A Domain Language for Constructive Block Topology for Hexa Mesh Generation

***R. Rainsberger[1], Pedro V Marcal[2]**

[1]XYZ Scientific Applications, Inc.,2255 Morello Ave. Suite 220, Pleasant Hill, CA. 94523

[2]MPACT Corp.,5297 Oak bend Lane, Suite 105, Oak Park, CA. 91377

*Presenting and Corresponding author : r.rainsberger@yahoo.com

Abstract

A Topological Domain Language has been developed to aid the **True*Grid*®** beginning user to model a restricted set of problems by simplifying the formation of the hexa block topology. A computer program has been developed to convert scripts in this language to the journal format of **True*Grid*®**.

Keywords : topology, hexa mesh generation, domain language

Introduction

The element type has always been important in determining the accuracy of the results in a Finite Element Analysis. Recent results, Marcal et al[1], Fong et al [2]and Marcal et al [3] have demonstrated the superiority of the hexa 27 fully quadratic element[4]. In order to exploit such an element, we need to be able to generate hexa meshes over a wide spectrum of shapes and sizes. Ideally it would be preferable to generate the hexa meshes automatically. The only such method known to the authors is to generate the mesh via a subdivision of a tetra mesh. Such a procedure results in hexa meshes with poor element quality and an unnecessarily large number of degrees of freedom. The **True*Grid*®** Pre-processor [5] has been developed for hexa mesh generation. This program has many advanced features for the generation of complex meshes. It is not our intention to exercise such advanced features. Instead, we wish to explain the basis of this mesh generator and explore the possibility of developing an expert system which could help a new user generate some useful meshes. The **True*Grid*®** program uses a surface projection method. This method in its simplest form defines a block in 3D space and in a unit topological space, respectively. The block in 3D space is in turn mapped to an enclosing surface. Here we restrict the enclosing surface to those resulting from simple geometric solids. This allows us to develop a simple Domain Language.

We introduce the projection method by creating the simplest 3 X 3 X 3 unit topology and using this to create a mesh for a cylinder. The result is shown in Fig. 1. The **True*Grid*®** Journal for these problems is given in Appendix A.

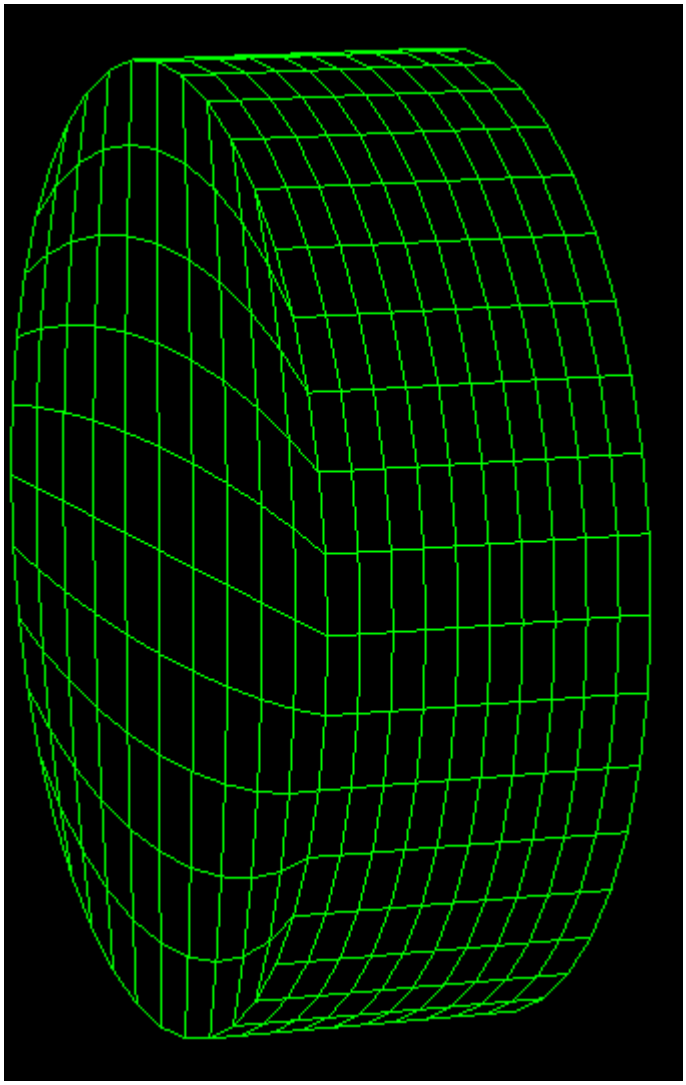


Fig. 1, Hexa mesh for cylinder, simple topology

The problem with such a mesh is that the quality of the elements at the edges of the topological blocks is poor.

We can improve this by introducing a cross pattern for the topology and requiring that the adjacent faces of the cross be joined for the projected model as shown in Fig. 2.

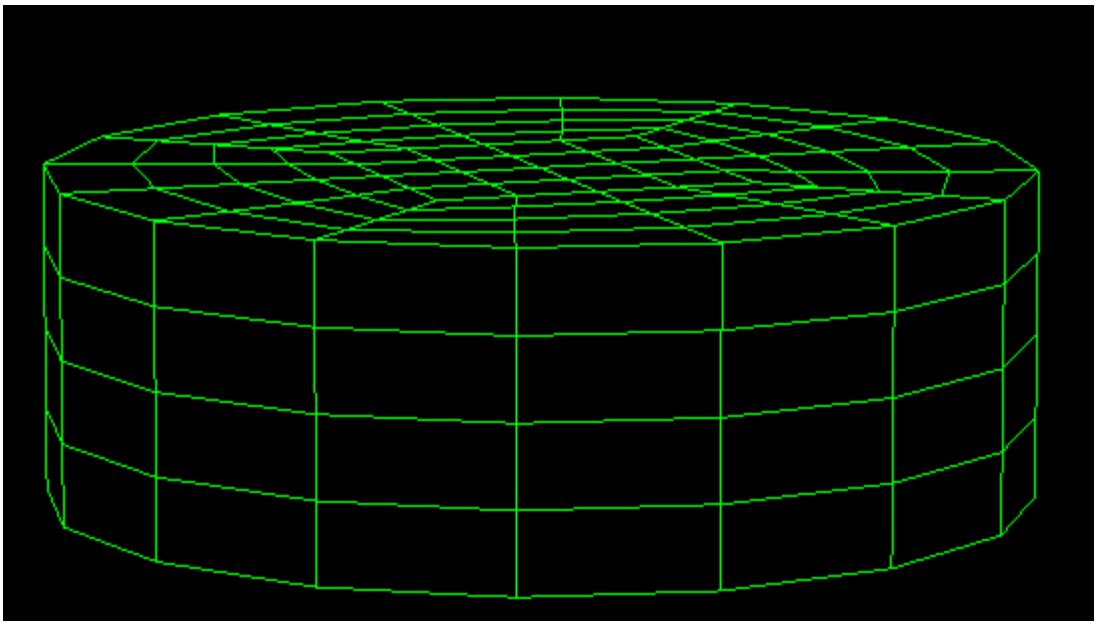
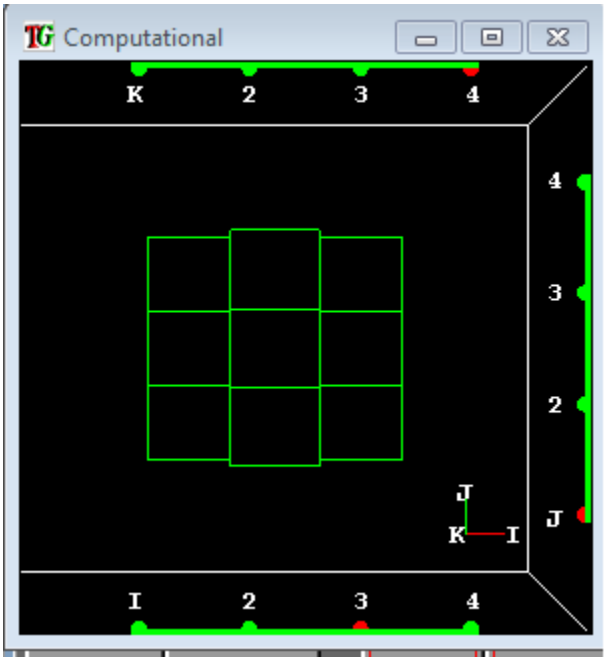


Fig. 2 a , Hexa mesh with cross topology



Frig. 2 b, Cross pattern in topological space

We can now see that the problem with the element quality has disappeared.

Finally we generalize this concept by introducing the cruciform topology in three directions. This is used to project to sphere and sphere like geometries.

This is shown in Fig. 3

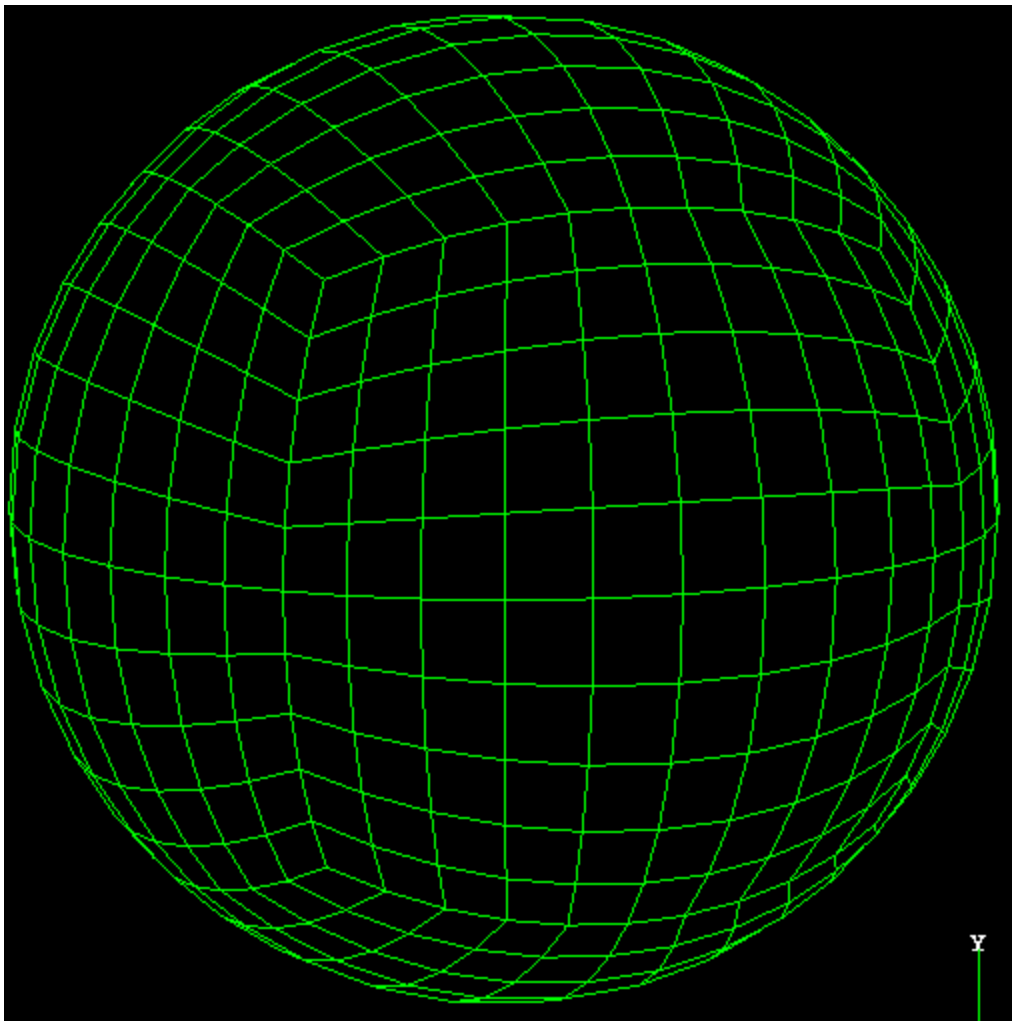


Fig. 3 a, Hexa Mesh for sphere with Cruciform topology

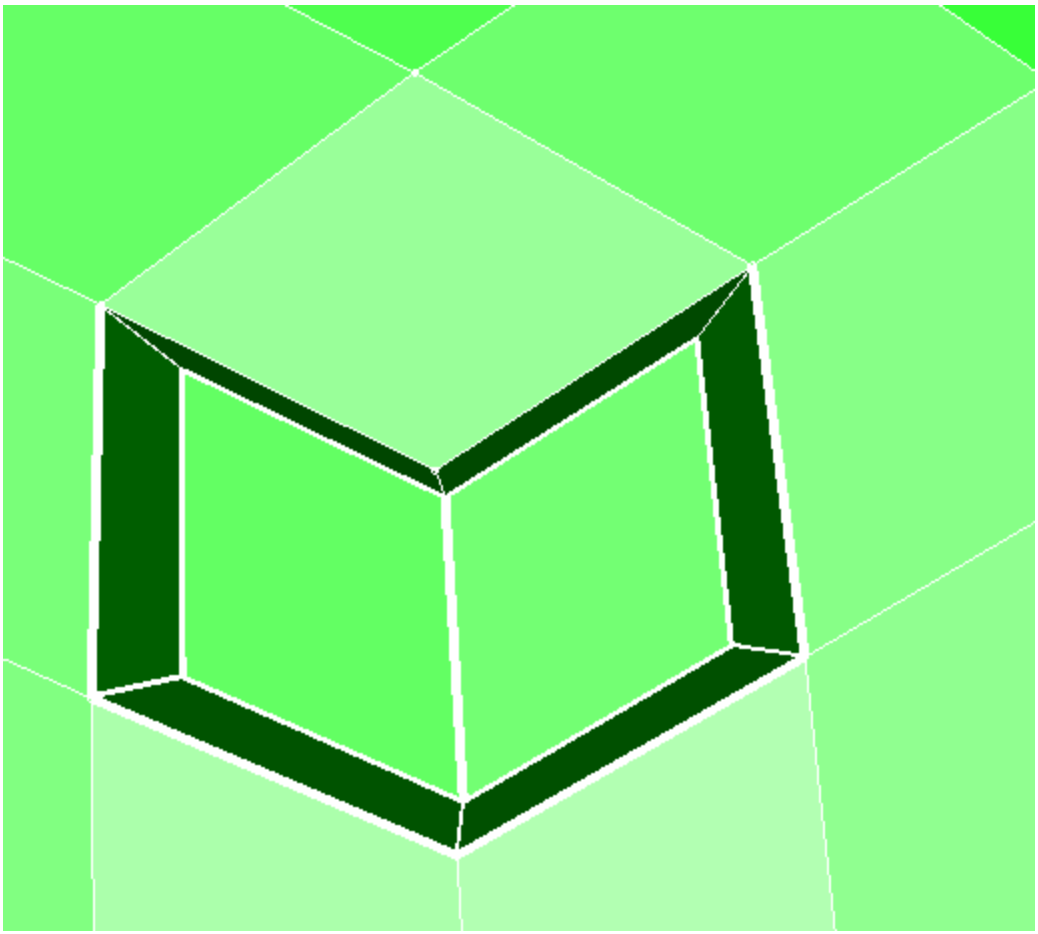


Fig. 3 b , Projection of the corner with three adjoining elements (two hidden)

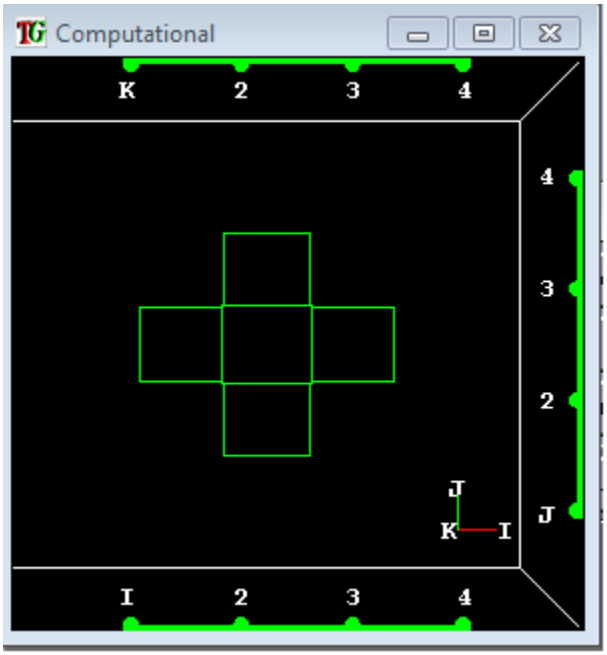


Fig 3 c, Cruciform topology

We have now established the basic requirements for the creation of a topology that can be used to project to the boundary surfaces to create hexa meshes via the projection method.

Theoretical Considerations

In the projection method, we first develop the topology in a three dimensional integer space. Our topology is then described here as a sequence of blocks. The blocks are joined together in a topologically constructive process. Because blocks are the only basic topology admitted in our process, the constructive topology may be restricted to a few simple operations. Here we adopt the concepts and the procedures used by **True*Grid*®**. However, to simplify our task, we will create a simple Topological Domain Language (TDL).

Naturally, in order to take advantage of the existing **True*Grid*®** software, we must also define a method to map the defined TDL operations to a language that the **True*Grid*®** can understand. This we achieve by mapping our operations into the parametric journal language currently used by **True*Grid*®**. The TDL is defined so that it can be easily translated from the TDL to the **True*Grid*®** Scripting Language.

In order to create a general purpose TDL, we have identified the following functions for the language. The functions that are preceded by a required name are the functions that will perform the translation to equivalent terms in the **True*Grid*®** journal.

Finite Element Format

name=FemFormat(mpact)

FemOutput(mpact)

or

Mesh

Coordinate System

Crd(3D||TP,(list i),(list j),(list k))

Crdr(3D||TP, (range i),(range j),(range k))

Crdf(3D,(min i,min j,min k),(max i,max j,max k))

Crdfi(3D,(min i,max i),(min j,max j),(min k,max k)) # see directions below in art. Translation to **True*Grid*®** Script.

CrdfPos(3D,(pos min i,pos max i),(pos min j,pos max j),(pos min k,pos max k)) # the values corresponds to count in the coordinates of the block definition list

SubPos(i or j or k,(min0,max0),(min1,max1)) # use node numbers

CrdfsPerm(SubPos0,SubPos1,SubPos2) # up to three SubPos can be included, permute the subpos between the two of i j k directions. This allows us to change the number of nodes in a block.

CrdfhPerm(SubPos) # is used as an Insert Sub Block (insprt) to add subdivisions to a direction of a block in preparation for creating a part. A negative value indicates insertion to the left of the min index. The max units are in elements (ie nodes+1)

CrdfPermute(TP,(i-min,i-max),(j-min,j-max),(k-min,k_max)) # when more pairs are used in a dir. Add 0 as a separator. Use (:) to indicate the full range in a particular direction.

Block Creation

The basic building block is created by defining a rectangular block in 3D space and its equivalent in topological space.

name=Block(Crd in 3D,Crd in TP)

name=SubBlock(Block, Crd of Sub_block)

name=Cross(Block start, CrdfPermute)

name=Cruciform(Block start,CrdfPermute,CrdfPermute) # two orthogonal CrdfPermutes

name=TieFace(Crdf, tie-#)

name=MapBlock(Sub_block , Block)

name=EndBlock(Block) # end operations for the named Block

Utilities and other useful quantities

Vec((i0,j0,k0),(i1,j1,k1))

VecAdd(vec1,vec2)

VecSub(vec1,vec2)

Dir(0 or 1, 0 or 1,0 or 1)

Pos(i, j,k)

DisplayWindow(x,y)

Block Editing

BlockTranslate(Block,Vec)

BlockRotate(Block,VecDir, num rt angles)

Block Boolean Operations

Union(Block 1, Block 2)

Subtract(Block 1, Block 2)

Intersection(Block 1, Block 2)

InsertSubBlock(SubBlock)

Slice(Block,VecPos,normal to VecDir) # result is two blocks _p and _m added to block name

DeleteBlock(Block)

Topological Properties for Surface Projection

Faces(Crdf) # list corresponds to position on list of block definition

name=Project(Faces,Geom)

Geometric Surfaces For Blocks

Sph(center,rad)

Cyl(center,dir vector)

Tor(center,torus rad, Center,Major rad)

Con(center,dir vector,end rad, other end radius)

Projection Operations to Surfaces

name=BlockSurface(Crd,geometric surfaces) # from block to surface

Insertion of a Sub-Block

The operation of inserting a block into a current block is about the most complex process that we will address with our TDL. Because of the number of operations involved, we have to be systematic about the procedure. We assume that the current block has been defined in the usual way. Because we need to increase the number of elements around our point of insertion, we need to do so consistently because of the block nature of our elements. To be specific, we will use the example of a cylinder joined to a hemisphere and insert another cylinder with its axis in the y direction. This is actually the example we use in our case 2 discussions.

It is listed here before the insertion

prog=FemFormat(mpact)

coord_1=Crdl(3D,(1 6 11 16),(1 6 11 16),(1 6 11 16))

coordt=Crd(TP,(0 12 18 18),(-6 -6 6 6),(-6 -6 6 6))

blk1=Block(coord_1,coordt)

create holes in topology for better elements about spheres and cylinders.

crdfp5=CrdfPermute(TP,(3 ,4),(1, 2 ,0 ,3, 4),(2,3))

crdfp6=CrdfPermute(TP,(:),(1, 2 ,0 ,3, 4),(1 ,2, 0 ,3 ,4))

```
crdfp7=CrdfPermute(TP,( 3,4 ),(2,3),(1 ,2, 0 ,3 ,4))
cruci=Cruciform(blk1,crdfp6,crdfp5,crdfp7)
crdfp8=CrdfPermute(TP,(1,3),(2,3),(2,3))
cross5=Cross(blk1,crdfp8)
# Projection of block face to geometry faces.
orig9=Pos(12,0,0)
rad9=Rad(5)
sph1=Sph(orig9,rad9)
faces8=Crdfi(3D,(2,-3),(-2,-3),(-2,-3))
proj1=Project(faces8,sph1)
orig6=Pos(12,0,0)
rad6=Rad(6)
sph2=Sph(orig6,rad6)
faces7=Crdfi(3D,(2,-4),(-1,-4),(-1,-4))
proj2=Project(faces7,sph2)
orig=Pos(0,0,0)
dir=Dir(1,0,0)
rad=Rad(5)
cyl1=Cyl(orig,dir,rad)
faces1=Crdfi(3D,(1,2),(-2,-3),(-2,-3))
proj=Project(faces1,cyl1)
orig1=Pos(0,0,0)
dir1=Dir(1,0,0)
rad1=Rad(6)
cyl2=Cyl(orig1,dir1,rad1)
faces=Crdfi(3D,(1,2),(-1,-4),(-1,-4))
proj3=Project(faces,cyl2)
```

```
# insert Crdfh here
sub5=SubPos(k,(2,4))
crdfh1=CrdfhPerm(sub5)
sub6=SubPos(k,(3,7))
crdfh2=CrdfhPerm(sub6)
sub7=SubPos(i,(1,5))
crdfh3=CrdfhPerm(sub7)
sub8=SubPos(i,(2,5))
crdfh4=CrdfhPerm(sub8)
crdfp9=CrdfPermute(TP,(2,3),(3,4),(3,4))
cross6=Cross(blk1,crdfp9)
# insert Cfrds here
orig3=Pos(6,0,0)
dir3=Dir(0,1,0)
rad3=Rad(2)
cyl3=Cyl(orig3,dir3,rad3)
faces3=Crdfi(3D,(-2,-3)(3,4),(-3,-4))
proj5=Project(faces3,cyl3)
# insert hole and map 4 faces to the defined surface
crdf1=Crdf(3D,(2, 3, 3),( 2, 4, 4 ))
crdf2=Crdf(3D,(3, 3, 3),( 3, 4, 4 ))
crdf3=Crdf(3D,(2, 3,3),( 3, 4, 3 ))
crdf4=Crdf(3D,(2, 3, 4),( 3, 4, 4 ))
# insert tie definitions
tie1=TieFace(crdf1,1)
tie2=TieFace(crdf3,2)
tie3=TieFace(crdf2,3)
tie4=TieFace(crdf4,4)
Pause
```

endb1=EndBlock(blk1)

1. We now discuss the coding in the original block that implements the hole that allows the insert to be joined.

We now expand the number of elements. We make use of the CrdfsPerm function. This allows us to change the number of nodes in a block. We insert the following after the Block definition.

```
sub1=SubPos(i,(1,5))

crdfs=CrdfsPerm(sub1)

sub2=SubPos(j,(2,10))

sub3=SubPos(k,(2,10))

sub4=SubPos(i,(1,5))

crdfs1=CrdfsPerm(sub2,sub3,sub4)
```

2. Create nodes and topology for new elements created above. Use function CrdfhPerm. The new nodes and topology are updated and printed to the screen, so that they can help in defining the insert block later. These appear as shown below, respectively.

```
*** PrintList *** new_block_3d

 [1, 6, 11, 16, 21, 26]   [1, 6, 21, 26]   [1, 6, 10, 17, 21, 26]

 *** PrintList *** new_block_tp

 [0, 1, 2, 12, 18, 18]   [-6, -6, 6, 6]   [-6, -6, -5, -4, 6, 6]


sub5=SubPos(k,(2,4))
crdfh1=CrdfhPerm(sub5)
sub6=SubPos(k,(3,7))
crdfh2=CrdfhPerm(sub6)
sub7=SubPos(i,(1,5))
crdfh3=CrdfhPerm(sub7)
sub8=SubPos(i,(2,5))
crdfh4=CrdfhPerm(sub8)
```

3. Make a hole to allow the new block to be inserted. The following creates a hole and maps it to the surface of the shape of the insert. We note from the cylindrical surface being used that its origin is (6,0,0) and its radius is 2.

```
crdfp9=CrdfPermute(TP,(2,3),(3,4),(3,4))

cross6=Cross(blk1,crdfp9)
```

```
orig3=Pos(6,0,0)

dir3=Dir(0,1,0)

rad3=Rad(2)

cyl3=Cyl(orig3,dir3,rad3)

faces3=Crdfi(3D,(-2,-3)(3,4),(-3,-4))

proj5=Project(faces3,cyl3)
```

4. Label the 4 surfaces to be tied to the corresponding surfaces in the Sub-Block to be created.

```
crdf1=Crdf(3D,(2, 3, 3),( 2, 4, 4 ))

crdf2=Crdf(3D,(3, 3, 3),( 3, 4, 4 ))

crdf3=Crdf(3D,(2, 3,3),( 3, 4, 3 ))

crdf4=Crdf(3D,(2, 3, 4),( 3, 4, 4 ))

tie1=TieFace(crdf1,1)

tie2=TieFace(crdf3,2)

tie3=TieFace(crdf2,3)

tie4=TieFace(crdf4,4)
```

5. Now we are in a position to create the Sub-Block**.** We first terminate the specification for the first block. Then we define the block that will become the cylindrical insert. This block must have its nodal coordinates coincide with those of the cylindrical hole. (**True*Grid*®** only requires this to be close. It has eight different ways of making the hole and insert equal exactly.) The nodes (4,9),(1,6),(4,11) are chosen to coincide exactly with the nodes (6,11),(21,26),(10,17) of the first block respectively. With reference to the topology of the insert, we are free to choose a new reference system as long as its core coordinates are consistent with its nodal coordinates and its positioning on the original block, blk1. For the x direction we choose (5,5,7,7) as required by the symmetry for the cylinder and with the cylinder origin at 6. For the y direction we choose (5,6,10) because the first two indices coincide with the topology of the hole and cylinder in the first block. Finally in the z direction we choose (-1,-1,1,1) to give the required symmetry for the cylinder. Note that **True*Grid*®** looks for key points in the insert to match the two blocks. When it does not find it, it defaults to using the nodal coordinates. That is why only the symmetry is defined here since the hole origin and depth to be filled are already defined by the x and y topology, respectively.

```
endb1=EndBlock(blk1)
```

```
coord_2=Crdl(3D,(1 4 9 12),(1 6 16),( 1 4 11 14 ))
coordt2=Crd(TP,(5 5 7 7) ,(5 6 10),(-1 -1 1 1))
blk2=Block(coord_2,coordt2)
```

6. Now we define the topology holes and the surfaces to be tied.

```
crdfp10=CrdfPermute(TP,(1, 2 ,0 ,3, 4),( : ),(1 ,2, 0 ,3 ,4))
cross7=Cross(blk2,crdfp10)
crdfp11=CrdfPermute(TP,(2,3),( : ),(2,3))
cross8=Cross(blk2,crdfp11)
crdf5=Crdf(3D,( 1, 1, 2),( 1, 2, 3))
crdf6=Crdf(3D,(2, 1, 1),( 3, 2, 1))
crdf7=Crdf(3D,( 4, 1, 2),( 4, 2, 3))
crdf8=Crdf(3D,( 2, 1, 4),( 3, 2, 4))
tie5=TieFace(crdf5,1)
tie7=TieFace(crdf7,3)
tie6=TieFace(crdf6,2)
tie8=TieFace(crdf8,4)
orig4=Pos(6,0,0)
dir4=Dir(0,1,0)
rad4=Rad(2)
cyl4=Cyl(orig4,dir4,rad4)
faces4=Crdfi(3D,(-1,-4),(2, 3),(-1,-4))
proj6=Project(faces4,cyl4)
```

7. Finally we create the internal surface of the insert, namely another cylinder. Then end the part and merge the two blocks, concluding the project to insert a sub-block. The diagram for the insert block is shown as Fig. 6, when we discuss the case study for the cylinder with insert.

```
orig5=Pos(6,0,0)
dir5=Dir(0,1,0)
rad5=Rad(1)
cyl5=Cyl(orig5,dir5,rad5)
```

faces5=Crdfi(3D,(-2,-3),(:),(-2,-3))

proj7=Project(faces5,cyl5)

DisplayWindow(10,20)

Pause

endb1=EndBlock(blk2)

Merge

Mesh

Translation to True*Grid*® Script

The objective of the project was to develop a Domain Language with simple concepts that may subsequently be used in an expert system. In the above we have used the concept of Constructive Solid Geometry as our guide in defining a constructive Topological model for the Projection method in **True*Grid*®**. In the process we have simplified our model to first define a mapping from the Unit Topological Space to a Block indexed cube in 3D space. This block indexed cube is in turn projected to an enclosing surface space defined by simple engineering solids. By this process we have restricted the full capabilities of the **True*Grid*®** program. However, we are of the opinion that the domain covers a significant spectrum of mesh generation problems that it may be of interest to most analysts wanting to generate hexa meshes. A python program (TDL2TG.py) was written to translate the TDL script to the **True*Grid*®** journal script.

This program was first used to translate scripts used to generate the basic cases discussed for cylinders and spheres with simple topology and also with cross and cruciform topology respectively. These examples are listed in Appendix A.

The surfaces or faces of each block are defined by six faces which in our case are projected to the geometric figure specified by the sf index.

Each face is specified by sf defined by the indices
sf = i-min j-min k-min; i-max j-max k-max; sd* # =Crdf in our notation
Instead of specifying 6 faces, we introduce a short-hand indicial notation. We note that the back and front face is specified by -ve and +ve prefixes respectively. However we will visit each axis direction in turn and allow two index values *_min *_max. A - in front of both indices mean a range with either a back or front face. A single negative index selects only that face. Meanwhile the negative signs are ignored in the other two axis directions. The values there define the range to be combined with the active axis direction being processed.
We have the face indicial notation,
sfi = -//+ i-min -//+ i-max, -//+ j-min -//+ j-max, -//+ k-min -//+ k-max; sd * # Crdfi our notation
As an example of the sphere in SC2i.tg we have
block 1 6 16 21; 1 6 16 21; 1 6 16 21; -1 -1 1 1; -1 -1 1 1; -1 -1 1 1;
sfi = -1 -4; -1 -4; -1 -4; sd 1
To deal with a hemisphere, we only specify 5 faces, and to have same size mesh along the 3 axis,
block 1 6 16 21; 1 6 11; 1 6 16 21; -1 -1 1 1; 0 1 1; -1 -1 1 1;

sfi = -1 -4; 1 -3; -1 -4; sd 1
The top face in the j direction is selected by the -3 in the j-direction.

Case Studies

1. Case Study: Hexa mesh for Cylindrical Pressure Vessel with Hemispherical Closure

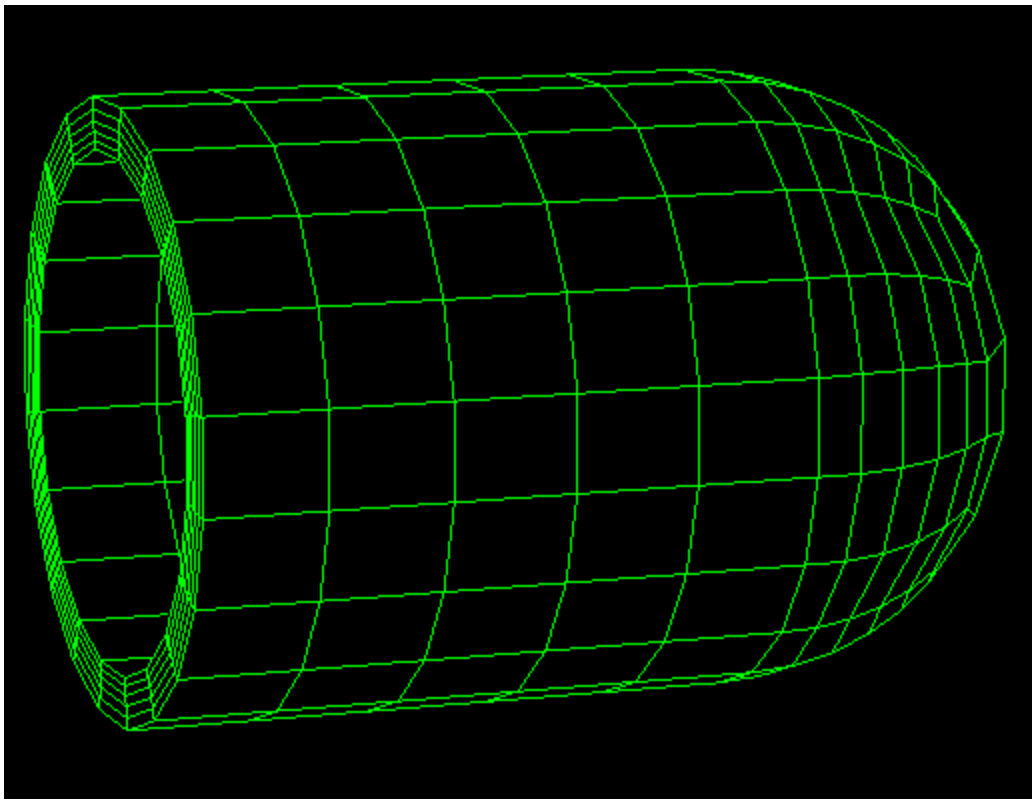


Fig. 4 a, Hexa Mesh for Pressure vessel

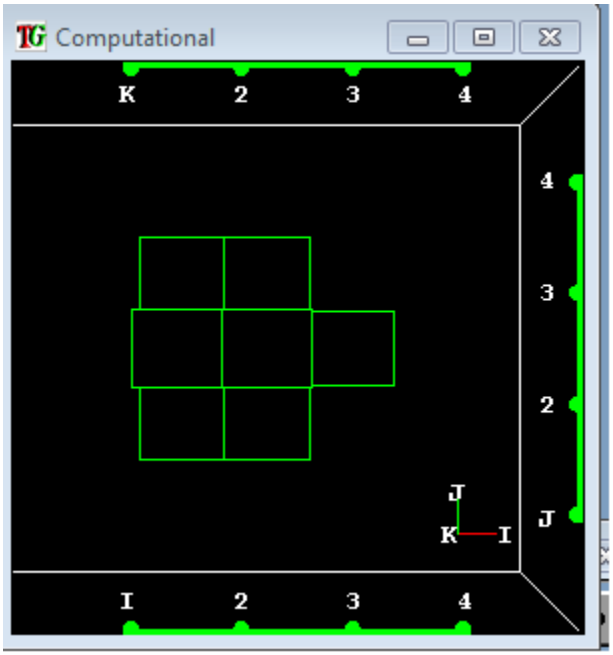


Fig. 4 b, Combined Topology for Cylinder and Sphere (half)

2. Case Study: Insertion of a Cylindrical Nozzle into Pressure Vessel.

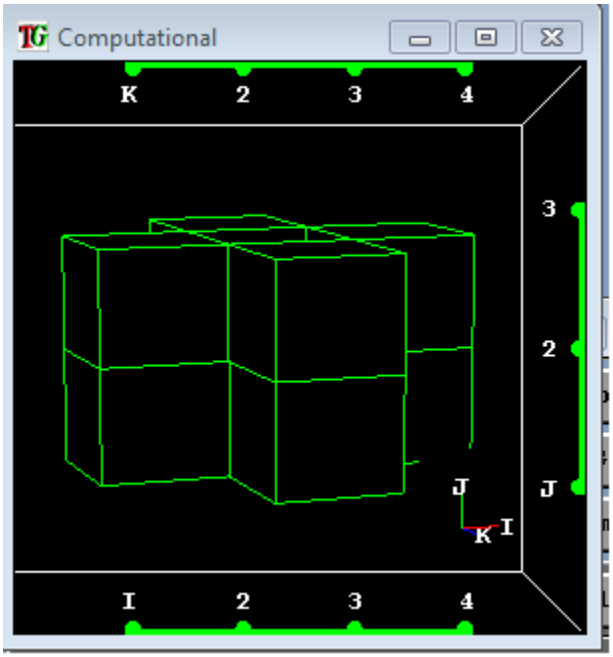


Fig. 5 a, Topology for Cylindrical Part before Insert

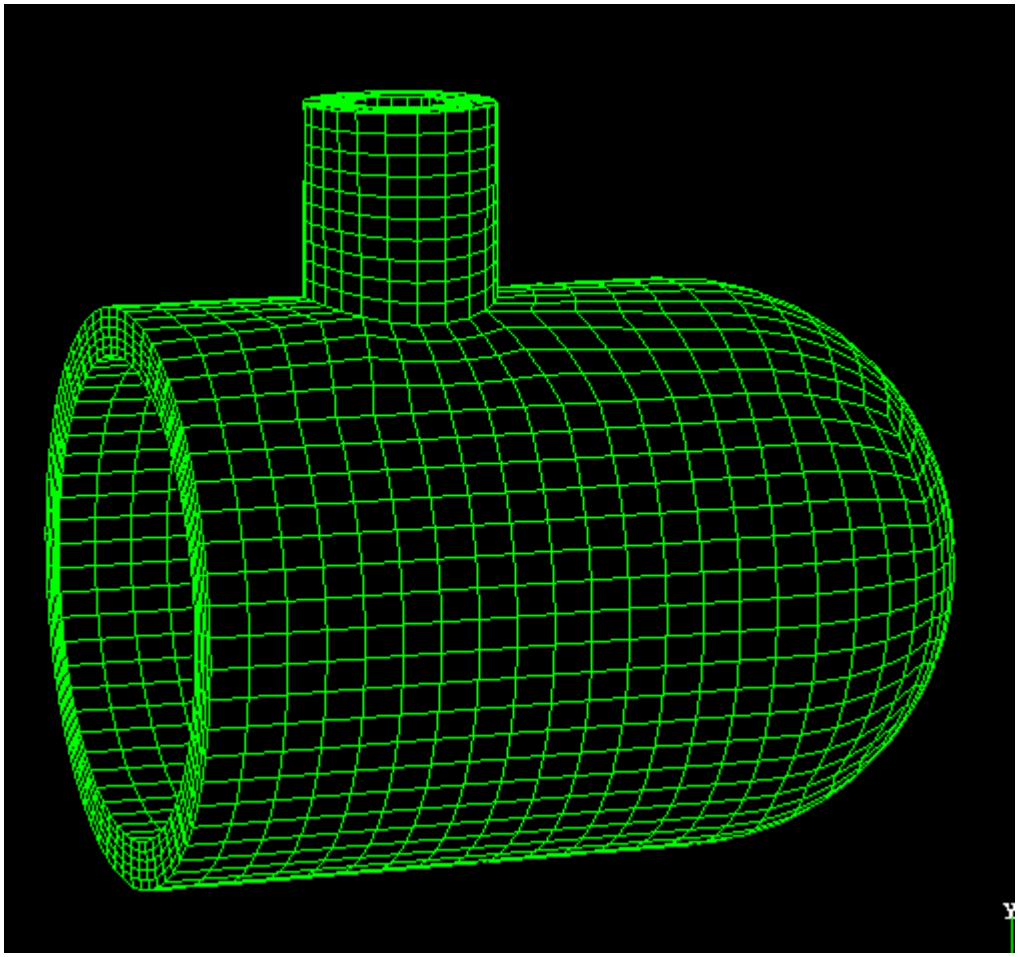


Fig. 5 b , Hexa Mesh for Pressure Vessel With Nozzle Insert

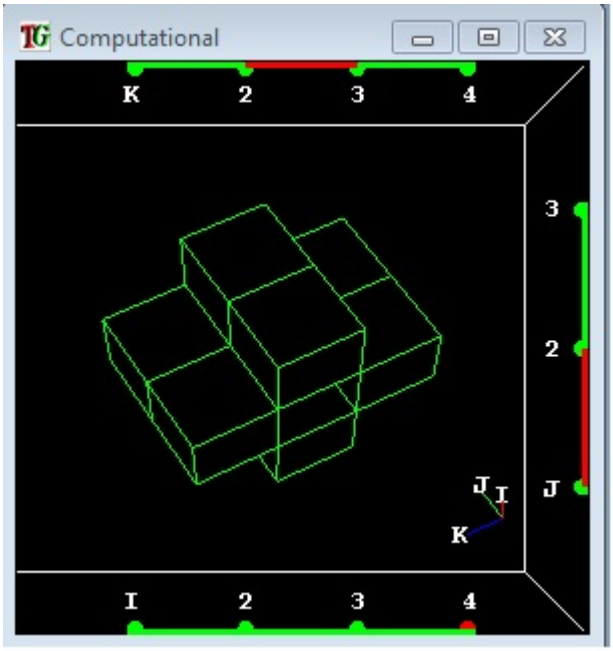


Fig. 6a Topology for insert. Same as fig. 5A but with different axis orientation.

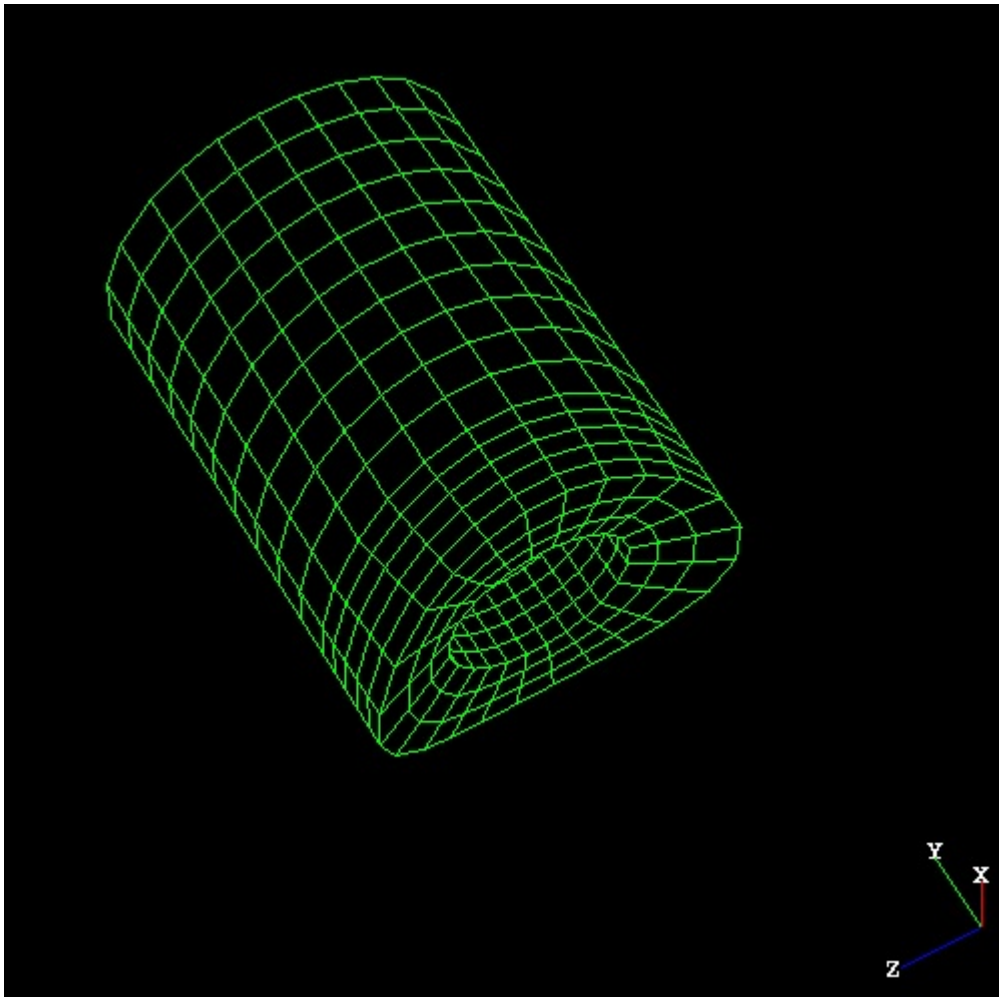


Fig. 6b Mesh for cylindrical insert.

Conclusions

A topological Domain Language has been developed to assist in the use of the **True*Grid*®** Hexa mesh generator. A Python program has been developed to convert scripts in the Domain Language to the journal format in **True*Grid*®**. The Language is useful in cases where the CAD model is created with Constructive Solid Modeling.

References

[1] P.V. Marcal, J.T. Fong, R. Rainsberger, L. Ma, Finite Element Analysis of a Pipe Elbow Weldment Creep-Fracture Problem Using an Extremely Accurate 27-node Tri-Quadratic Shell and Solid Element Formulation, Proc. 14th International Conf. on Pressure Vessel Technology, ICPVT-14, Sep. 23-26, 2015, Shanghai, China, 2015.

[2] J.T. Fong, J.J. Filliben, N.A. Heckert, P.V. Marcal, R. Rainsberger, L. Ma, Uncertainty Quantification and Extrapolation of Finite Element Method-estimated Stresses in a Cracked Pipe Elbow Weldment using a Logistic

Function Fit and a Nonlinear Least Square Algorithm, Proc. International Conf. on Pressure Vessel Technology, ICPVT-14, Sep. 23-26, Shanghai, China, 2015.

[3]P.V. Marcal, J.T. Fong, R. Rainsberger, L. Ma, High Accuracy Approach To Finite Element Analysis Using the Hexa 27-node Element, Proc. PVP-2016, July 17-21, Vancouver B.C., Canada, 2016

[4] P.V. Marcal, MPACT User Manual, Mpact Corp. , Oak Park, CA, 2001.

[5] R. Rainsberger, **True*Grid*®** User's Manual: A Guide and a Reference, Volumes I, II, and III, Version 3.0.0. Published by XYZ Scientific Applications, Inc., Pleasant Hill, CA 94523 , 2014, www.truegrid.com/pub/TGMAN300.1.pdf

APPENDIX A: EXAMPLE SCRIPTS

SC1.tdl

```
prog=FemFormat(mpact)
coord_1=Crdr(3D,11,11,11)
coordt=Crd(TP,(-1,1),(-1,1),(-1,1))
blk1=Block(coord_1,coordt)
orig=Pos(0,0,0)
dir=Dir(1,0,0)
cyl=Cyl(orig,dir)
faces=Crdf(3D,(1,2),(-1,-2),(-1,-2))
proj=Project(faces,cyl)
Pause
DisplayWindow(10,20)
Pause
endb1=EndBlock(blk1)
Merge
Mesh
```

SC1.tg, output from TDL2TG.py, see Fig. 1

```
mpact
block 1 11; 1 11; 1 11;-1 1 -1 1 -1 1
sd 1 cy 0 0 0 1 0 0 3
sfi 1 2; -1 -2; -1 -2; sd 1
interrupt
ry 20 rx 10
```

center

disp

interrupt

endpart

merge

write

spcy.tdl for Case Study of Cylindrical Pressure Vessel with Hemispherical Closure

```
prog=FemFormat(mpact)
coord_1=Crdl(3D,(1 6 16 21),(1 6 16 21),(1 6 16 21))
coordt=Crd(TP,(-1 -1 1 1),(-2 0 1 1),(-1 -1 1 1))
blk1=Block(coord_1,coordt)
crdfp5=CrdfPermute(TP,(1, 2 ,0 ,3, 4),(3,4),( : ))
crdfp6=CrdfPermute(TP,(1, 2 ,0 ,3, 4),( : ),(1 ,2, 0 ,3 ,4))
crdfp7=CrdfPermute(TP,( : ),(3,4),(1 ,2, 0 ,3 ,4))
cruci=Cruciform(blk1,crdfp5,crdfp6,crdfp7)
crdfp8=CrdfPermute(TP,(2,3),(1,3),(2,3))
cross5=Cross(blk1,crdfp8)
orig5=Pos(0,0,0)
rad5=Rad(3)
sph1=Sph(orig5,rad5)
faces5=Crdfi(3D,(-1,-4),(2,-4),(-1,-4))
proj1=Project(faces5,sph1)
orig6=Pos(0,0,0)
rad6=Rad(2)
sph2=Sph(orig6,rad6)
faces6=Crdfi(3D,(-2,-3),(2,-3),(-2,-3))
proj2=Project(faces6,sph2)
orig=Pos(0,0,0)
dir=Dir(0,1,0)
rad=Rad(3)
```

```
cyl1=Cyl(orig,dir,rad)
faces=Crdfi(3D,(-1,-4),(1,2),(-1,-4))
proj=Project(faces,cyl1)
orig1=Pos(0,0,0)
dir1=Dir(0,1,0)
rad1=Rad(2)
cyl2=Cyl(orig1,dir1,rad1)
faces1=Crdfi(3D,(-2,-3),(1,2),(-2,-3))
proj3=Project(faces1,cyl2)
Pause
DisplayWindow(10,20)
Pause
endb1=EndBlock(blk1)
Merge
Mesh
```

spcy_td.tg output for Case Study of Pressure Vessel, see Fig. 4

```
mpact
block 1 6 16 21 ; 1 6 16 21 ; 1 6 16 21 ; -1 -1 1 1 ;-2 0 1 1 ;-1 -1 1 1 ;
dei 1 2 0 3 4 ;3 4 ; ;
dei 1 2 0 3 4 ; ;1 2 0 3 4 ;
dei  ;3 4 ;1 2 0 3 4 ;
dei 2 3 ;1 3 ;2 3 ;
sd 1 sp 0 0 0 3
sfi -1 -4; 2 -4; -1 -4;  sd 1
sd 2 sp 0 0 0 2
sfi -2 -3; 2 -3; -2 -3;  sd 2
sd 3 cy 0 0 0 0 1 0 3
sfi -1 -4; 1 2; -1 -4;  sd 3
sd 4 cy 0 0 0 0 1 0 2
sfi -2 -3; 1 2; -2 -3;  sd 4
interrupt
```

ry 20 rx 10

center

disp

interrupt

endpart

merge

write

spcys2.tdl for Case Study with insert.

```
prog=FemFormat(mpact)
coord_1=Crdl(3D,(1 6 11 16),(1 6 11 16),(1 6 11 16))
coordt=Crd(TP,(0 12 18 18),(-6 -6 6 6),(-6 -6 6 6))
blk1=Block(coord_1,coordt)
sub1=SubPos(i,(1,5))
crdfs=CrdfsPerm(sub1)
sub2=SubPos(j,(2,10))
sub3=SubPos(k,(2,10))
sub4=SubPos(i,(1,5))
crdfs1=CrdfsPerm(sub2,sub3,sub4)
crdfp5=CrdfPermute(TP,(3 ,4),(1, 2 ,0 ,3, 4),(2,3))
crdfp6=CrdfPermute(TP,( : ),(1, 2 ,0 ,3, 4),(1 ,2, 0 ,3 ,4))
crdfp7=CrdfPermute(TP,( 3,4 ),(2,3),(1 ,2, 0 ,3 ,4))
cruci=Cruciform(blk1,crdfp6,crdfp5,crdfp7)
crdfp8=CrdfPermute(TP,(1,3),(2,3),(2,3))
cross5=Cross(blk1,crdfp8)
orig9=Pos(12,0,0)
rad9=Rad(5)
sph1=Sph(orig9,rad9)
faces8=Crdfi(3D,(2,-3),(-2,-3),(-2,-3))
```

```
proj1=Project(faces8,sph1)
orig6=Pos(12,0,0)
rad6=Rad(6)
sph2=Sph(orig6,rad6)
faces7=Crdfi(3D,(2,-4),(-1,-4),(-1,-4))
proj2=Project(faces7,sph2)
orig=Pos(0,0,0)
dir=Dir(1,0,0)
rad=Rad(5)
cyl1=Cyl(orig,dir,rad)
faces1=Crdfi(3D,(1,2),(-2,-3),(-2,-3))
proj=Project(faces1,cyl1)
orig1=Pos(0,0,0)
dir1=Dir(1,0,0)
rad1=Rad(6)
cyl2=Cyl(orig1,dir1,rad1)
faces=Crdfi(3D,(1,2),(-1,-4),(-1,-4))
proj3=Project(faces,cyl2)
sub5=SubPos(k,(2,4))
crdfh1=CrdfhPerm(sub5)
sub6=SubPos(k,(3,7))
crdfh2=CrdfhPerm(sub6)
sub7=SubPos(i,(1,5))
crdfh3=CrdfhPerm(sub7)
sub8=SubPos(i,(2,5))
crdfh4=CrdfhPerm(sub8)
crdfp9=CrdfPermute(TP,(2,3),(3,4),(3,4))
cross6=Cross(blk1,crdfp9)
orig3=Pos(6,0,0)
```

```
dir3=Dir(0,1,0)
rad3=Rad(2)
cyl3=Cyl(orig3,dir3,rad3)
faces3=Crdfi(3D,(-2,-3)(3,4),(-3,-4))
proj5=Project(faces3,cyl3)
crdf1=Crdf(3D,(2, 3, 3),( 2, 4, 4 ))
crdf2=Crdf(3D,(3, 3, 3),( 3, 4, 4 ))
crdf3=Crdf(3D,(2, 3,3),( 3, 4, 3 ))
crdf4=Crdf(3D,(2, 3, 4),( 3, 4, 4 ))
tie1=TieFace(crdf1,1)
tie2=TieFace(crdf3,2)
tie3=TieFace(crdf2,3)
tie4=TieFace(crdf4,4)
Pause
endb1=EndBlock(blk1)
coord_2=Crdl(3D,(1 4 9 12),(1 6 16),( 1 4 11 14 ))
coordt2=Crd(TP,(5 5 7 7) ,(5 6 10),(-1 -1 1 1))
blk2=Block(coord_2,coordt2)
crdfp10=CrdfPermute(TP,(1, 2 ,0 ,3, 4),( : ),(1 ,2, 0 ,3 ,4))
cross7=Cross(blk2,crdfp10)
crdfp11=CrdfPermute(TP,(2,3),( : ),(2,3))
cross8=Cross(blk2,crdfp11)
crdf5=Crdf(3D,( 1, 1, 2),( 1, 2, 3))
crdf6=Crdf(3D,(2, 1, 1),( 3, 2, 1))
crdf7=Crdf(3D,( 4, 1, 2),( 4, 2, 3))
crdf8=Crdf(3D,( 2, 1, 4),( 3, 2, 4))
tie5=TieFace(crdf5,1)
tie7=TieFace(crdf7,3)
tie6=TieFace(crdf6,2)
```

```
tie8=TieFace(crdf8,4)
orig4=Pos(6,0,0)
dir4=Dir(0,1,0)
rad4=Rad(2)
cyl4=Cyl(orig4,dir4,rad4)
faces4=Crdfi(3D,(-1,-4),(2, 3),(-1,-4))
proj6=Project(faces4,cyl4)
orig5=Pos(6,0,0)
dir5=Dir(0,1,0)
rad5=Rad(1)
cyl5=Cyl(orig5,dir5,rad5)
faces5=Crdfi(3D,(-2,-3),( : ),(-2,-3))
proj7=Project(faces5,cyl5)
DisplayWindow(10,20)
Pause
endb1=EndBlock(blk2)
Merge
Mesh
```

spcys2_td.tg output from Case Study, see figs. 5-6.

```
mpact
block 1 6 11 16 ; 1 6 11 16 ; 1 6 11 16 ; 0 12 18 18 ;-6 -6 6 6 ;-6 -6 6 6 ;
lmseq i  1 5
lmseq j  2 10  lmseq k  2 10  lmseq i  1 5
dei  ;1 2 0 3 4 ;1 2 0 3 4 ;
dei 3 4 ;1 2 0 3 4 ;2 3 ;
dei 3 4 ;2 3 ;1 2 0 3 4 ;
dei 1 3 ;2 3 ;2 3 ;
sd 1 sp 12 0 0 5
sfi 2 -3; -2 -3; -2 -3;  sd 1
```

```
sd 2 sp 12 0 0 6
sfi 2 -4; -1 -4; -1 -4;  sd 2
sd 3 cy 0 0 0 1 0 0 5
sfi 1 2; -2 -3; -2 -3;  sd 3
sd 4 cy 0 0 0 1 0 0 6
sfi 1 2; -1 -4; -1 -4;  sd 4
insprt 1  6 2 4
insprt 1  6 3 7
insprt 1  2 1 5
insprt 1  2 2 5
dei 2 3 ;3 4 ;3 4 ;
sd 5 cy 6 0 0 0 1 0 2
sfi -2 -3; 3 4; -3 -4;  sd 5
bb 2 3 3  2 4 4  1;
bb 2 3 3  3 4 3  2;
bb 3 3 3  3 4 4  3;
bb 2 3 4  3 4 4  4;
interrupt
endpart
block 1 4 9 12 ; 1 6 16 ; 1 4 11 14 ; 5 5 7 7 ;5 6 10 ;-1 -1 1 1 ;
dei 1 2 0 3 4 ; ;1 2 0 3 4 ;
dei 2 3 ; ;2 3 ;
bb 1 1 2  1 2 3  1;
bb 4 1 2  4 2 3  3;
bb 2 1 1  3 2 1  2;
bb 2 1 4  3 2 4  4;
sd 6 cy 6 0 0 0 1 0 2
sfi -1 -4; 2 3; -1 -4;  sd 6
sd 7 cy 6 0 0 0 1 0 1
```

sfi -2 -3; ; -2 -3; sd 7

ry 20 rx 10

center

disp

interrupt

endpart

merge

write