FOR THESE OFFSET ITEMS (I.E. PAINT, INTERNAL.CORE, ETC.)

OUTPUTS

- OFFVOLS - A LIST OF THE OFFSET VOLUMES GENERATED

cadStyleAlign (goto python line: 1616) :

ALIGN ENTITIES USING FACE NORMALS ONLY, 2 PAIRS OF SURFACES *** BECAUSE OF THE WAY CUBIT STORES CIRCULAR NORMALS THIS IS NOT RECOMMENDED FOR CYLINDICAL ALIGNMENT *** ; THIS FUNCTION NEEDS TO BE CHECKED, AS IT SEEMS THAT THE METHODS USED MIGHT BE SLIGHTLY OUTDATED <<<

INPUTS

- A LIST OF TWO SURFACE IDS TO BE ALIGNED
- AN ANGLE TOLERANCE FOR ALIGNMENT AS A FLOAT (IN RADIANS?)

OUTPUTS

- THE ALIGNED VOLUMES

copyMoveReplace (goto python line: 1653) :

REPLACE VOLUMES IN LIST REPL WITH VOLUME IN REF, USING REFSURFS AS A GUIDE; THIS FUNCTION NEEDS TO BE REVISITED AGAIN, THERE ARE A NUMBER OF IMPROVEMENTS THAT CAN BE MADE <<<

INPUTS

- A NAME STRING OF THE VOLUMES TO COPY
- A NAME STING OF THE VOLUMES TO RAPLACE WITH THE COPIES
- A LIST OF 2 REFERENCE SURFACE IDS USED TO ORIENT THE COPIES CORRECTLY, IDEALLY THIS MEANS THESE SURFACES HAVE SIMILAR SURFACE AREAS AND ORIENTATIONS

OUTPUTS

- NONE, REPLACED VOLUMES

copyAlign (goto python line: 1699) :

COPY AND ALIGN ITEMS TO A SURFACE USING A SERIES OF VERTEX POINTS

INPUTS

- ALIGNSURF - THE SURFACE TO ALIGN TO AS A CUBIT STRING (CAREFUL, THIS SHOULD BE CLOSE TO THE VERTS IN QUESTION)
- ALIGNVERTS - VERTS TO MAP ONTO THE SURFACE FOR A NORMAL PROJECTION
- COPYSURF - A SURFACE TO USE AS THE ALIGNMENT TARGET AS A STRING OR SURFACE ID
- COPYRENAME - A NEW NAME TO GIVE THE COPIED OBJECT UPON ALIGNMENT AS A STRING

OPTIONAL

- FLIPAXIS - FLIP THE AXIS IF WE GOT THE POINT UPSIDE DOWN

OUTPUTS

- NONE - ALIGNED COPIES OF THE OBJECT GIVEN

planeCutWithCurve (goto python line: 1736) :

WEBCUT VOLUME(S) USING AN ARC CURVE OR SPLINE CURVE FOR PLANE COORDINATES

INPUTS

- A VOLUME LIST AS A CUBIT LIST STRING
- A CURVE ID AS A CUBIT STRING OR INTEGER

OUTPUTS

- NONE, THE WEBCUT VOLUME(S)

webcutWithCurveNorm (goto python line: 1756) :

USE CURVE AND A NORMAL DIRECTION TO WEBCUT A VOLUME *** THIS DIFFERS FROM CUBITS COMMAND BECAUSE IT USES A FULL SURFACE PROJECTION ***

INPUTS

- A VOLUME LIST AS A CUBIT LIST STRING
- A CURVE ID AS A CUBIT STRING OR INTEGER
- A NORMAL DIRECTION AS A LIST OF FLOAT NUMBERS [X, Y, Z]

OUTPUTS

- NONE, THE WEBCUT VOLUME(S)

webcutWithCurveSandT (goto python line: 1780) :

USE A CURVE, A SCALE OF THAT CURVE, AND A DISTNACE PROJECTION OF THE SCALED CURVE TO CONSTRUCT A SURFACE FOR WEBCUTTING

INPUTS

- CIDS - A CUBIT LIST STRING CONTAINING THE CURVE IDS
- VIDS - THE VOLUME IDS TO WEBCUT WITH THE GENERATED SURFACE
- SCALE - THE AMOUNT TO SCALE THE CURVES BY AS A LIST OF LENGTH 3 [XSCALE, YSCALE, ZSCALE]
- TRANSLATION - A TRANSLATIONAL VECTOR FROM THE ORIGIN OF THE STARTING CURVES IN WHICH TO MOVE THE SCALED CURVES
- EXTEND - ATTEMPT TO EXTEND THE CUT OR NOT, FLASE IS BEST FOR FULL LOOPS

OUTPUTS

- NONE - A WEBCUT SURFACE

tweak3PointPlane (goto python line: 1822) :

TWEAK A SURFACE TO A PLANE USING 3 VERTS, THIS REMOVES THE NEED TO CREATE A SURFACE, TWEAK TO IT AND THEN DELETE IT

INPUTS

- SIDS - A SURFACE LIST ID AS A CUBIT LIST STRING OR SINGLE INTEGER

- VERTLIST - A LIST OF 3 VERTS AS A CUBIT LIST STRING (WILL TRUNCATE TO THE FIRST 3 IF MORE ARE PROVIDED)
- OUTPUTS
- NONE, A TWEAKED PLANE

snapToMidplane (goto python line: 1842) :

SNAP SURFACES TO A MIDPLANE GIVEN A SURFACE LIST

- INPUTS
- SURFLIST - A CUBIT LIST STRING
- OUTPUTS
- NONE, MODIFIES OR TWEAKS THE SURFACES TO THE MIDPLANE IF POSSIBLE

autoRemovePlanarGaps (goto python line: 1871) :

REMOVE GAPS FROM THE MODEL BASED ON NEAREST NEIGHBORING PLANAR SURFACES IN OTHER VOLUMES

- INPUTS
- VOLLIST - A CUBIT LIST STRING OF VOLUME IDS TO COVER
- OPTIONAL
- TOL - A GAP TOLERANCE SIGNIFYING A PROJECTED DISTANCE
 - MINTOL - A MINIMUM TOLERANCE TO BE CONSIDERED (THIS ALLOWS FOR USERS TO TWEAK FOR OVERLAP WHEN NEGATIVE)
 - SNAPDIR - A SPECIFIED NORMAL DIRECTION TO CONSIDER ONLY
 - ANGLE - THE MAXIMUM DIVIATION ANGLE TO CONSIDER FOR SURFACES TO SNAP
- OUTPUTS
- NONE, MODIFIED GEOMETRY; TWEAKING THE SURFACE WITH THE SMALLER SURFACE AREA TO THE LARGER ONE

slowSubtraction (goto python line: 1929) :

REMOVE VOLUMES FROM A SET NAMED VOIDNAME THIS KEEPS ALL ORIGINAL VOLUMES, AND TRACKS AND MODIFIES VOLUMES WHEN THEY ARE SPLIT/REBUILT FROM THE ORIGINALS

- INPUTS
- VOIDIDS - VOLUME ID(S) AS A CUBIT LIST STRING TO START REMOVING VOLUMES FROM
 - VOIDNAMES - THE NAME OF THE VOID VOLUME(S) SO THAT WE CAN TRACK THE SPLITTING USING NAMES
- OPTIONAL
- IGNOREDVOLS - VOLUMES TO OMIT FROM THE SUBTRACTION PROCESS
- OUTPUTS
- NONE, THE MODIFIED GEOMETRY

gen2DRotModel (goto python line: 1981) :

- INPUTS
- COORDAXIS - A STRING GIVING THE ROTATION AXIS OF THE PART
- OPTIONAL
- VOLLIST - A LIST OF VOLUMES TO USE IN THE CROSS SECTION
 - ROTANGLE - THE ROTATION OF THE CROSS SECTION PLANES
 - NODESPERCURVE - NUMBER OF SEGMENTS TO GIVE EACH CURVE
 - FLIPUS - FLIP THE U PARAMETERS IF THINGS ARE BACKWARDS
- OUTPUTS
- A FILE NAMED ROT2DGEOM.NG CONTAINING A LIST OF POINTS WHICH CREATE THE SURFACES USED FOR ROTATIONAL MODELS, WRITES TO THE CURRENT DIRECTORY

cmdLooper (goto python line: 2108) :

EXECUTES A GENERIC CUBIT FUNCTION INSIDE A LOOP cubitCMD IS A STRING PROVIDED IN REGULAR FORMAT SYNTAX (EXAMPLE

'move vol

- INPUTS
- A CUBIT COMMAND LINE ARGUMENT WHICH HAS PYTHON IN LINE CHARACTER REPLACEMENT STRINGS INTERNAL TO IT
 - A TUPLE LIST USED TO REPLACE THE STRING FORMATTING ARGS *** THERE NEEDS TO BE THE SAME NUMBER OF REPLACEMENTS AS TUPLE ITEMS ***
- OUTPUTS
- NONE, LOOPS THE CUBIT COMMAND

12 Examples

EXAMPLE 1: FLANGE SIMPLIFICATION AND MESHING

```
#!/python
import sys, os
#ADD THE SCRIPT PATH TO THE RELATIVE PATH
scriptPath = 'C:\\Users\\jdoe\\cubit-python-enhancements-master\\Scripts\\'
sys.path.append(scriptPath)
import PyCubed_Main
from PyCubed_Main import *
from cubit import *

#-----#
#--- GENERAL GEOMETRY MANIPULATION AND MESHING ---#
#OBJECTIVE:
#MODIFY THE GEOMETRY AND MESH IT SO THAT IT IS USABLE FOR MOST SIMULATIONS
#WE ARE GOING TO DO THIS IN AN ID-LESS WAY USING NAMES AND VOLUME/SURFACE/CURVE
CHARACTERISTICS
#(YOU CAN CHOSE TO DO THIS BUT YOU CAN ALSO SIMPLY PROVIDE IDS AS WELL; THIS
IS UP TO THE USER)
reset()
workingDir = 'H:/Desktop/'
cmd('import acis "%sV2.sat" heal' % (workingDir))
#LET US RENAME USING THE VOLUMETRIC SIZE OF THE COMPONENTS
#(THIS STEP IS USUALLY NOT NEEDED WHEN IMPORTING CAD GEOMETRY FROM CREO)
renameVols(rename='BOLTS', volList=l2s(selByVolume('all', 'EQ', 28546.7,
tol=.5)))
renameVols(rename='NUTS', volList=l2s(selByVolume('all', 'EQ', 14589.5, tol=.5)))
renameVols(rename='T_FLANGE', volList=l2s(selByVolume('all', 'EQ', 5604137.1,
tol=.5)))
renameVols(rename='FLANGE_CAP', volList=l2s(selByVolume('all', 'EQ', 1901203.5,
tol=.5)))
autoGroupByName(True)
#####
#NOW LET'S SIMPLIFY THE NUTS AND BOLTS#
#####
sa = selByArea('in vol with name "*BOLTS*"', 'EQ', 572.641307)
cmd('remove surface %s extend' % l2s(sa))
#SELECT THE HEXES
ps = selPolygonSurfs('in vol with name "*BOLTS*"', 6)
#SELECT CONNECTED GEOMETRY, BUT REMOVE THE HEX FROM SELECTION
m = selMore(l2s(ps), l2s(ps))
#RECURSIVELY SELECT MORE ATTCHED GEOMETRY, AGAIN OMITTING THE HEX
m = selMore(l2s(m), l2s(ps))
cmd('remove surface %s extend' % l2s(m))
sa = selByArea('in vol with name "*NUTS*"', 'EQ', 894.764236)
cmd('remove surface %s extend' % l2s(sa))
#SELECT ALL THE BLENDS IN THE VOLUME WITH A SIZE LESS THAN 10
bs = selBlendSurfs('in vol with name "*T_FL*"', 10.0)
cmd('remove surface %s extend' % l2s(bs))
cmd('vol all size 5.0')
cmd('split periodic vol with name "*BOLT*"')
```

```

removeExtraVerts('in vol with name "*BOLT*")
sa = selByArea('in vol with name "*BOLT*', 'EQ', 201.062, 0.5)
cmd('surface %s scheme circle fraction 0.3333' % l2s(sa))
cmd('curve in surface %s interval 6' % l2s(sa))
sa = selByArea('in vol with name "*BOLT*', 'EQ', 603.186, 0.5)
cmd('surface %s scheme hole rad_intervals 3' % l2s(sa))
## -----##
## sa = selByArea('in vol with name "*T_FL*", 'EQ', 1963.5, 0.5)
## cmd('remove surf %s extend' % l2s(sa))
## sa = selByArea('in vol with name "*CAP*", 'EQ', 1963.5, 0.5)
## cmd('remove surf %s extend' % l2s(sa))
## genSlotModel(45, 10, 5.0, 'with name "*BOLT*', ncs=25)
##-----##
sweepVolsInDir([-1.0,0.0,0.0], 0.1, 'with name "*BOLT*")
cmd('mesh vol with name "*BOLT*")
sa = selByArea('in vol with name "*NUTS*', 'EQ', 1130.97, 0.5)
cmd('tweak surface %s offset %s' % (l2s(sa), 1.0))
sl = selByLength('in vol with name "*NUTS*', 'EQ', 50.2655, 0.5)
cmd('curve %s interval 12' % l2s(sl))
sa = selByArea('in vol with name "*NUTS*', 'EQ', 838.169, 0.5)
cmd('surf %s scheme hole rad_intervals 3' % l2s(sa))
cmd('mesh vol with name "*NUTS*")
#####
#NOW LET'S WORK ON THE FLANGE CAP#
#####
flangecap = 'in vol with name "*FLANGE_CAP*'
sa = selByArea('in vol all', 'LE', 120)
cmd('remove surface %s extend' % l2s(sa))
sa = selByArea(flangecap, 'EQ', 5277.875658)
sa1 = selByArea(flangecap, 'EQ', 1884.96, .5)
cmd('tweak surf %s target surf %s' % (l2s(sa), l2s(sa1)))
sl = selByLength(flangecap, 'EQ', 439.823, .5)
ssc = selBySSC(l2s(sl), 2)
cmd('webcut vol with name "*FLANGE_CAP*" sheet extended from surf %s ' %
ssc[0])
sl = selByLength(flangecap, 'EQ', 298.451/2.0, .5)
ssc = selBySSC(l2s(sl), 4)
cmd('webcut vol with name "*FLANGE_CAP*" sheet extended from surf %s ' %
ssc[0])
autoGroupByName(True)
#SET THE INTERVAL AROUND THE BOLT HOLES
cmd('split periodic vol with name "*CAP*")
sl = selByLength(flangecap, 'EQ', 0.5*78.5398, .5)
cmd('curve %s interval 6' % l2s(sl))
cmd('imprint vol with name "*CAP*" ')
cmd('merge vol with name "*CAP*" ')
#FIND THE PLANAR SURFACES IN THE RANGE -102 to -40.5 IN THE X-DIRECTION
sbb = selBoxBound('surface', 'in surface in vol with name "*CAP*', xs=[-102.0,
-40.5])
sstp = selBySType(l2s(sbb), 'plane')
cmd('surf %s scheme hole rad_intervals 3' % l2s(sstp))
#SPLIT A FEW SURFACES TO REMOVE SKEW
sstt = selBySType('in vol with name "*CAP*', 'torus')
cmdLooper('split surface %s skew', sstt)

```

```

sa = selByArea(flangeCap, 'EQ', 5890.49, 0.5)
splitRingSkew(l2s(sa), imp=True)
#SET A SWEEP DIRECTION FOR THE FLANGE CAP
sweepVolsInDir([1.0,0.0,0.0], 0.6, 'in vol with name "*CAP*"')
cmd('match intervals vol with name "*CAP*"')
sbb = selBoxBound('surface', flangeCap, xs=[-42.0, -39.5])
cmd('mesh vol in surface %s' % l2s(sbb))
sbb = selBoxBound('surface', flangeCap, xs=[-102.0, -75.0])
cmd('mesh vol in surface %s' % l2s(sbb))
cmd('mesh vol in group 3')
#####
#NOW WORK ON THE T FLANGE...#
#####
sa = selByArea('in vol with name "*T_FL*"', 'EQ', 39307.6, 0.5)
cmd('webcut vol with name "*T_FL*" with sheet extended from surface %s' %
l2s(sa))
sa = selByArea('in vol with name "*T_FL*"', 'EQ', 37993.6, 0.5)
cmd('webcut vol with name "*T_FL*" with sheet extended from surface %s' %
l2s(sa))
sv = selByVolume('with name "*T_FL*"', 'EQ', 2413020.4, 0.5)
sv1 = selByVolume('with name "*T_FL*"', 'EQ', 252945.2, 0.5)
cmd('unite vol %s' % l2s(sv+sv1))
sa = selByArea('in vol with name "*T_FL*"', 'EQ', 37582.1, 0.5)
cmd('webcut vol with name "*T_FL*" sweep surface %s perpendicular inward
through_all' % l2s(sa))
sv = selByVolume('with name "*T_FL*"', 'EQ', 1726551.0, 0.5)
sl = selByLength('in vol with name "*T_FL*"', 'EQ', 459.854, 0.5)
planeCutWithCurve(l2s(sv), l2s(sl))
sv = selByVolume('with name "*T_FL*"', 'EQ', 1049016.41, 0.5)
sl = selByLength('in vol with name "*T_FL*"', 'EQ', 20.7673, 0.5)
webcutWithCurveNorm(l2s(sv), sl[0], [-1.0,0.0,0.0])
sv = selByVolume('with name "*T_FL*"', 'EQ', 690054.4, 0.5)
webcutWithCurveNorm(l2s(sv), sl[1], [-1.0,0.0,0.0])
sbb = selBoxBound('volume', 'with name "*T_FL*"', xs = [-246.0, -122.0],
zs=[-650.0, -435.0])
cmd('unite vol %s' % l2s(sbb))
sv = selByVolume('with name "*T_FL*"', 'EQ', 977793.8, 0.5)
sl = selByLength('in vol with name "*T_FL*"', 'EQ', 19.7606, 0.01)
webcutWithCurveNorm(l2s(sv), sl[0], [0.0,1.0,0.0])
autoGroupByName(True)
cmd('imprint vol with name "*T_FL*" ')
cmd('merge vol with name "*T_FL*" ')
sa = selByArea('in vol with name "*T_FL*"', 'EQ', 2095.86, 0.5)
cmd('surf %s scheme submap' % l2s(sa))
sweepVolsInDir([0.0,0.0,1.0], 0.707, l2s(selByVolume('with name "*T_FL*"',
'EQ', 774297.3, 0.5)))
sv = selByVolume('in group 5', 'EQ', 677287.8, 0.5)
sweepVolsInDir([1.0,0.0,0.0], 0.707, l2s(sv))
sl = selByLength('in vol in group 5', 'EQ', 78.5398, .5)
cmd('curve %s interval 12' % l2s(sl))
cmd('mesh vol in group 5')
blockByGroups()

```

EXAMPLE 2: MESH MODIFICATION OF THE FLANGE MESH

```

#----- EXAMPLE 2 START -----#
#-- MESH MANIPULATION AND ADDITIONAL MESH FUNCTIONS --#
#OBJECTIVE:
#MODIFY THE T-FLANGE TO ACCOMODATE FOR SMALL DESIGN CHANGES
#CREATE AN INPUT DECK FOR SALINAS AND GENERATE SPIDER ELEMENTS FOR THE BOLTS
#CHECK FOR MESH CONNECTIVITY BEFORE EXPORTING
#OK, FIRST LET'S GET THE SPIDERS MADE
sa = selByArea(flangeCap, 'EQ', 981.748, 0.5)
autoGenSidesets(l2s(sa), True)
sa = selByArea('in vol with name "*T_FL*"', 'EQ', 1963.5, 0.5)
autoGenSidesets(l2s(sa))
autoSpiderSidesets([(3,5)], 200.0)
#CREATE SOME SURFACES TO TWEAK THE MESH TO
#WE ARE GOING TO MODIFY THE OPEN FLANGE SURFACES TO HAVE A CONICAL TAPER
tfl_sc = 'in vol with name "*T_FL*"'
tfl_v = 'with name "*T_FL*"'
sa = selByArea(tfl_sc, 'EQ', 18064.2, 0.5)
sloops1 = getSurfaceLoops(sa[0])
midCurves1 = getC2CMidpoint(l2s(sloops1[0]), l2s(sloops1[1]), 6)
sloops2 = getSurfaceLoops(sa[1])
midCurves2 = getC2CMidpoint(l2s(sloops2[0]), l2s(sloops2[1]), 6)
v1 = vertsFromLocs(midCurves1)
v2 = vertsFromLocs(midCurves2)
c1 = cubit.get_last_id('curve')
cmd('create curve arc three vertex %s full' % l2s(v1[0:3]))
s11 = selByLength('in vol with name "*T_FL*"', 'EQ', 427.25, 0.5)
cc1 = selBoxBound('curve', l2s(s11), xs=[-340, -335])
cmd('create surface skin curve %s %s' % (c1+1, cc1[0]))
s1 = cubit.get_last_id('surface')
c2 = cubit.get_last_id('curve')
cmd('create curve arc three vertex %s full' % l2s(v2[0:3]))
s12 = selByLength('in vol with name "*T_FL*"', 'EQ', 414.69, 0.5)
cc2 = selBoxBound('curve', l2s(s12), zs=[-345, -330])
cmd('create surface skin curve %s %s' % (c2+1, cc2[0]))
s2 = cubit.get_last_id('surface')
#SQUEEZE THE MESH TO AVOID INVERSION
squeezeMesh(sa[1], [0.0, 0.0, 1.0], 0.5)
squeezeMesh(sa[0], [-1.0, 0.0, 0.0], 0.5)
#NOW MOVE THE MESH TO THE NEW SURFACES
sa = selByArea('in vol with name "*T_FL*"', 'EQ', 27862.8, 0.5)
sbb = selBoxBound('surface', l2s(sa), zs = [-345, -342.5])
nodes2Surfs(str(s2), sbb[0], 200)
sa = selByArea('in vol with name "*T_FL*"', 'EQ', 27020.8, 0.5)
sbb = selBoxBound('surface', l2s(sa), xs = [-340.0, -330.0])
nodes2Surfs(str(s1), sbb[0], 200)
#SMOOTH OUT THE VOLUMES
cmd('volume all smooth scheme condition number beta 2 cpu 0.25')
cmd('smooth vol all')
cmd('volume all smooth scheme condition number beta 2 cpu 0.25')
cmd('smooth vol all')
#NOW LET'S CHECK THE CONNECTIVITY
cm = selConMesh(list(cubit.parse_cubit_list('node', 'in vol with name "*T_FL*"')[0])

```

```
#NOW GENERATE BLOCKS FOR SALINAS
cmd('block with name "*SPIDER*" element type bar')
cmd('block with name "*J2G*" element type spring')
cmd('block with name "*J2G*" attribute count 6')
propname = ['KX', 'KY', 'KZ', 'KRX', 'KRY', 'KRZ']
stiff = [1e6, 1e6, 1e6, 1e7, 1e7, 1e7]
cntr = 0
for i in range(0, 6):
    cntr += 1
    cmd('block with name "*J2G*" attribute index %s %s name "%s" ' % (cntr,
stiff[i], propname[i]))
#NOW SET THE MATERIALS FOR THE DECK WRITE-UP
cmd('create material "Steel" property_group "CUBIT-ABAQUS" ')
cmd('modify material "Steel" scalar_properties "MODULUS" 2.86e+07 "POISSON"
0.28 "DENSITY" 0.285 ')
cmd('block 2 to 5 material "Steel"')
genSalinasBlockMats()
#SALINAS WILL NOT KNOW WHAT A SPRING TYPE ELEMENT IS...SO WE NEED TO CHANGE
IT BACK BEFORE EXPORT
cmd('block with name "*J2G*" element type beam')
#CONVERT TO INCHES FROM WHAT IS ASSUMED TO BE MM
cmd('transform mesh output scale 0.0393701')
cmd('export mesh "v2.exo" block all except block in ( block with name "*NUT*"
block with name "*BOLT*" ) qualityfile overwrite')
```