B. TECH. PROJECT REPORT

On

Development of python script to alter the face numbering of C3D8 elements in ABAQUS

BY

Suryawanshi Yash Sanjeev (180003056)

DISCIPLINE OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

May 2022

Development of python script to alter the face numbering of C3D8 elements in ABAQUS

A PROJECT REPORT

Submitted in partial fulfillment of the

requirements for the award of the degrees

of

BACHELOR OF TECHNOLOGY

In

MECHANICAL ENGINEERING

Submitted by:

Suryawanshi Yash Sanjeev (180003056)

Guided by:

Dr. Indrasen Singh

INDIAN INSTITUTE OF TECHNOLOGY INDORE

May 2022

CANDIDATE'S DECLARATION

We hereby declare that the project entitled **"Development of python script to alter the face numbering of C3D8 elements in ABAQUS"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in 'Mechanical Engineering' completed under the supervision of **Dr. Indrasen Singh, Assistant Professor, Mechanical Engineering,** IIT Indore is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

25 May 2022

Signature and name of the student(s) with date

CERTIFICATE by BTP Guide(s)

It is certified that the above statement made by the students is correct to the best of my/our knowledge.

25-05-2022

Signature of BTP Guide(s) with dates and their designation

Preface

This report on “Development of python script to alter the face numbering of C3D8 elements in ABAQUS" is prepared under the guidance of Dr. Indrasen Singh. Through this report, we have tried to give a detailed explanation of face numbering and element connectivity of C3D8 elements. The algorithm behind the altercation of face numbering is explained using examples and tried on different geometries. The code for this project is made publicly available.

We have tried to explain the content lucidly to the best of our abilities and knowledge. We have also added figures and tables to make them more illustrative.

Suryawanshi Yash Sanjeev
B.Tech. IV Year
The discipline of Mechanical Engineering
IIT Indore

Acknowledgments

I wish to thank Dr. Indrasen Singh for his kind support and valuable guidance. I am also thankful for the insights provided by Mr. Sumit Chorma.

Through their help and support, I was able to complete the script and technical report.

Without their support, this report would not have been possible.

Suryawanshi Yash Sanjeev
B.Tech. IV Year
The discipline of Mechanical Engineering
IIT Indore

Abstract

The finite element method (FEM) is a numerical technique to solve a partial differential equation. The commercially available software package ABAQUS is very popular in academia as well as industries to analyze engineering problems using FEM.

Advanced constitutive theories for materials require the implementation of new elements in the software package ABAQUS. In Abaqus, the essential boundary conditions such as displacement or temperature boundary conditions are taken care of by the software, whereas the natural boundary conditions such as force and flux boundary conditions need to be implemented in the user element (UEL) subroutine. For this, purpose, the information the surfaces of a particular element must be available in the element subroutine. It must be mentioned that ABAQUS does not provide the information on surfaces at the element level, therefore it becomes mandatory for the user to number the element surfaces compatible with the implementation of the element subroutine. If a uniform mesh is created within ABAQUS itself, the face numbering is uniform and can be altered as per requirement. However, controlling face numbering is impossible if the mesh is imported from some other tool or has transition elements. Therefore, the node numbers must be altered to make them compatible with the implementation of UEL before running the analysis. In a three-dimensional element, the face numbering can be altered by changing the element connectivity.

In this project, a python script is developed to change the face numbering of brick elements before running the simulations in ABAQUS. The algorithm employed in the script is explained with clear illustrations. Three methods have been deployed to select the faces of a part instance for which the face numbering is to be made uniform.

Table of Contents

1. Introduction

1.1 Overview of FEM

The finite element solution to a general continuum problem always follows an orderly step-by-step process:

Step 1:

Divide the structure into discrete elements (discretization). The first step in the finite element method is to divide the structure or solution region into subdivisions or elements. The number, type, size, and arrangement of the elements are to be decided.

Step 2:

Select a proper interpolation.

Step 3:

Derive element stiffness matrices and load vectors. From the assumed displacement model, the stiffness matrix $[K^{(e)}]$ and the load vector $\vec{P}^{(e)}$ of element e are to be derived by using a suitable variational principle, a weighted residual approach (such as the Galerkin method), or equilibrium conditions. The element stiffness matrix and load vector derivation using a weighted residual approach (such as the Galerkin method).

Step 4:

Assemble element equations to obtain the overall equilibrium equations. Since the structure is composed of several finite elements, the individual element stiffness matrices and load vectors are to be assembled in a suitable manner and the overall equilibrium equations have to be formulated as

$$[\underline{K}]\,\underline{\vec{\phi}} = \underline{\vec{P}}$$

Where [$\underline{K}$] is the assembled stiffness matrix, is the vector of nodal displacements $\underline{\vec{\phi}}$ and $\underline{\vec{P}}$ is the vector of nodal forces for the complete structure.

Step 5:

Solve for the unknown nodal displacements. The overall equilibrium equations have to be modified to account for the boundary conditions of the problem. After the incorporation of the boundary conditions, the equilibrium equations can be expressed as

$$[K]\,\vec{\phi} = \vec{P}$$

For linear problems, the vector $\vec{\phi}$ can be solved very easily. However, for nonlinear problems, the solution has to be obtained in a sequence of steps, with each step involving the modification of the stiffness matrix [K] and/or the load vector $\vec{P}$.

Step 6:

Compute element strains and stresses. From the known nodal displacements $\vec{\phi}$; if required, the element strains and stresses can be computed by using the necessary equations of solid or structural mechanics.

Data can be entered in or using an input file prepared with a text editor and executed through the command line, or using a script prepared with Python. Python is an object-oriented programming language and is included in Abaqus as Abaqus Python. The latter is an advanced option reserved for experienced users.

1.2 ABAQUS

This finite element software has strong capabilities to specifically solve nonlinear problems and was developed by Hibbitt, Karlsson, and Sorenson. Three stages are involved while solving a general problem in ABAQUS.

1) ABAQUS Pre-Processor
2) ABAQUS Solver
3) ABAQUS Post-Processor

ABAQUS/CAE provides a complete ABAQUS environment that offers a simple, consistent interface for creating, submitting, monitoring, and evaluating results from ABAQUS/Standard and ABAQUS/Explicit simulations. ABAQUS/ CAE is divided into modules in which each one defines a logical aspect of the modeling process: for example, to define the geometry, define the material properties, and generate a mesh. As we move from one module to another, we build the model from which ABAQUS/CAE generates an input file that we can submit to ABAQUS/Standard or ABAQUS/Explicit to carry out the analysis.

The examples presented are based on writing script files. An input (script) file can be written in any text editor tool such as WordPad. The input file contains information including the kind of problem, the geometry of the structure, the properties of the structure, the analysis parameters, and the output requirements.

1.3 Need for change in face numbering.

Certain steps in formulating a finite element analysis of a physical problem are common to all for example analysis of structural beams, heat transfer, fluid flow, or some other problem. These steps are employed infinite element software packages, although we do not use these steps anywhere in this project it's necessary to have a basic understanding of its working refer. We have described the steps as follows

Preprocessing:

The preprocessing step consists of describing defining the model and includes:

a) Define the geometric domain of the problem.

b) Define the element type(s) to be used.

c) Define the material properties of the elements.

d) Define the geometric properties of the elements (length, area, and the like).

e) Define the element connectivity (mesh the model).

f) Define the physical constraints (boundary conditions).

g) Define the loadings.

Solution

During the solution step, FEM software assembles the governing algebraic equations in matrix form and computes the unknown values of the primary field variable(s). The values calculated are then used by back substitution to compute additional, derived variables, such as reaction forces, element stresses, and heat flow.

Postprocessing

Analysis and evaluation of the solution results are called postprocessing. Postprocessor software contains sophisticated techniques used for printing, sorting, and plotting selected results from a finite element solution obtained.

Need

For analyzing a problem using an in-build element. However, the development of new modelling techniques and constitutive theories such as phase-field-based models employed in the fracture mechanics and simulations of the deformation response of piezoceramics demand the writing of user element subroutine (UEL).

When specifying certain BC, we need to have the information about the face numbering of the element set because these BCs apply on surfaces of the mesh. If we already have the same face numbering for every element, we can avoid the hectic process of gathering it through visualization tools provided by ABAQUS for thousands of elements and changing it every time in the UEL for different BCs.

1.4 Input File

Abaqus input file is an ASCII data file. It can be created by using any commercially available text editor or by using a graphical preprocessor e.g., Abaqus/CAE. The input file consists of a series of lines containing the Abaqus keyword and data lines. The geometry of the model can be majorly described by elements and their nodes. The rules and methods followed for defining nodes and elements in ABAQUS are described in the upcoming chapter. For now, we are just concerned with information about nodal coordinates and element connectivity.

In the figure an example of an input file is given for geometry with 1 C3D8 element.

Fig.1.1

```
*Heading
** Job name: case3 Model name: Model-1
** Generated by: Abaqus/CAE 2017
*Preprint, echo=NO, model=NO, history=NO, contact=NO
**
** PARTS
**
*Part, name=Part-1
*Node
      1,2.0,0.0,2.0
      2,7.0,0.0,3.0
      3,8.0,0.0,7.0
      4,2.0,0.0,6.0
      5,7.0,5.0,2.0
      6,8.0,5.0,5.0
      7,4.0,5.0,7.0
      8,3.0,5.0,5.0
*Element, type=C3D8R
1, 1,2,3,4,5,6,7,8
*End Part
**
**
** ASSEMBLY
**
*Assembly, name=Assembly
**
*Instance, name=Part-1-1, part=Part-1
*End Instance
**
*End Assembly
```

```
**
** MATERIALS
**
*Material, name=Material-1
** ----------------------------------------------------------------
**
** STEP: Step-1
**
*Step, name=Step-1, nlgeom=NO
*Static
1., 1., 1e-05, 1.
**
** OUTPUT REQUESTS
**
*Restart, write, frequency=0
**
** FIELD OUTPUT: F-Output-1
**
*Output, field, variable=PRESELECT
**
** HISTORY OUTPUT: H-Output-1
**
*Output, history, variable=PRESELECT
*End Step
```

The first step in changing the face numbering in ABAQUS is to understand how face numbering is done in ABAQUS and what is its relation to element connectivity.

2. The methodology adapted by ABAQUS for face numbering

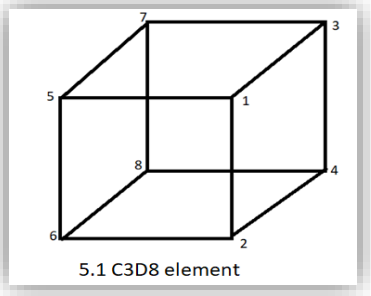
2.1 How ABAQUS/CAE pre-processor does face numbering?

C3D8 type of elements is considered for developing the algorithm.

In the figure below we can see every node has a number and they have some coordinates concerning the global coordinate system. We should know that face numbering has no connection with the coordinates of the nodes. But the coordinates play an important role in changing face numbering.

Element Connectivity is one of the important components of the input file in ABAQUS. Generally, in a pre-processor such as ABAQUS/CAE when we define a model geometry the pre-processor automatically creates the nodes and elements needed for analysis but if we want to create an input file manually, we have to define the nodes as well as element connectivity

Fig 2.1



5.1 C3D8 element

2.2 Element Definition:

Following are the steps that should be followed to define a C3D8 element:

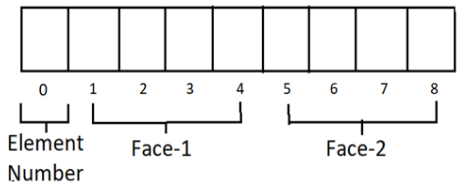
a) Assign a numerical label to the element called the element number. The element should be a positive number and can be given a maximum value of 999999999. In ABAQUS every model is defined as an assembly of part instances created and when defining the elements, every element must belong to the part instance. Element number is unique within a part but they can be repeated in different part instances.

e.g., Part - 1: Elements - (1,2,3…….1000)

Part - 2: Elements - (1,2, 3…….100)

b) Now we have to mention the node number of the nodes that define that particular element. In our case, as we are considering a C3D8 node, an element is always defined by 8 nodes. We can't mention the node numbers in any order but a particular pattern is to be followed. The pattern depends on stack direction and the possible arrangement of nodes that can give that particular stack direction.

So, the element definition for an element is illustrated in figure 5.2:



5.2 Element Definition

Fig 2.2

It can be considered as an array taken for visual understanding.

E1, E2, E3, E4, E5, E6, E7, and E8 are node numbers and as a C3D8 element has six faces and all the faces have some assigned number that belongs in the set {1,2,3,4,5,6}.

Each face consists of 4 nodes and it's important to know that Face-1 is defined by nodes [E1, E2, E3, E4] and Face-2 is defined by nodes [E5, E6, E7, E8]. It's important to understand how we name Face-1 and Face-2 because it defines the way Face-3, Face-4, Face-5, and Face-6 are assigned to the element.

c) The face given on the right side can be named in the following number of ways, and the face-name itself decides how the element definition is written

Fig 2.3

i. Clockwise Direction: 1,3,4,2; 3,4,2,1; 4,2,1,3; 2,1,3,4
ii. Anticlockwise Direction: 1,2,4,3; 2,4,3,1; 4,3,1,2; 3,1,2,4

Note: ‘,’ is used between the nodes to differentiate in case the node numbers are not single-digit.

d) So, in array E as shown in fig5.2, Face-1’s name should be written as “E1, E2, E3, E4” and Face-2’s name should be written as “E5, E6, E7, E8”.

*e) The assignment of assigning face numbers is illustrated below. Before going to the examples an important note to remember is that while starting to write face names of Face-1 and Face-2 ensure that node numbers placed at E1, and E5 should lie on the edges of the element similarly for E2, and E6; E3, and E7; E4, and E8. Just ensuring this makes all the other respective nodes aligned on the same edge, note that this is just for Face-1 and Face-2 because we need to specify only these faces for the element definition.

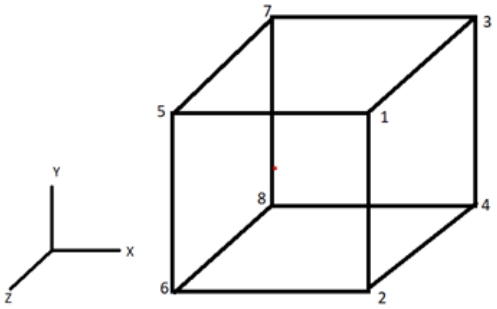


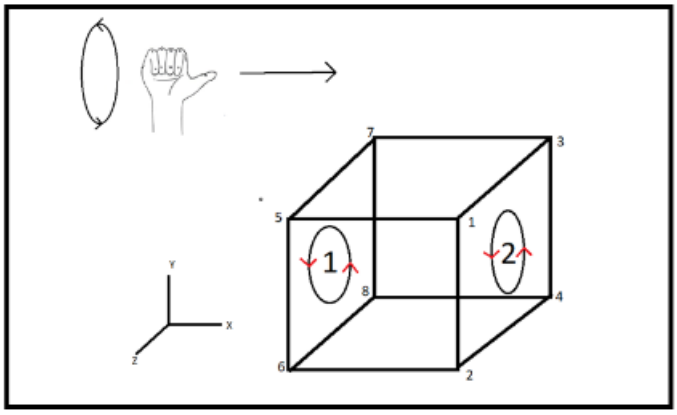
Fig 2.4 **C3D8 Element and Coordinate Axes**

Before proceeding to the examples where we will understand how Abaqus assigns face numbering let’s understand stack direction.

2.3 Stack Direction:

A. Figure 5.4 explains the situation when stack direction is along the positive X-axis.

Let’s understand it if the stack direction is positive X-axis, it means we take faces perpendicular to X-axis (both +ve X-axis and -ve X-axis) and name them 1 and 2. Now assignment of Face-3, Face-4, Face-5, Face-6 depends on which axis has been assigned to Face-1 and Face-2.



5.4 Stack Direction - X axis

Fig.2.5

So, if the stack direction is +ve X-axis it means that the face which has normal along -ve X-axis has been assigned as Face-1, and the face which has normal along +ve X-axis has been assigned as Face-2.

Let's elaborate on this process by taking a few more examples.

1. Stack Direction (+ve X axis): Face-1 = -ve X axis | Face-2= +ve X axis

2. Stack Direction (-ve X axis): Face-1 = +ve X axis | Face-2= -ve X axis

3. Stack Direction (+ve Y axis): Face-1 = -ve Y axis | Face-2= +ve Y axis

4. Stack Direction (-ve Y axis): Face-1 = +ve Y axis | Face-2= -ve Y axis

5. Stack Direction (+ve Z axis): Face-1 = -ve Z axis | Face-2= +ve Z axis

6. Stack Direction (-ve Z axis): Face-1 = +ve Z axis | Face-2= -ve Z axis

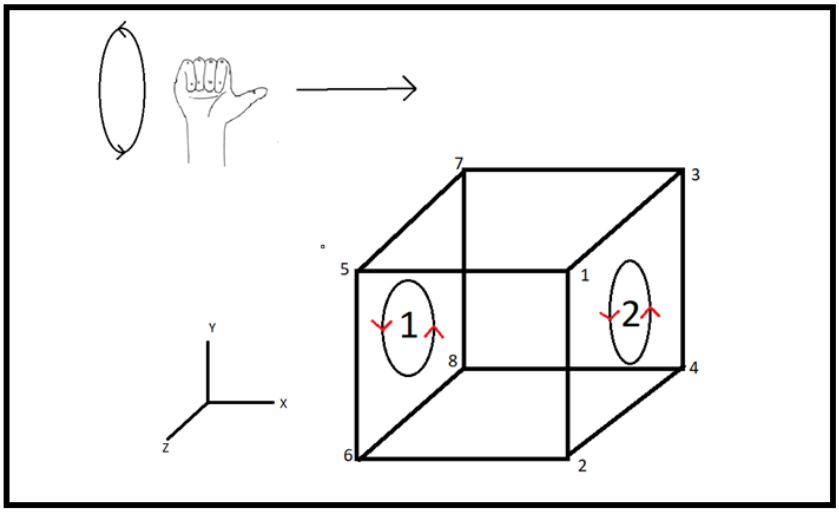
B. The process of defining Face-3,4,5,6 is done using the right-hand thumb rule. The right-hand thumb rule suggests that the stack direction will be positive X if and only if the names of Face-1 and Face-2 are given in a counter-clockwise direction (an *important thing to note while naming is Property *e) in Element Definition must be followed*) which in turn leads to Faces – (3,4,5,6) being

taken in the direction of the curled fingers respectively. This will be clearer when the examples are explained.

2.4 Face Numbering Cases

The pre-requisite required for understanding face numbering have been explained above let's look at the possible element definitions for the element provided in figure 5.4. I have re-attached the figure below.

1. (Stack Direction: Positive X-axis)



5.4 Stack Direction - X axis

Fig.2.6

So, the possible combination of element definition is:

a) 1, 5,6,8,7, 1,2,4,3

b) 1, 6,8,7,5, 2,4,3,1

c) 1, 8,7,5,6, 4,3,1,2

d) 1, 7,5,6,8, 3,1,2,4

All the four-element definitions have the same faces as Face-1 and Face-2, the only thing that changes are Face-3, Face-4, Face-5, and Face-6. Below illustration of how changing the face-name of Face-1 and Face-2 will change the assignment of other faces

a) Element Definition: 1, 5,6,8,7, 1,2,4,3

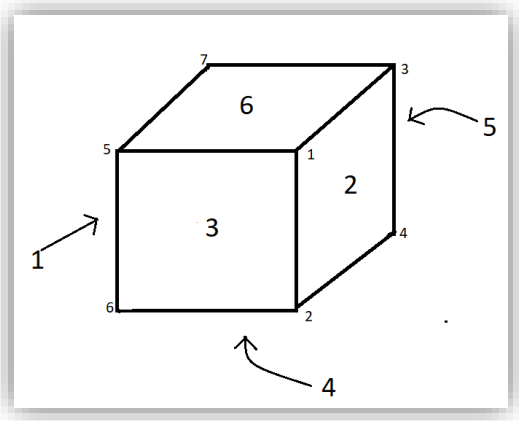


Fig.2.7

Face-1 Name: 5,6,8,7

5,6: Face-3 | 6,8: Face-4 | 8,7: Face-5 | 7,5: Face-6

Explanation:

Now we can observe from the Face-1 name that the first edge in the name is 5,6 (Note: I am using commas for avoiding confusion if the nodes are not single digits). So, the face with edges 5,6 is Face-3. Similarly, we have a face with edge 6,8 is Face-4, a face with edge 8,7 is Face-5 and a face with edge 7,5 is Face-6.

Face-2 Name: 1,2,4,3

1,2: Face-3 | 2,4: Face-4 | 4,3: Face-5 | 3,1: Face-6

Explanation: Due to property e) in element definition

An interesting point to notice here is that to change the direction of the normal of Face-3,4,5,6 we just need to change the face-name of Face-1 or Face-2 clockwise or anticlockwise decided by the

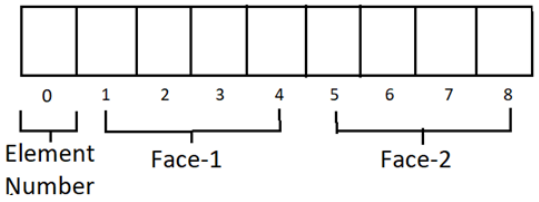


Fig.2.8 **Element Definition**

stack direction.

In the above example, the stack direction is +ve X-axis so the face name should be changed counter-clockwise as seen from the top of the right-hand thumb. If you change the face-name of Face-1 exactly replicate the changes in Face-2. It is very important to remember this point as it will be helpful when we will enter the section for a change of face numbering. All cases will become clearer as you see more and more examples.

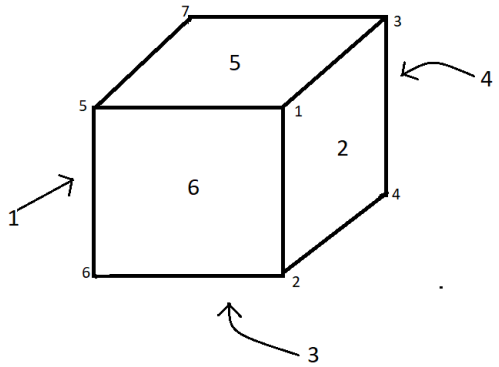
b) Element Definition: 1, 6,8,7,5, 2,4,3,1

Face-1 Name: 6,8,7,5

6,8: Face-3 | 8,7: Face-4 | 7,5: Face-5 | 5,6: Face-6

Face-2 Name: 2,4,3,1

2,4: Face-3 | 4,3: Face-4 | 3,1: Face-5 | 1,2: Face-6



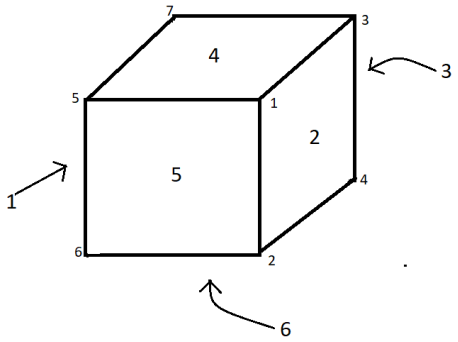
c) Element Definition: 1, 8,7,5,6, 4,3,1,2

Face-1 Name: 8,7,5,6

8,7: Face-3 | 7,5: Face-4 | 5,6: Face-5 | 6,8: Face-6

Face-2 Name: 4,3,1,2

4,3: Face-3 | 3,1: Face-4 | 1,2: Face-5 | 2,4: Face-6



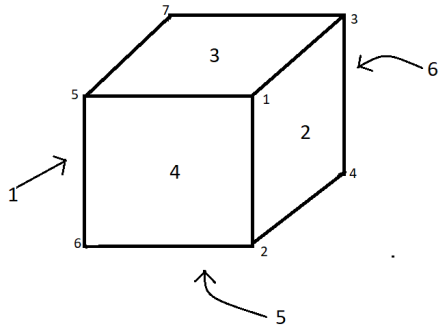
d) Element Definition: 1, 7,5,6,8, 3,1,2,4

Face-1 Name: 7,5,6,8

7,5: Face-3 | 5,6: Face-4 | 6,8: Face-5 | 8,7: Face-6

Face-2 Name: 3,1,2,4

1,2: Face-3 | 2,4: Face-4 | 4,3: Face-5 | 3,1: Face-6



2. (Stack Direction: Negative X-axis)

So, the possible combination of element definitions is:

a) 1, 2,1,3,4, 6,5,7,8 b) 1, 1,3,4,2, 5,7,8,6

c) 1, 3,4,2,1, 7,8,6,5 d) 1, 4,2,1,3, 8,6,5,7

a) Element Definition: 1, 2,1,3,4, 6,5,7,8

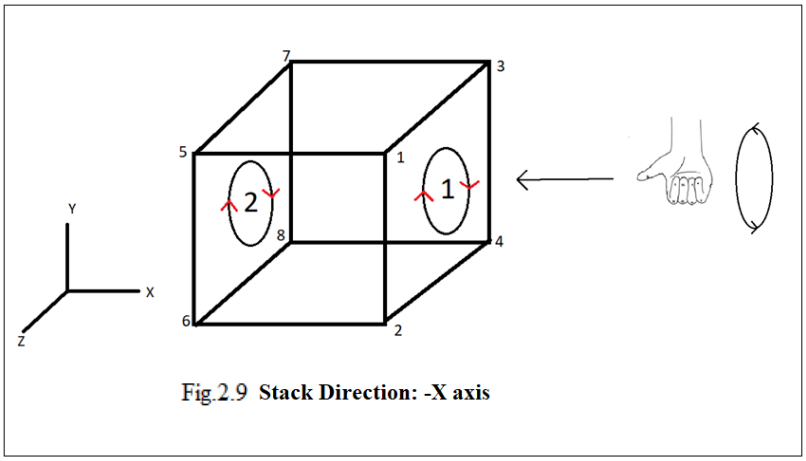


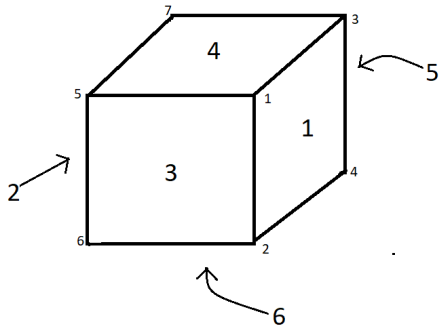
Fig.2.9 **Stack Direction: -X axis**

Face-1 Name: 2,1,3,4

2,1: Face-3 | 1,3: Face-4 | 3,4: Face-5 | 4,2: Face-6

Face-2 Name: 6,5,7,8

6,5: Face-3 | 5,7: Face-4 | 7,8: Face-5 | 8,6: Face-6



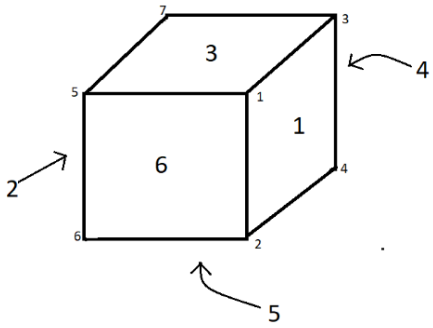
b) Element Definition: 1, 1,3,4,2, 5,7,8,6

Face-1 Name: 1,3,4,2

1,3: Face-3 | 3,4: Face-4 | 4,2: Face-5 | 2,1: Face-6

Face-2 Name: 5,7,8,6

5,7: Face-3 | 7,8: Face-4 | 8,6: Face-5 | 6,5: Face-6



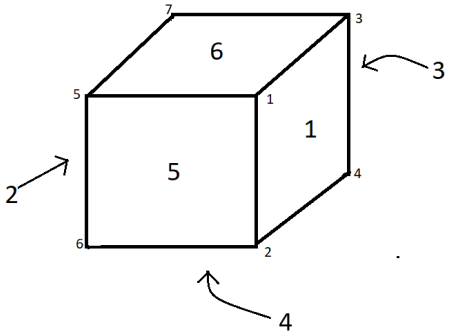
c) Element Definition: 1, 3,4,2,1, 7,8,6,5

Face-1 Name: 3,4,2,1

3,4: Face-3 | 4,2: Face-4 | 2,1: Face-5 | 1,3: Face-6

Face-2 Name: 7,8,6,5

7,8: Face-3 | 8,6: Face-4 | 6,5: Face-5 | 5,7: Face-6



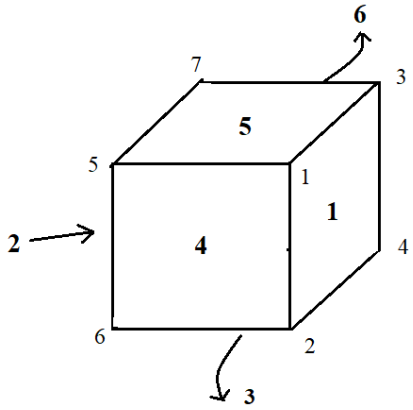
d) Element Definition: 1, 4,2,1,3, 8,6,5,7

Face-1 Name: 4,2,1,3

4,2: Face-3 |: Face-4 | 2,1: Face-5 | 1,3: Face-6

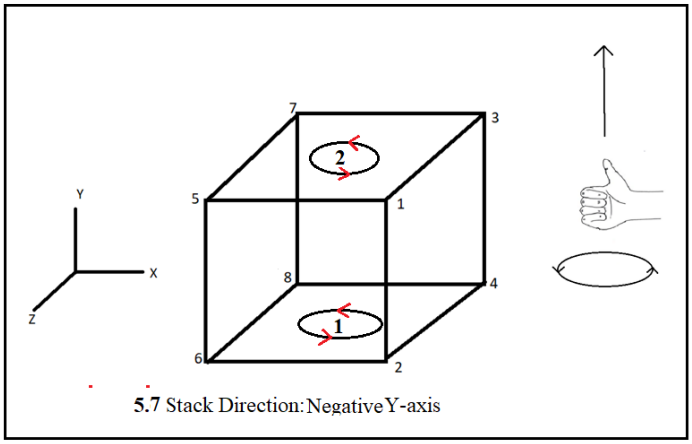
Face-2 Name: 8,6,5,7

8,6: Face-3 | 8,6: Face-4 | 6,5: Face-5 | 5,7: Face-6



3. (Stack Direction: Positive Y-axis)

Fig.2.10



5.7 Stack Direction: Negative Y-axis

So, the possible combination of element definitions is:

a) 1, 2,4,8,6, 1,3,5,7 b) 1, 4,8,6,2 3,7,5,1

c) 1, 8,6,2,4, 7,5,1,3 d) 1, 6,2,4,8, 5,1,7,3

4. (Stack Direction: Negative Y-axis)

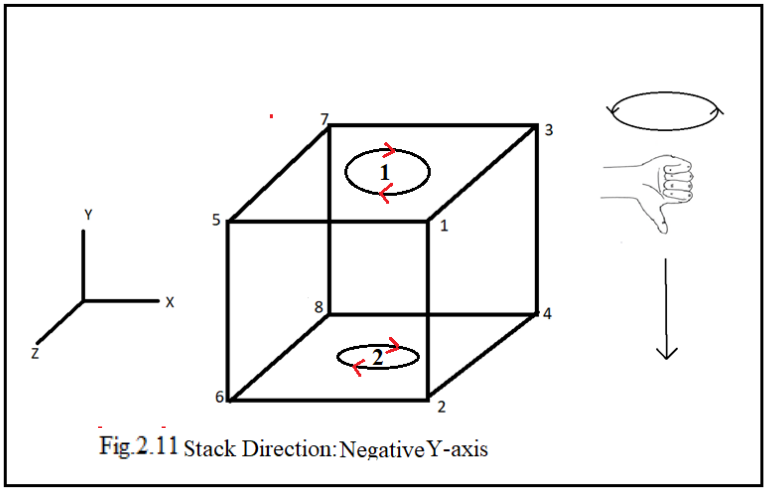


Fig.2.11 Stack Direction: Negative Y-axis

So, the possible combination of element definitions is:

a) 1, 1,5,7,3, 2,6,8,4 b) 1, 5,7,3,1, 6,8,4,2

c) 1, 7,3,1,5, 8,4,2,6 d) 1, 3,1,5,7, 4,2,6,8

5. (Stack Direction: Positive Z-axis)

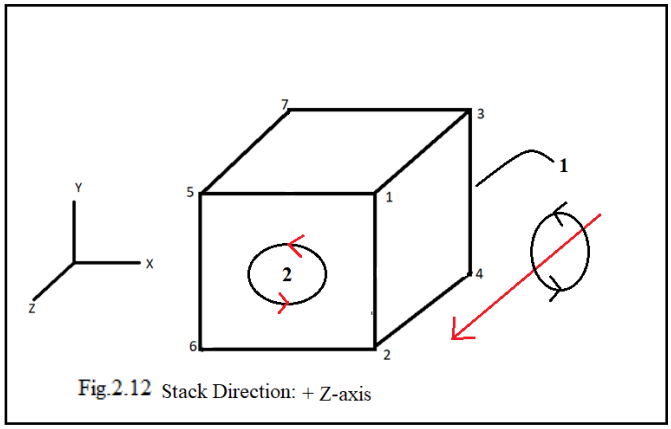


Fig.2.12 Stack Direction: + Z-axis

So, the possible combination of element definitions is:

a) 1, 3,7,8,4, 1,5,2,6 b) 1, 7,8,4,3, 5,2,6,1

c) 1, 8,4,3,7, 2,6,1,5 d) 1, 4,3,7,8, 6,1,5,2

6. (Stack Direction: Negative Z-axis)

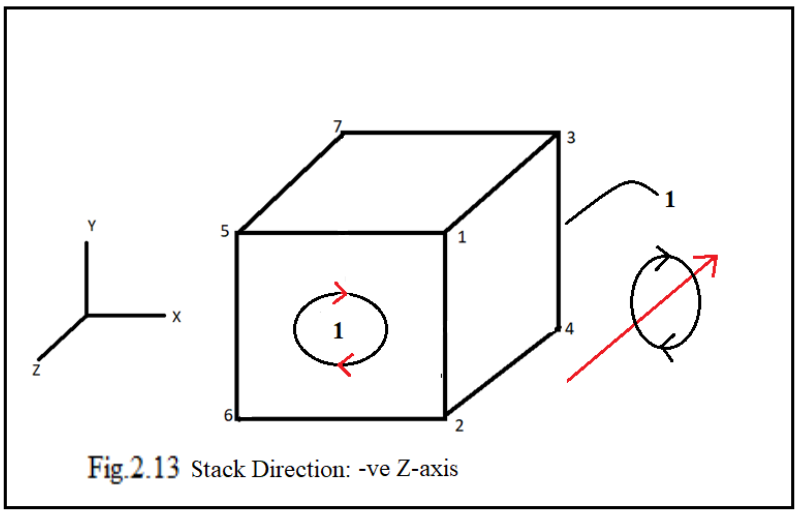


Fig.2.13 Stack Direction: -ve Z-axis

So, the possible combination of element definitions is:

a) 1, 1,2,6,5, 3,4,8,7 b) 1, 3,4,8,7, 1,2,6,5

c) 1, 4,8,7,3, 2,6,5,1 d) 1, 8,7,3,4, 6,5,1,2

In the next chapter, we will understand the algorithm employed for change in face numbering by us.

2. The strategy employed to change face numbering.

Now that we have learned how to define an element in ABAQUS and how face numbering is done, it's time to know how to change the face numbering of a particular face of an element.

There are 30 possible cases for changing the face numbering, I have mentioned all of them below

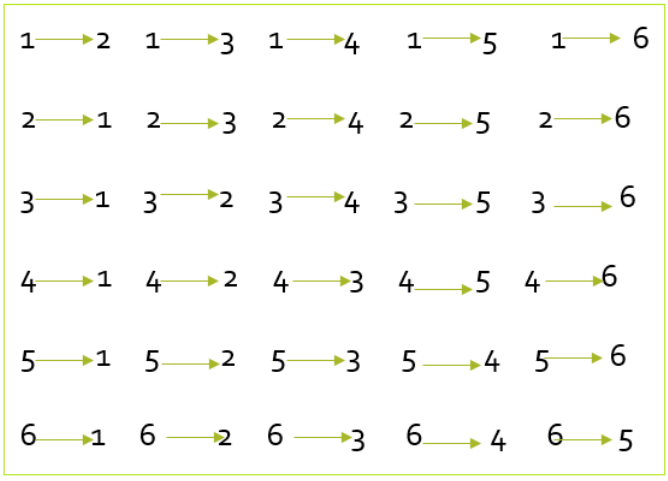


Fig.3.1

We don't have to figure out the logic for every case, because when you change one face to a different number, five other faces of the element have already been changed to a different number.

3.1 Pre-requisite

The way ABAQUS has defined its face numbering if we fix 2 faces, all the other 3 faces are going to be fixed. Suppose we fix Face-1 to be in a certain direction of the axis and similarly we fix Face-3. Fixing Face-1 will fix the stack direction and the way Faces-3,4,5,6 should be assigned whether it is in a clockwise or anticlockwise manner, after fixing Face-1 we will have 4 DOF to choose Face-3 and as soon as we fix Face-3 we will fix how Face-1 and Face-2 will be named. Let's illustrate this using an example

Example – 3. a):

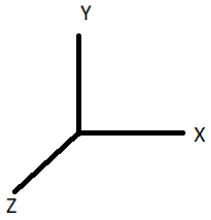
As shown in figure 6.1, we have an element with no face numbering assigned to it.

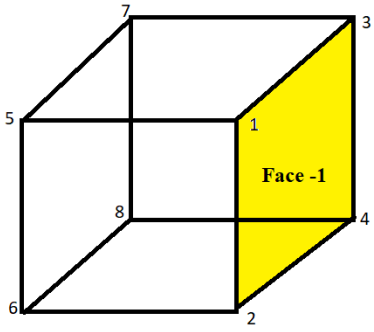
Let's assign Face-1 in the direction of the positive X-axis. This will imply that our stack direction is negative X-axis and the dace should be named in a counter-clockwise direction when viewed from the negative X-axis.

Face-1 is highlighted.

As Face-2 is always opposite to Face-1 it has also been assigned. Now let's define an element with the information we currently have.

Fig.3.2





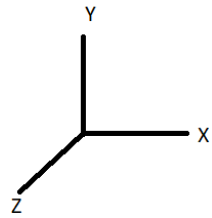
Face-1 Fixed

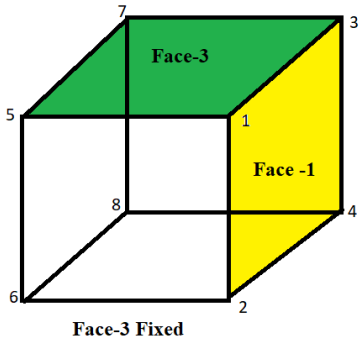
The possible combinations for this particular stack direction are:

a) 1, 1,3,4,2 ,5,7,8,6
b) 1, 3,4,2,1 ,7,8,6,5
c) 1, 4,3,1,3 ,8,6,5,7
d) 1, 2,1,3,4 ,6,5,7,8

Now, as soon as we fix assign Face-3 the element defines the user wants is obtained. Suppose we say that we want Face-3 to be assigned to the face perpendicular to the positive X-axis

Fig.3.3





Face-3 Fixed

Face-3 is assigned.

Assigning Face-3 assigns Face-4, Face-5, and Face-6 in a counter-clockwise direction. For the final connection recall the point where I mentioned that Face-3 is formed of E [1], E [2] from Face-1 and E [3], and E [4] from Face-2 (i.e., Face-3 is formed by nodes 1,3 and 5,7).

The face which has a normal parallel to the positive Y-axis consists of nodes {1,3,5,7} just check from the combinations a, b, c, and d which connection has the node-set mentioned in E [1], E [2], E [3], E [4] {i.e. nodes forming Face-3} and we will find that the combination is a).

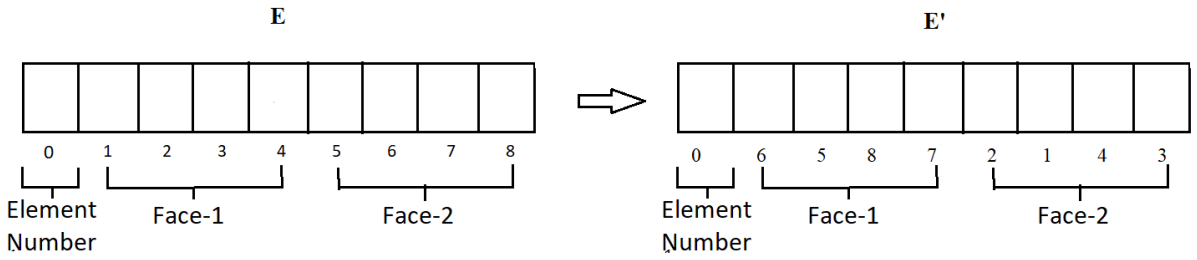
The final element connectivity after specifying directions of Face-1 and Face-3 is a).

Now we know that for changing face numbering the information that we need from the user in the direction of the new Face-1 and the direction of the new Face-3. We should also note that we get one more input from the user about the default connectivity of the elements provided by ABAQUS.

After looking at many combinations of face numbering, I have obtained the following:

Let's consider the element connection as an array E as shown below:

Note: To change the assignment of face-1 and face-2 we have to rearrange the entire array

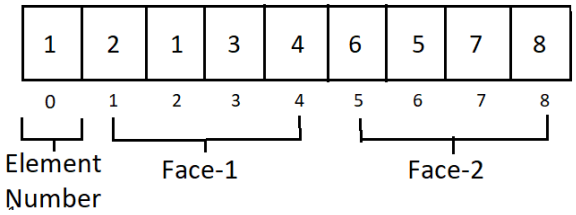


6.1 This shows how to change Face-2 to Face-1 , for any valid connection that exists in ABAQUS without focusing on the assignment of Faces-3,4,5,6 which will be different for different connections

Let's illustrate the above change by an example.

Example-1

Initial Element Definition - E:



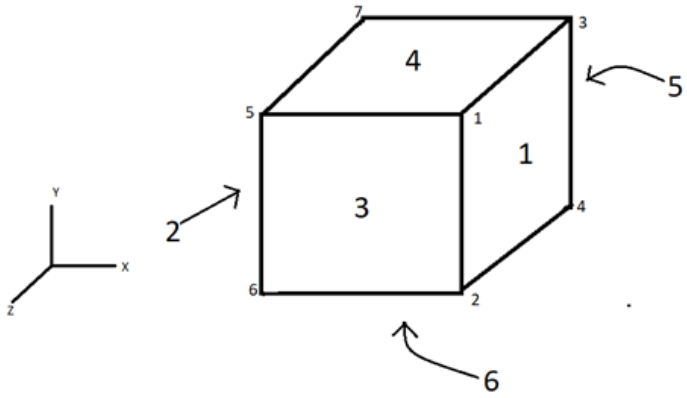
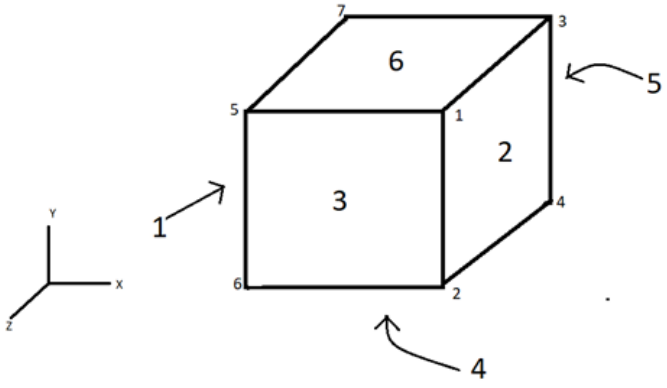


Fig. 3.4

Now we do a specific re-arrangement, suppose the user tells us that the direction of normal of Face-1 should be in the direction of the negative X-axis which means that the user wants Face-2 to be converted to Face-1. Let's see how this works.

Final Element Definition – E':

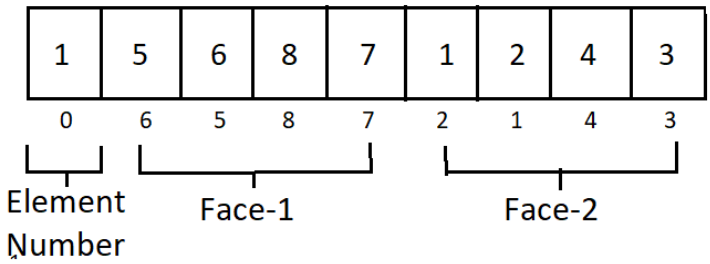
Fig. 3.5



This re-arrangement always leads to:

a) Face-2 to Face-1 b) Face-1 to Face-2

But it differs connection to connection on how Face-3,4,5,6 change among themselves. In this example, Face-3 remains the same Face-4 is converted to Face-6 and all other faces remain the same.



Now as we discussed some concepts in example 6. a) we know that as soon as the user fixes Face-1, Face-2 gets fixed and now that remains is fixing Face-3 which will lead to fixing Face-4, Face-5, Face-6.

Now the part where the right-hand thumb rule comes into fruition has already been taken care of by the rearrangement the only thing to do is rotate the sub-arrays consisting of Face-1 and Face-2 simultaneously and make the same changes happen. Let's see how the rotation of sub-arrays gives us the desired face-numbering.

Figure 6.2 has been implemented in the code by function “restricted_perm”. Now let’s apply this rotation to Example-1 and as E’ will be stored in a new array it will have new indices as shown in 6.2, not the old ones.

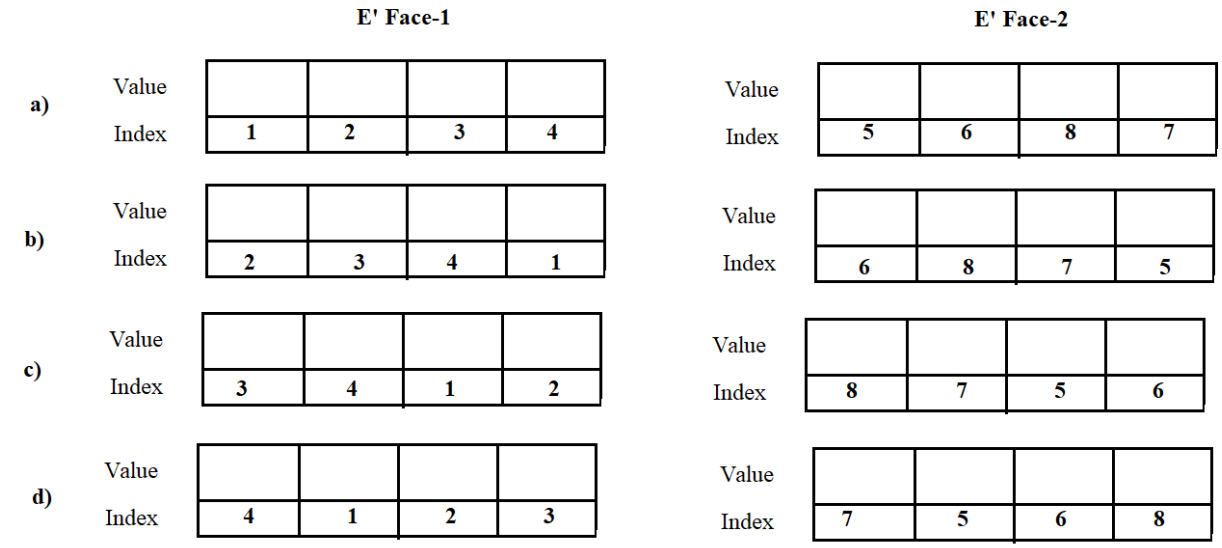
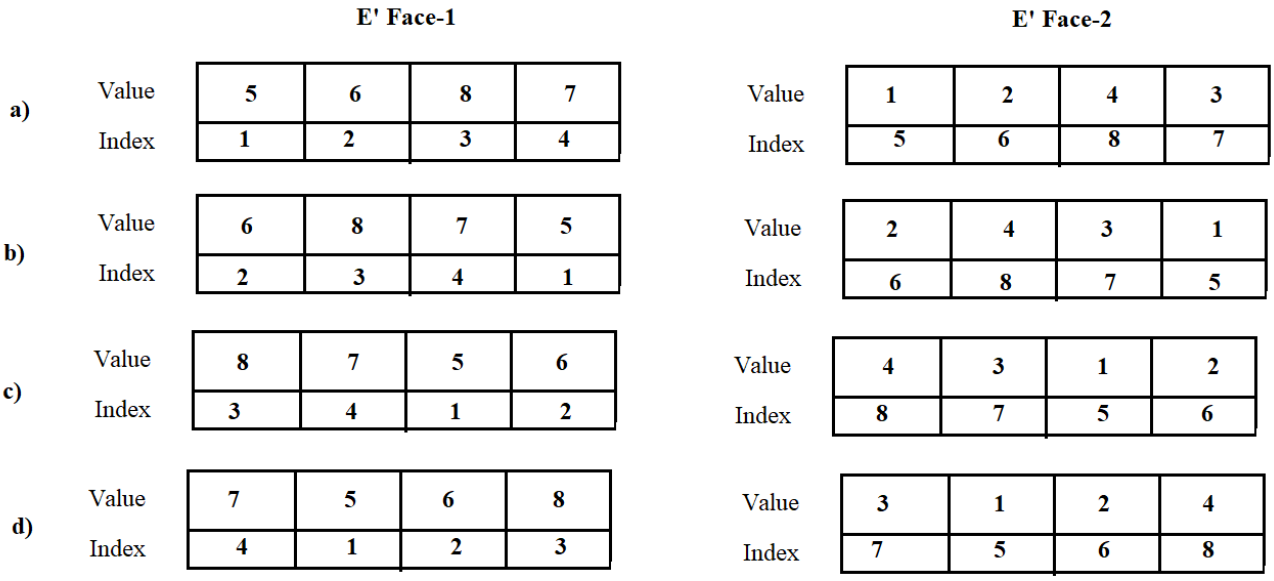
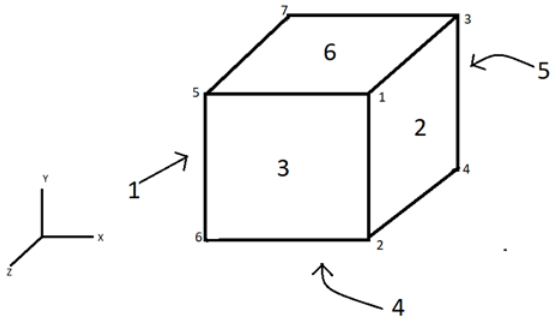


Fig. 3.6 Permutations of the array which lead to changing of Faces-[3,4,5,6]

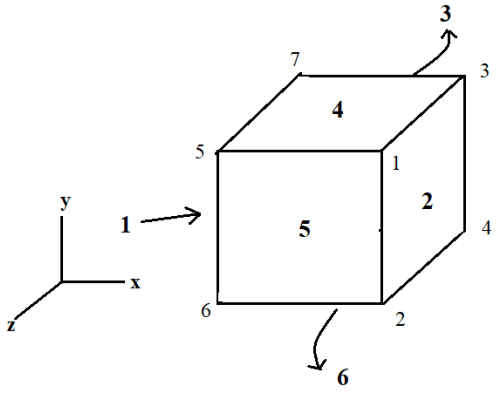


Applied figure 3.7 to Example-1

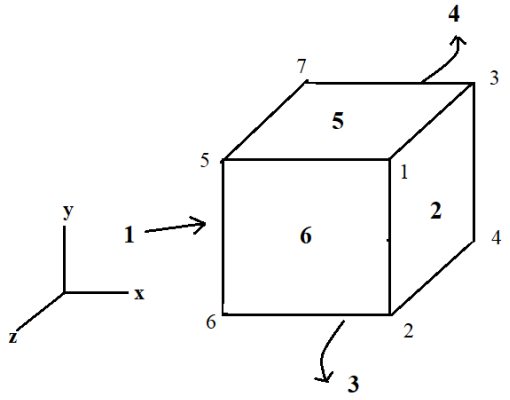
a)



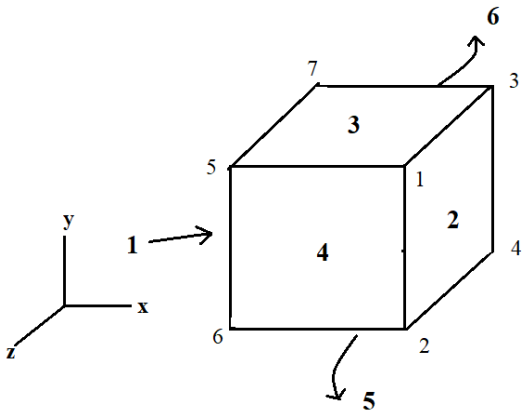
b)



c)

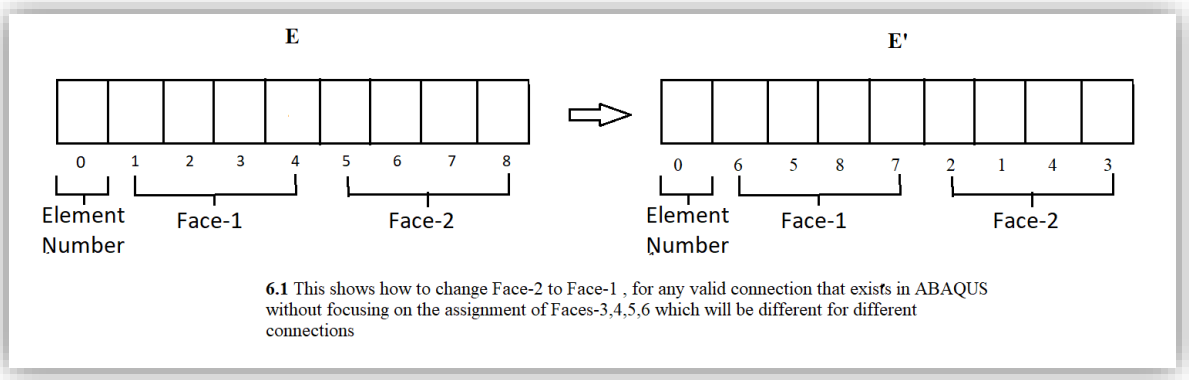


d)

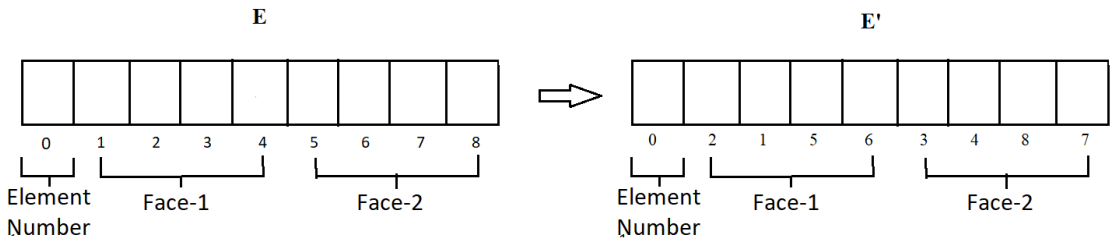


After the user fixes that they want Face-1's normal to be pointing towards the negative Z-axis, we use the re-arrangement and we narrow down our space to 4 from 24 combinations. Now the user fixes the direction of Face-3's normal and as soon as he fixes that, we calculate the normal of Face-3 from all 4 connections and see which connection has a Face-3 normal parallel to the user's desired axis. The topic on the calculation of normal and finding out nodes that consist of Face-3 will be covered in the next section but I think if we have understood everything until this point, you can predict how to establish that very easily.

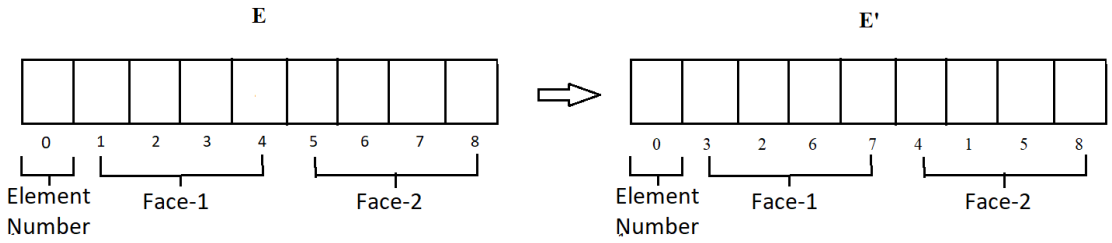
3.2) 6 Tables: Re-arrangement of connections



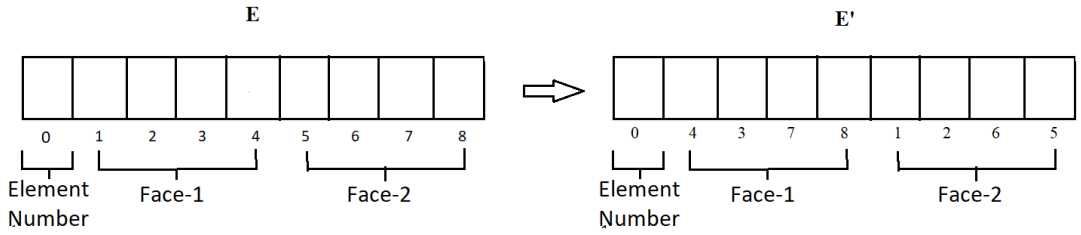
6.1 This shows how to change Face-2 to Face-1 , for any valid connection that exists in ABAQUS without focusing on the assignment of Faces-3,4,5,6 which will be different for different connections



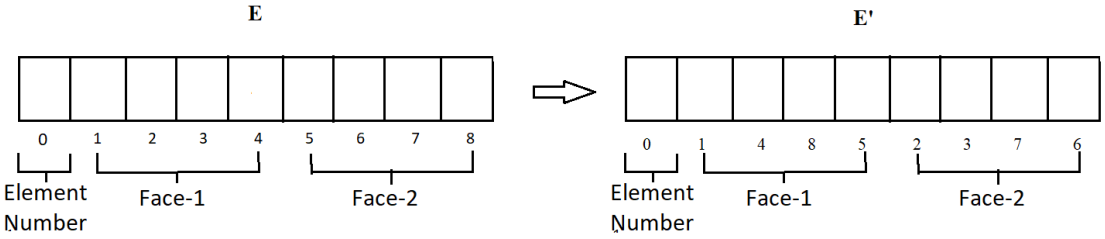
6.2 This shows how to change Face-3 to Face-1 , for any valid connection that exists in ABAQUS without focusing on the assignment of Faces-3,4,5,6 which will be different for different connections



6.3 This shows how to change Face-4 to Face-1 , for any valid connection that exists in ABAQUS without focusing on the assignment of Faces-3,4,5,6 which will be different for different connections



6.4 This shows how to change Face-5 to Face-1 , for any valid connection that exists in ABAQUS without focusing on the assignment of Faces-3,4,5,6 which will be different for different connections



6.5 This shows how to change Face-6 to Face-1 , for any valid connection that exists in ABAQUS without focusing on the assignment of Faces-3,4,5,6 which will be different for different connections

We have now covered 24 combinations that exist for an element connection. The only part remaining is calculating normal and finding its direction which gets difficult if the element is rotated or distorted but for now, the part of changing the element connectivity is over.

Note: After the user input's the axis along which he/she desires to have the normal of Face-1 after changing connections. We find which face's normal in the current configuration is pointing towards the desired axis. Suppose we notice that Face-6 is supposed to be the next Face-1 we go to the re-arrangement chart and use that re-arrangement. Then we get narrowed down to 4 combinations, we ask the user for the direction of normal of the new Face-3 and then we check among the 4 combinations which one has Face-3 normal aligned with the user's input, and voila we have changed the face numbering.

3.3 Normal Vector Calculation

The way ABAQUS has defined face numbering for any combination if we take certain node number values from an array E which represents element connection, we can extract both the nodes from which the face is formed and a way to calculate normally.

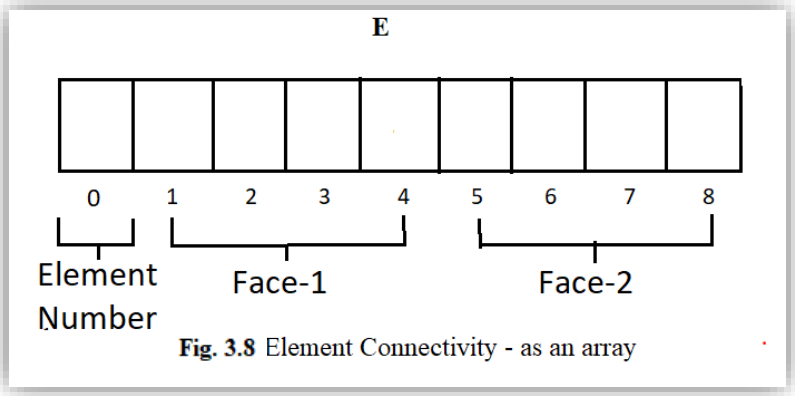


Fig. 3.8 Element Connectivity - as an array

So, no matter which connection is given to the program by ABAQUS.

- Face-1: [E[1], E[2], E[3], E[4]]
- Face-2: [E[5], E[6], E[7], E[8]]
- Face-3: {E[1], E[2], E[5], E[6]}
- Face-4: {E[2], E[3], E[6], E[7]}
- Face-5: {E[3], E[4], E[7], E[8]}
- Face-6: {E[1], E[4], E[5], E[8]}

Fig. 3.9 Node Extraction for Face's-1,2,3,4,5,6

The reason why Face-1 and Face-2 nodes have been written in the form of a list is that the order of the nodes matter, that's what decides the face numbering, whereas for Face-3,4,5,6 we need not follow an order but we have to find a method to extract the nodes in a unique way to calculate face-normal let's demonstrate why?

Note: Normal vector of faces-1,2,3,4,5,6 is directed outwards for the cubic element as shown in the below figure

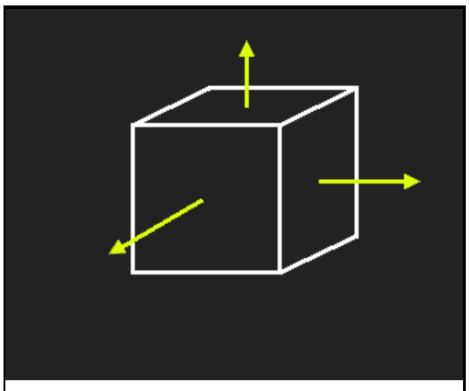


Fig. 3.10 Normal directing outwards

Example: 7.1.a)

Suppose we want to calculate the normal of Face-4, from figure 7.2 we have to extract the node-set the forms Face-4: {7,5,3,1}. We can extract the coordinates of these nodes from the input file under the section of node definition. Now to calculate the normal of Face-4 we need to take the

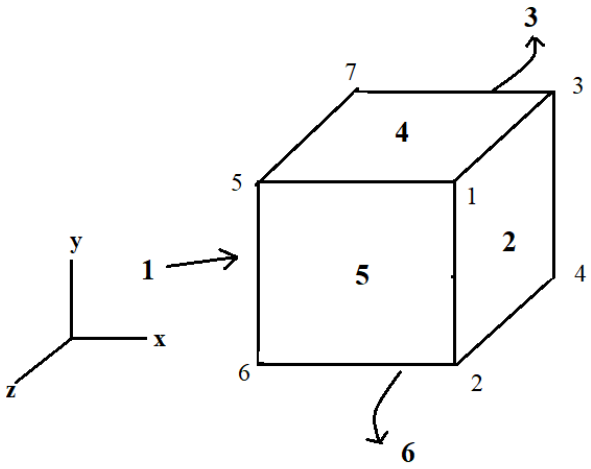


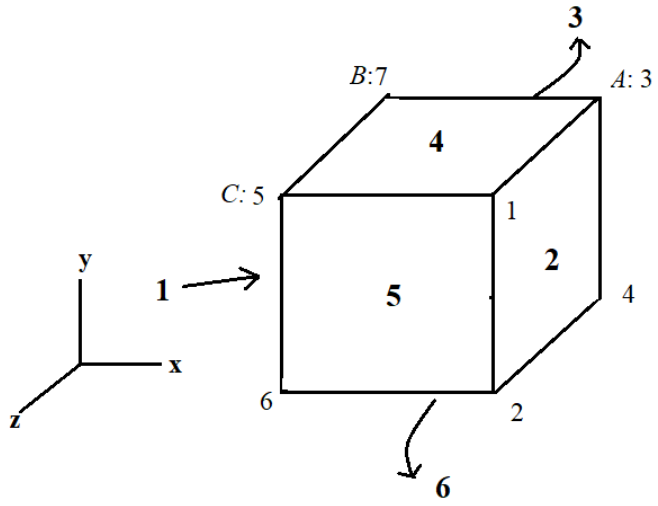
Fig. 3.11 Element Connectivity : 1, 8,7,5,6, 4,3,1,2

cross product of any two sides.

Let's assume:

Assumption:

Fig. 3.12



Vector $\vec{B}$: *A*=Coordinates (Node '3'), *B*=Coordinates (Node '7')

Vector $\overrightarrow{CB}$: *C*=Coordinates (Node '5'), *B*=Coordinates (Node '7')

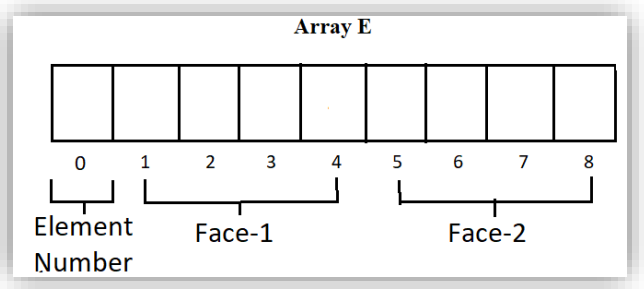
Cross Product $\overrightarrow{AB}$ x $\overrightarrow{CB}$ points towards negative Y-axis which is wrong cause according to out convention normal should be pointing towards positive Y-axis.

Cross Product $\overrightarrow{CB}$ x $\overrightarrow{AB}$ points towards positive Y-axis which is correct

How do we make sure we calculate the correct normal?

After looking at 24 various connections I have found a pattern that will correctly calculate the normal no matter the connection.

Choice of Nodes and Vectors:



i. **Normal Face-1:** *A* = Coordinates (Node E[1])

B = Coordinates (Node E[2])

C = Coordinates (Node E[3])

$\overrightarrow{N1}$ (Normal Vector): $\overrightarrow{AB}$ x $\overrightarrow{CB}$

ii. **Normal Face-3:** *A* = Coordinates (Node E[2])

B = Coordinates (Node E[1])

C = Coordinates (Node E[5])

$\overrightarrow{N3}$ (Normal Vector): $\overrightarrow{AB}$ x $\overrightarrow{CB}$

iii. **Normal Face-4:** A = Coordinates (Node E[3])

B = Coordinates (Node E[2])

C = Coordinates (Node E[6])

$\overrightarrow{N4}$ (Normal Vector): $\overrightarrow{AB}$ x $\overrightarrow{CB}$

We need not calculate the normal for Face's-2,5,6 because Face-2 will be always opposite to Face-1, Face-5 will be opposite to Face-3 and, Face-6 will be opposite to Face-4. So, we need to just the vector in opposite direction.

Calculations of normal are done now we will move to a part for capturing elements that will be useful later.

3.4 Capturing Elements

Sometimes the user doesn't want to apply the entire program to alter face numbering to all elements that exist in a meshed geometry. To provide that functionality to the user we make use of element sets that can be defined in ABAQUS.

Element Set (*ELSET)*:

The element set contains the elements you've chosen. You can make element sets using native ABAQUS elements on a module-meshed part, orphan mesh elements, or elements on any instances of parts. This collection can contain items from a single part of numerous parts. The Set toolset will allow us to design sets that contain elements. We can request output or add loads to specified locations without destroying the mesh or partitioning the geometry using elements within sets.

Now we will get to the part of implementing the algorithm.

4. Implementation

Axes: {‘x’, ‘-x’, ‘y’, ‘-y’, ‘z’, ‘-z’}

Vectors: {‘x’: [1, 0, 0], ‘-x’: [-1, 0, 0], ‘y’: [0, 1, 0], ‘-y’: [0, -1, 0], ‘z’: [0, 0, 1], ‘-z’: [0, 0, -1]}

4.1 Undistorted Elements:

Calculations of normal faces is an easy task in the case of undistorted elements, normal will usually present in the direction of axes present in Set Axes, and the normal vector will be corresponding the value of the key of a dictionary called “Vectors” which will be a member of Set Axes.

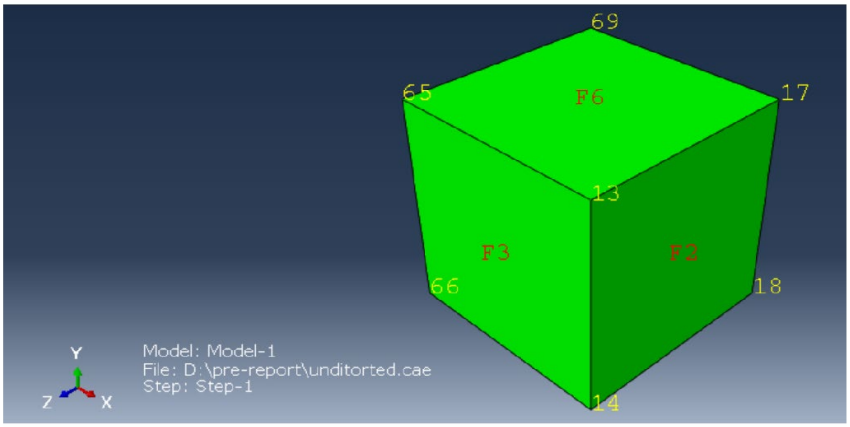


Fig 4.1 Undistorted Element

4.2 Distorted Elements:

Calculating the normal vector of the face representing a distorted element is an easy task but it gets tricky to assign the normal vector a direction among one of the axes presented. It’s an important task to assign the faces directions among one of the axes because it will determine which re-arrangement to apply and select one combination among the 4 final ones by calculating the normal to Face-3 and determining is it aligned with the user’s input axis.

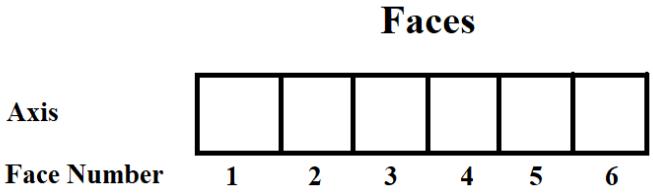


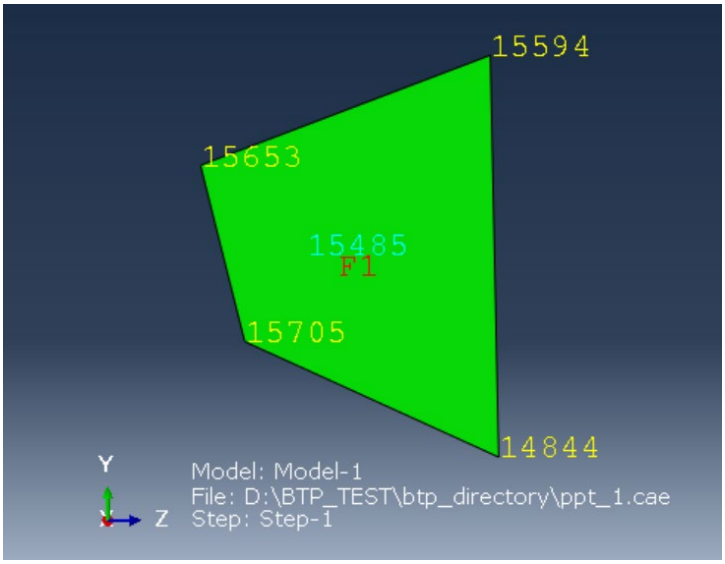
Fig 4.2 Faces [Face Number] = Axis
e.g.
Face-1 : 'x' - Faces[1]='x'
Face-3 : 'y' - Faces[3]='y'

We need to find an array that will contain the axes for particular faces as shown in figure 4.2

Let's consider an example of a distorted element and see how it works along with it we will see how sometimes highly distorted elements can cause trouble to calculate our array *"Faces"*.

4.3 Example:

YZ Plane:



Element Connectivity: 15485, 15594,14844,15705,15653, 20995,20245,21106,21054

Faces:

Faces

-x	x	z	-y	-z	y
1	2	3	4	5	6

9.3 Faces Array : Element -15485

We can visually see and create an array of "Faces" but we can't do that for every element our code needs to figure it out by itself. The first step is to calculate the angle made normal of faces-1,3,4 with Axes.

N1 = Normal Vector of Face-1

N3 = Normal Vector of Face-3

N4 = Normal Vector of Face-4

Code:

```
for i in [N1, N3, N4]:
    for j in Vectors :
        angle between i and Vectors[j]
```

For every element we will create a data frame of angles:

```
Element: 15485

Current :
          -x          -y          -z      x           y           z
Face
1        0.0   90.000000   90.000000  180.0   90.000000   90.000000
3       90.0   91.193713  178.806287   90.0   88.806287    1.193713
4       90.0   24.544572   65.455428   90.0  155.455428  114.544572
```

Fig. 4.3 Angles made by faces-1,3,4 and 'Axes'

If we apply the rule of taking the minimum of all the angles made by a face and the axes and then assign the axes in the “Face” array with which it makes the minimum. Let’s see if this works.

Local Minimum:

Faces

-x	x	z	-y	-z	y
1	2	3	4	5	6

Fig. 4.4 Faces array predicted by code using local minumum criteria for Element :15485

User Input: Face-1: +y; Face-3: +z

Now we know that currently, Face-6 is facing towards the +y axis, so we use the re-arrangement given in the figure 6.5 which will narrow down our combinations to 4, then we will use projections to avoid calculating arrays “Faces” using local minimum or global minimum criteria which might not work in some highly distorted elements. We will discuss it later in example 9. b). Now let’s see how we can make projections

Projections:

We can’t exactly call it projection but we are indeed projection Face’s on XY, YZ, and ZX planes using max and min of individual coordinates and updating it singularly.

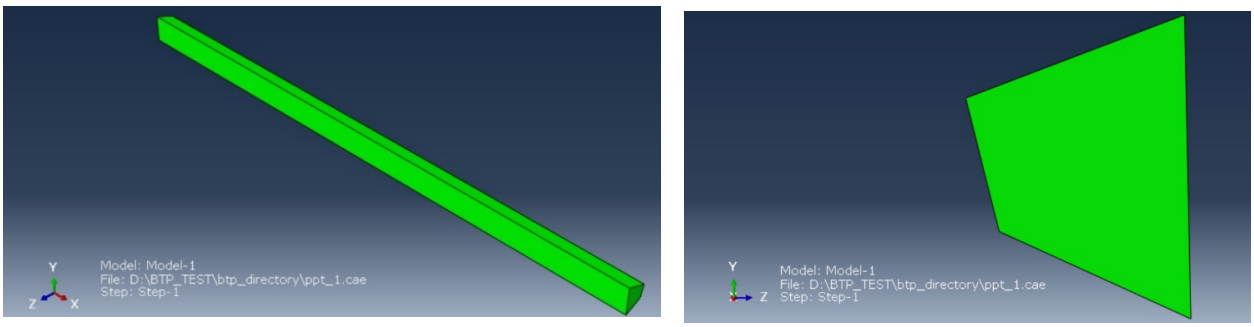


Fig. 4.5 i. Original Geometry

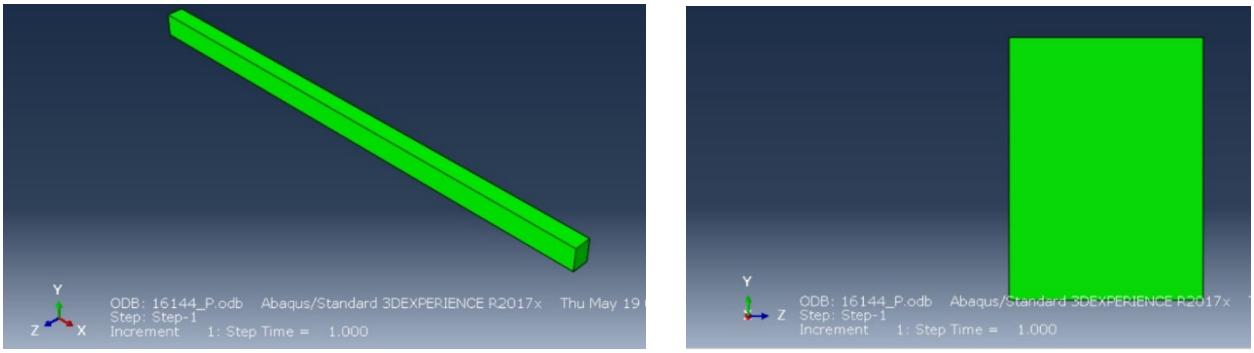


Fig. 4.5 **ii.** Projected Geometry

Now as our geometry has turned to an undistorted element, we can easily figure out the normal and their associated axes.

```
[[15485 15594 15653 21054 20995 14844 15705 21106 20245]
 [15485 20995 15594 15653 21054 20245 14844 15705 21106]
 [15485 21054 20995 15594 15653 21106 20245 14844 15705]
 [15485 15653 21054 20995 15594 15705 21106 20245 14844]]
```

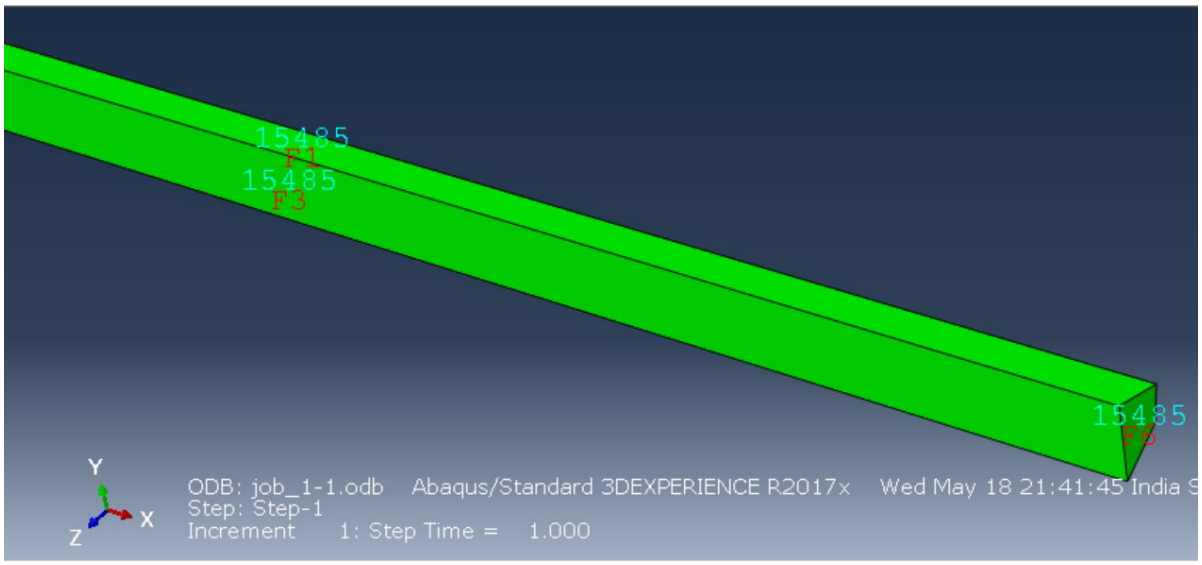
Fig. 4.6 Four combinations formed by fig.6.5 and function restricted_perm with -Y axis as stack direction as provided by user

```
Checked face-3 for connection :  [15485 20995 15594 15653 21054 20245 14844 15705 21106]
        -x     -y     -z      x     y     z
Face
1     90.0  180.0   90.0   90.0   0.0  90.0
3     90.0   90.0  180.0   90.0  90.0   0.0
4      0.0   90.0   90.0  180.0  90.0  90.0
Faces ['y', '-y', 'z', '-x', '-z', 'x']
```

Fig. 4.7 2) Checking Faces : Found

It's working.

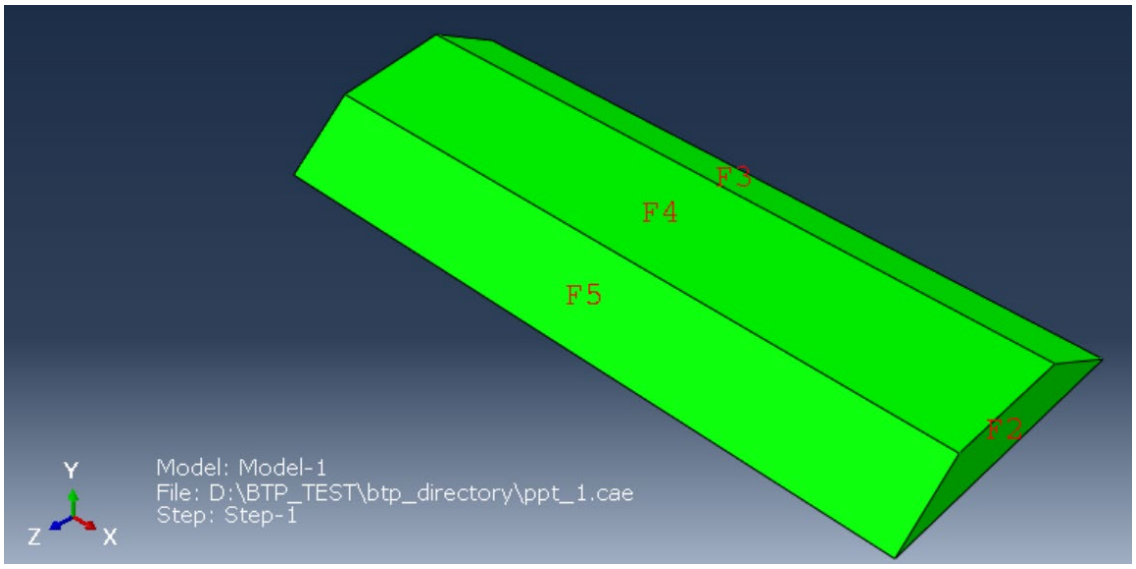
Output Geometry:



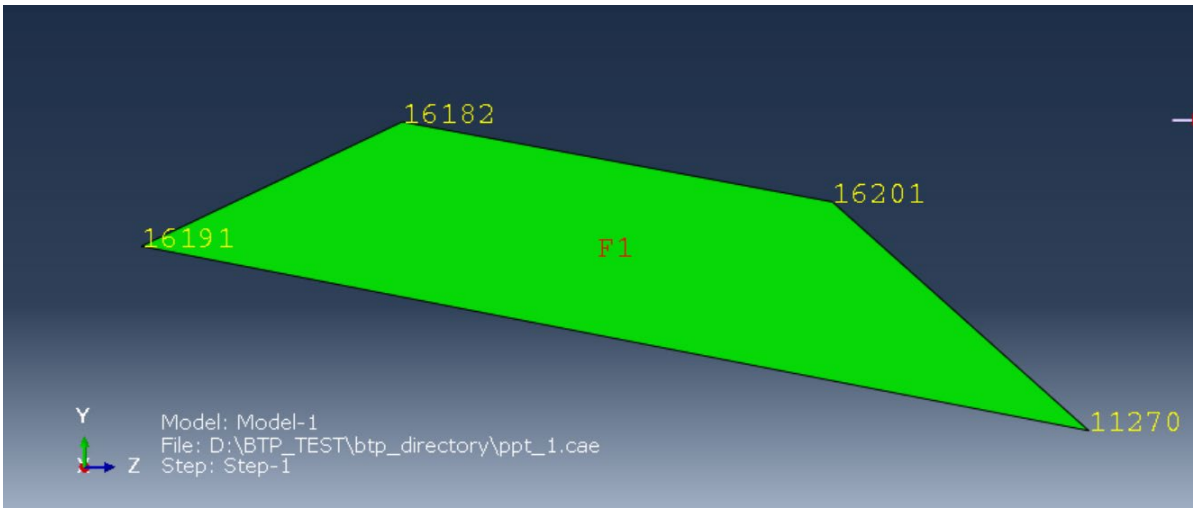
Let's try it for some other elements and see if the local minimum criteria work.

Example 9.b:

Isometric View:



YZ Plane:



Element Connectivity: 16154, 21602, 16201, 16182, 21583, 16671, 11270, 16191, 21592

Faces:

Faces

-x	x	-z	y	z	-y
1	**2**	**3**	**4**	**5**	**6**

Fig. 4.8 Faces Array : Element-16154

Angles Data Frame:

Element: 16154

Current :

Face	-x	-y	-z	x	y	z
1	0.0	90.000000	90.000000	180.0	90.000000	90.000000
3	90.0	154.437244	64.437244	90.0	25.562756	115.562756
4	90.0	169.423706	100.576294	90.0	10.576294	79.423706

Fig. 4.9 Angles made by faces-1,3,4 and 'Axes' : Element -16154

Local Minimum:

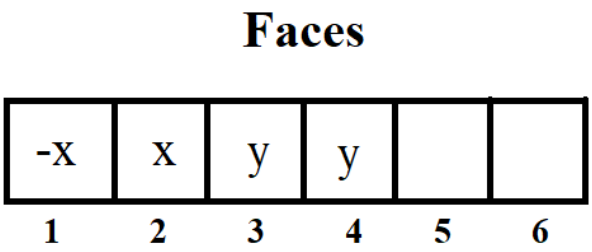


Fig. 4.10 Face-3 and Face-4 make a minimum angle with '+ve Y-axis'

Global Minimum:

Both Face-3 and Face-4 make minimum angles locally with the same axis i.e., a positive Y-axis. We can resolve it by comparing the global minimum value of Face-3 and Face-4 with +ve Y-axis and assigning this axis to the face which makes the minimum angle between them i.e., Face-4 so Face-4 will be assigned as +ve Y-axis and Face-3 will be assigned the next minimum angle locally i.e. -ve Z-axis.

Now ‘Faces ‘array will become as shown in 9.22,

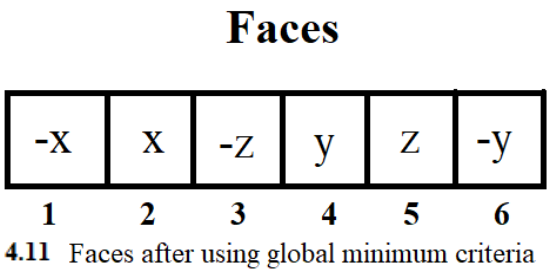


Fig. 4.11 Faces after using global minimum criteria

Alternate Faces:

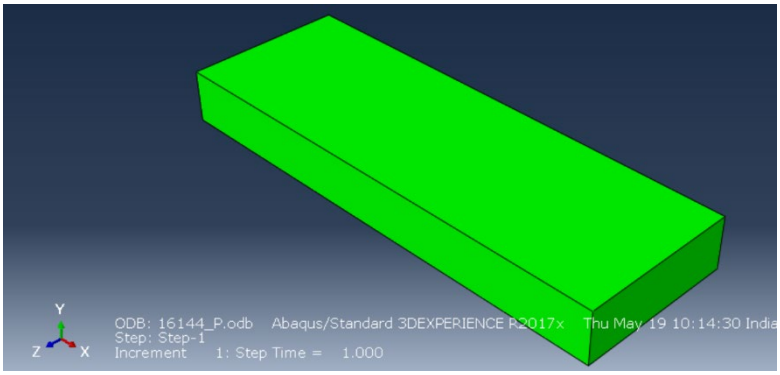
Sometimes global minimum criteria don’t work as well in that case we can use Face-5 and Face-6’s normal to determine the axes. Although only the Face-1 and Face-3 axis are required to obtain the “Faces” array due to distorted elements we have to consider every face, the alternate criteria will only work if clashes are present within Face-3 and Face-4.

Minimum First:

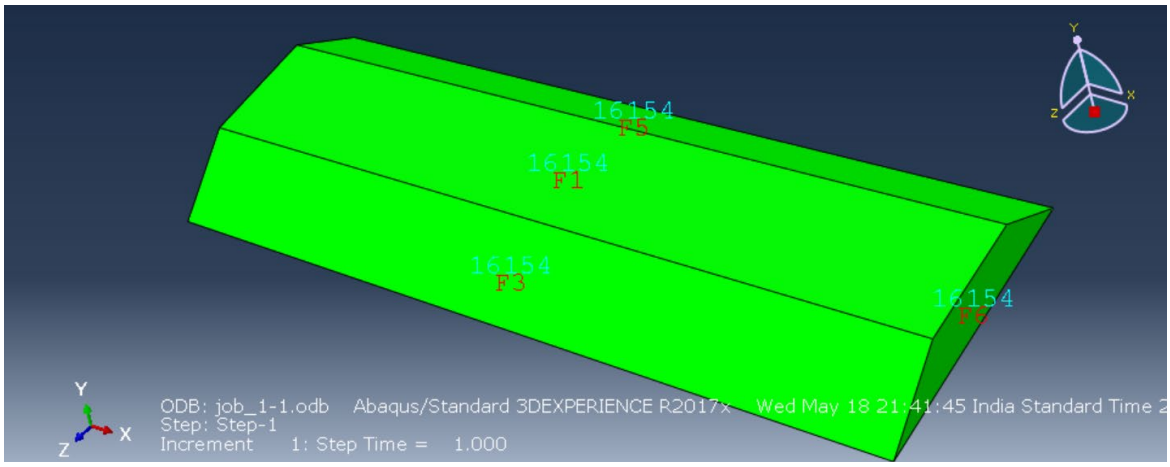
A most effective method is to start assigning axes to faces that make the least angle, we just need to specify three faces and we are done. I’ll illustrate this with an example

User Input: Face-1: +y; Face-3: +z

Projection:



Output Geometry:



***Note 1:** Face-3, Face-4, and Face-1 are supposed to be perpendicular to each other so they can't have axes assigned that are opposite to each other such as "+ve X" and "-ve X".

***Note 2:** The use of projections can be avoided by using local minimum again at the next step of face numbering where we have to pick amongst 4 combinations, the one combination which aligns Face-3 with the user's desired axis. We calculate in all4 cases angles made by Face-3 and input axis and select the combination which makes a minimum angle

***Note 3:** Many warning systems the alert if one of the elements is too distorted to determine the directions, in that case, the user will be asked e code will input axes which are aligned with the current element's Face-1 and Face-3, and from that information 'Faces' array will be created and further steps will be the same.

4.4 Key Notes (Code)

Python is the programming language used in the development stages of the project due to less availability of time, extensive libraries, and visualization.

Some of the methods used might not have the best time complexity but can be changed.

Warning systems might require some input when an error is occurred (only for highly distorted and rotated elements), the user needs to be handy with using ABAQUS and should have an understanding of how the algorithm created works to provide the input necessary.

Every line of code is explained using comments.

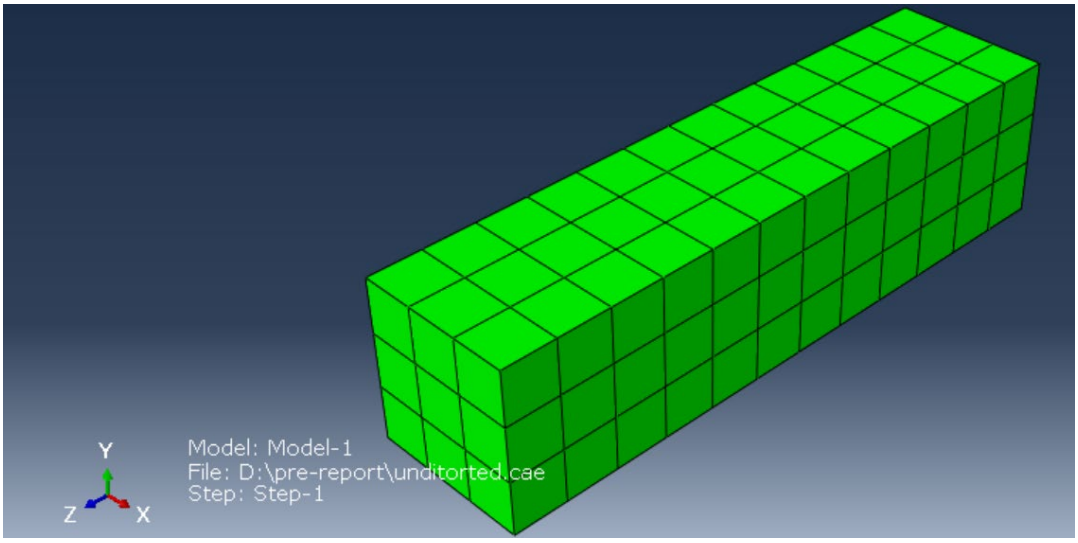
5. Results

Now we will apply our script to an input file that contains node and element definitions of a meshed part or geometry. The script was tested on many geometries but I have narrowed it down to 3. These parts contain undistorted, distorted, and 'rotated and distorted' elements.

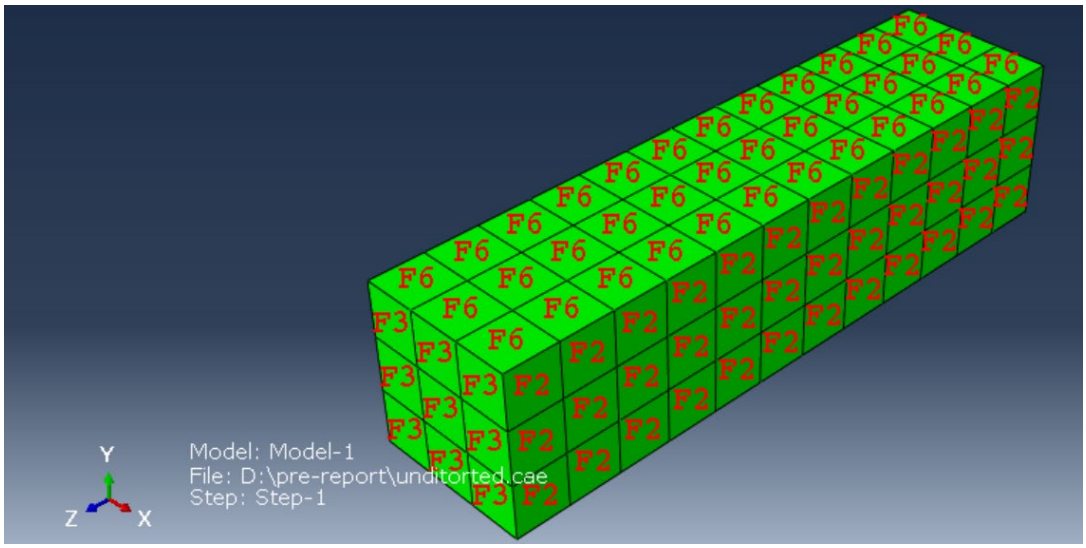
5.1 Part-1: (Undistorted elements):

a) Input Geometry (Meshed):

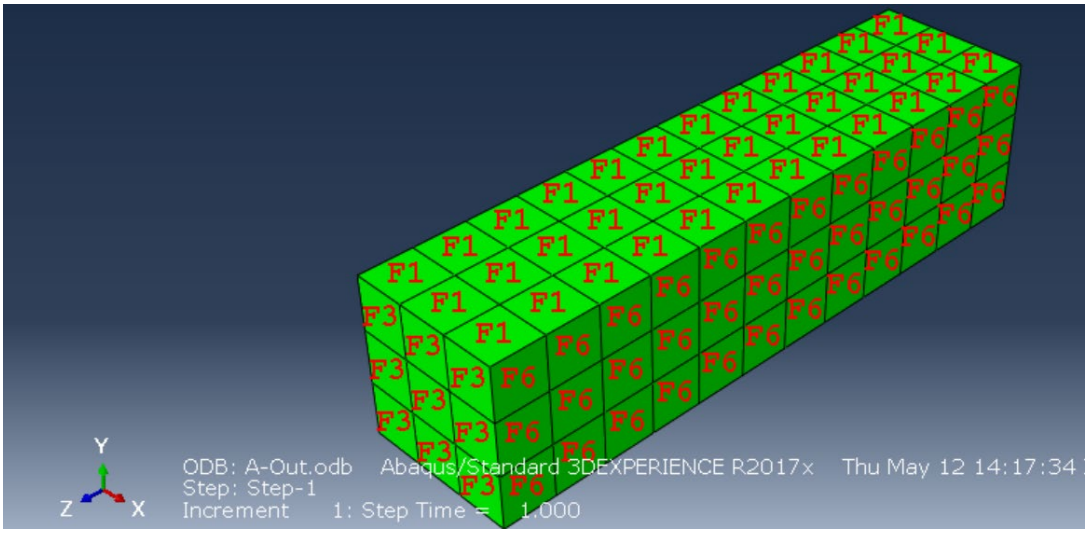
Isometric View



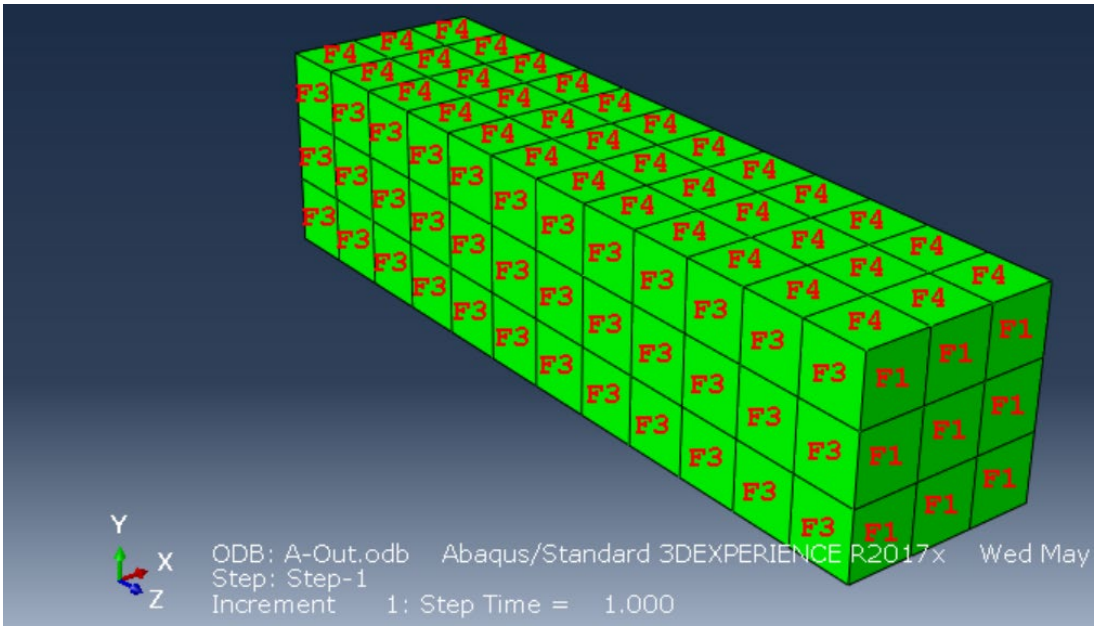
b) Input Geometry (Uniform Face Numbering)



c) Output Geometry (Meshed and Uniform)) [User Input: Face-1= '+ Y'; Face-3='+Z']



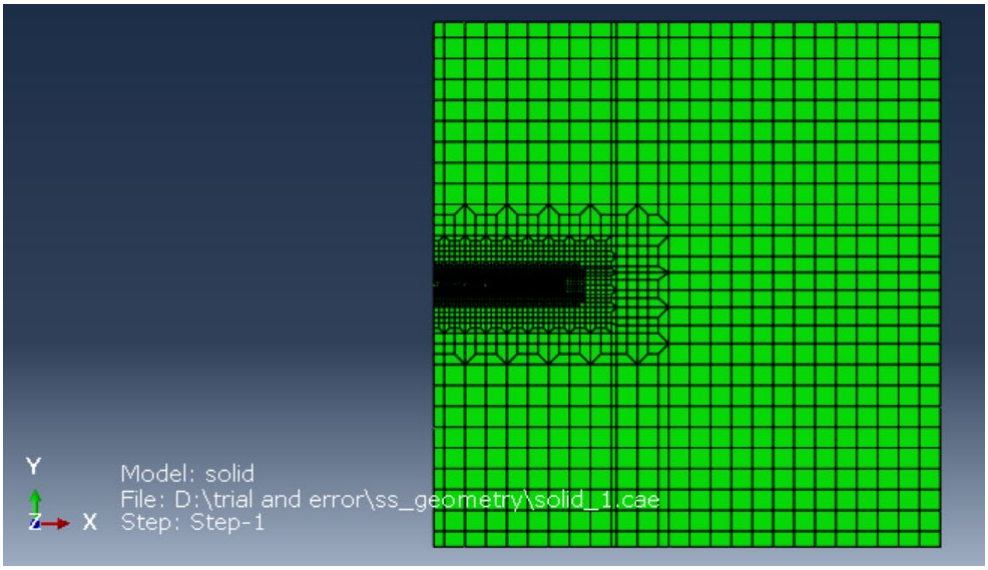
d) Output Geometry (Meshed and Uniform)) [User Input: Face-1= '+Z'; Face-3='-X']



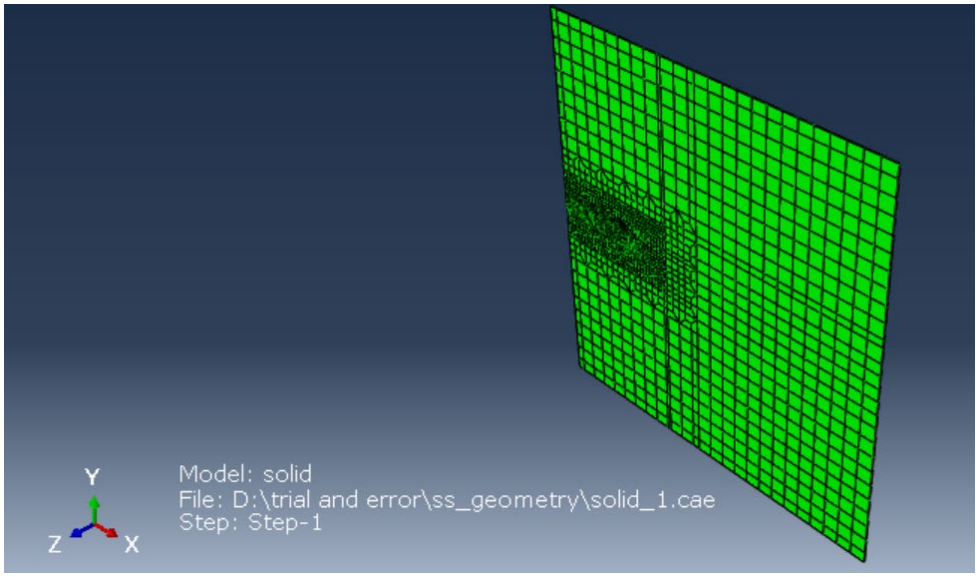
5.2 Part-2 :(Undistorted and Distorted Elements)

5.2.a) Input Geometry (Meshed):

X-Y plane

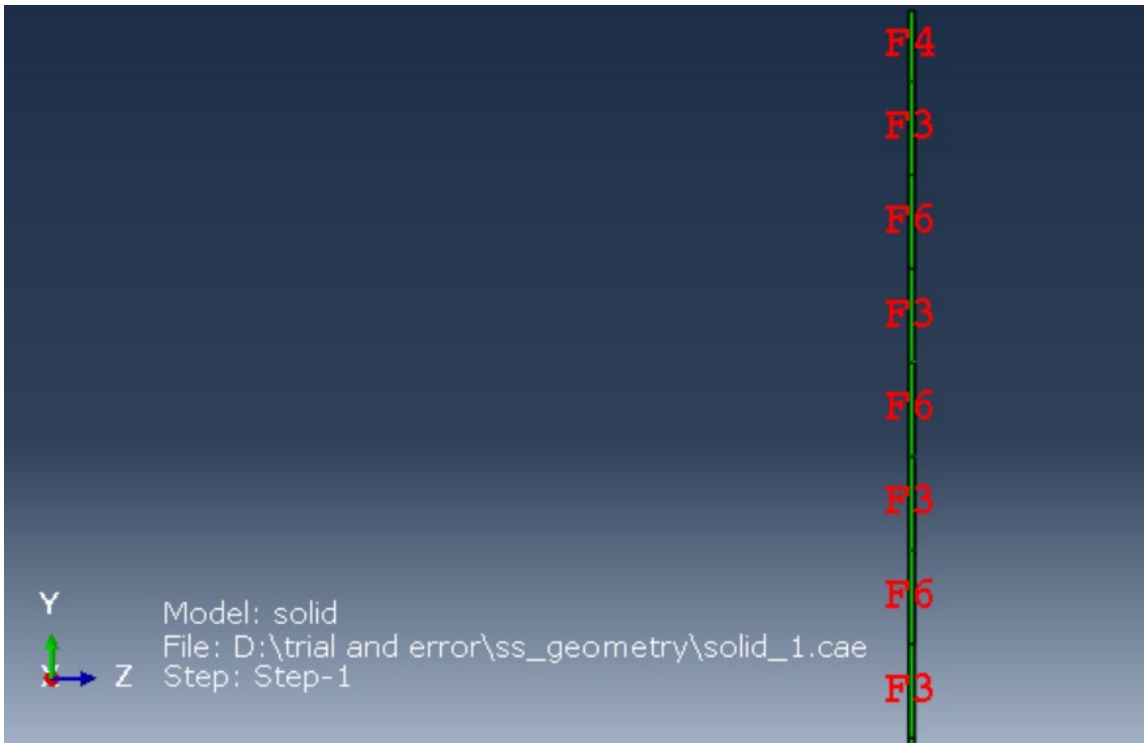


Isometric View



5.2.b) Input Geometry (Meshed and Non-uniform)

It is hard to show the entire geometry and point out where non-uniform face numbering exists, so only the area where non-uniform face numbering is present is shown.



Viewing from the Y-Z plane we can see that along the edges of the geometry we have non-uniform face numbering.

5.2.c) Output Geometry (Meshed and Uniform)

Now we have applied our script to the input file and changed the element connectivity. As a user Face-1 and Face-3 have been aligned in such a way that their normal point toward a positive Y-axis and positive Z-axis respectively.

So according to our input Face-4's normal should be pointing towards the negative X-axis.

Face-1: Positive Y-axis

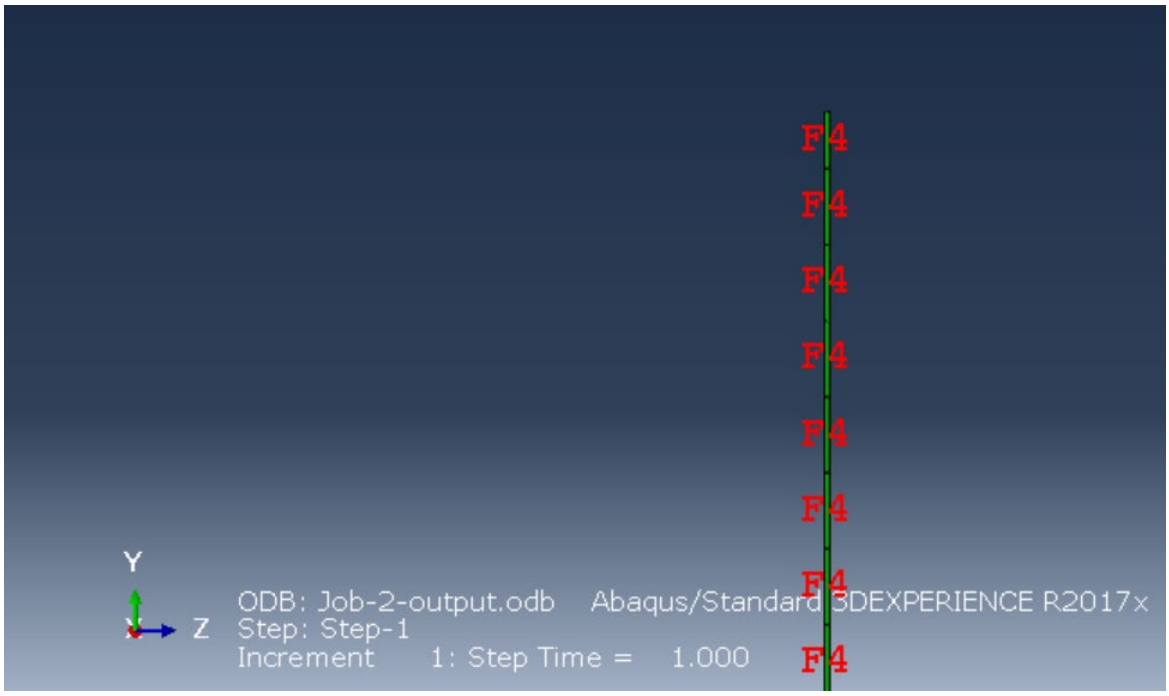
Face-2: Negative Y-axis

Face-3: Positive Z-axis

Face-4: Negative X-axis

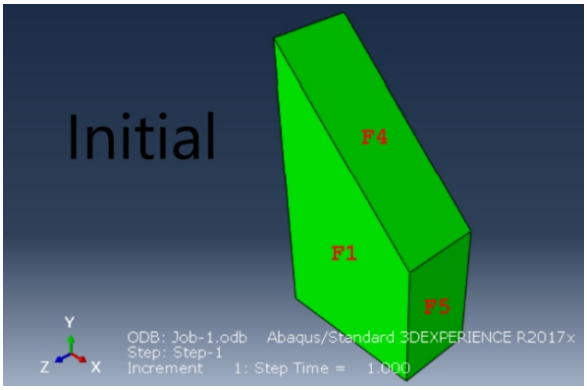
Face-5: Negative Z-axis

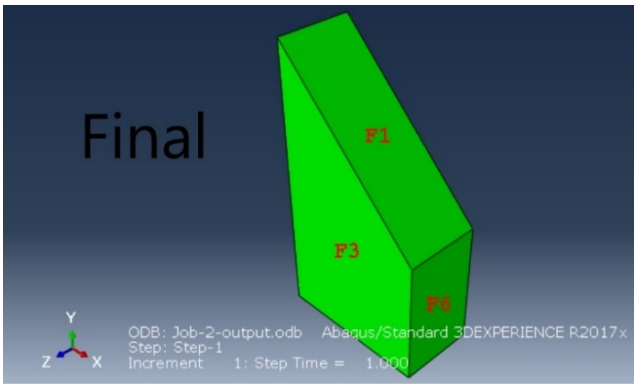
Face-6: Positive X-axis



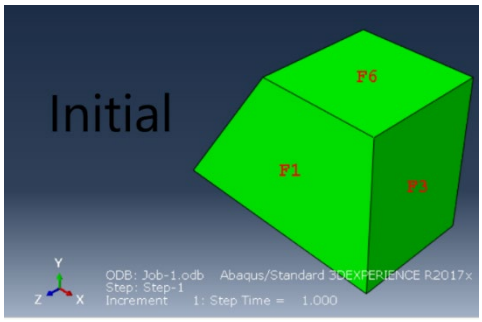
5.2.d) Individual Elements

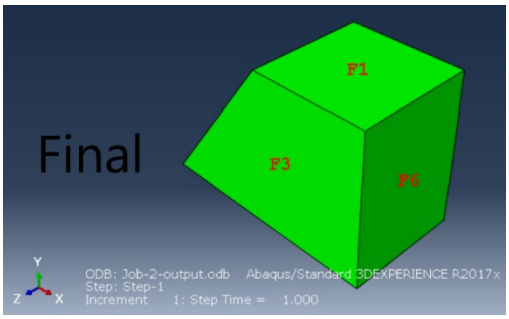
d.1) Element 2609 (Initial and Final) [User Input: Face-1= '+ Y'; Face-3='+Z']



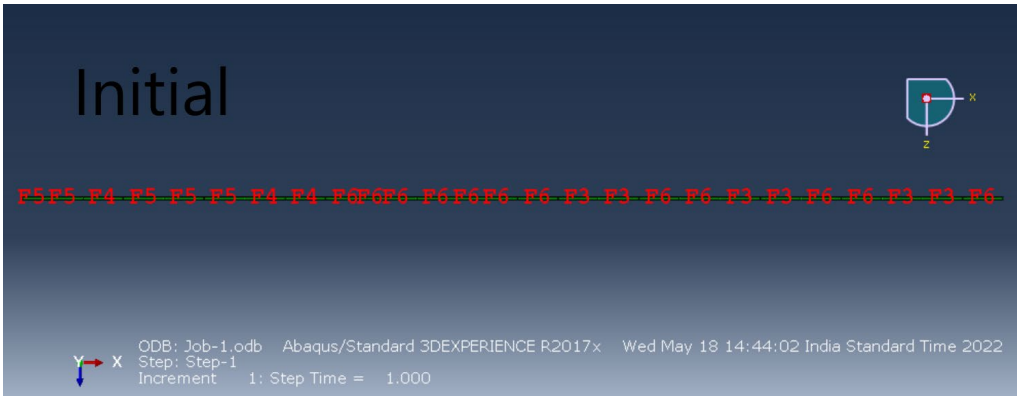


d.2) Element 1000 (Initial and Final) [User Input: Face-1= '+ Y'; Face-3='+Z']

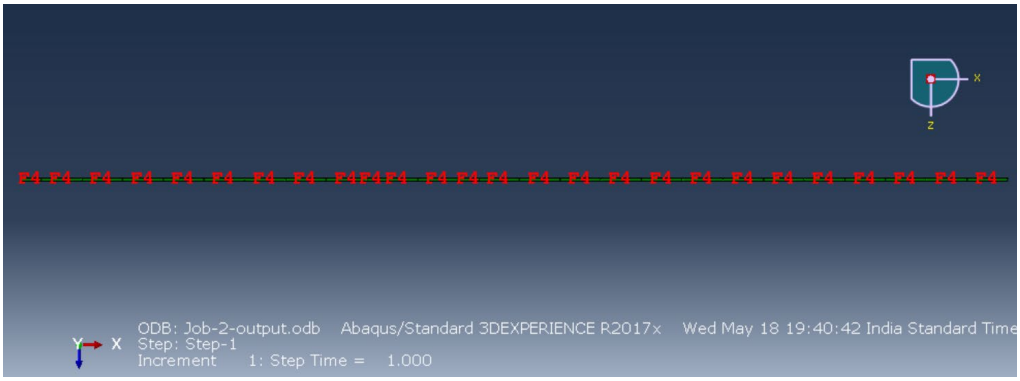




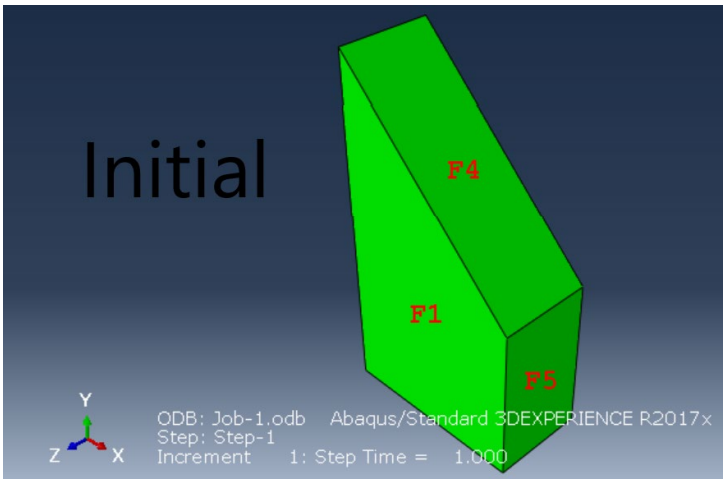
f) More Views 1: (Final) [User Input: Face-1= '+ Y'; Face-3='+Z']

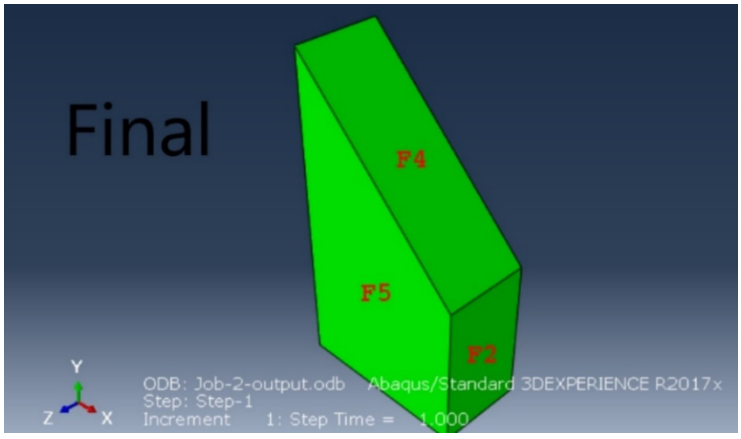


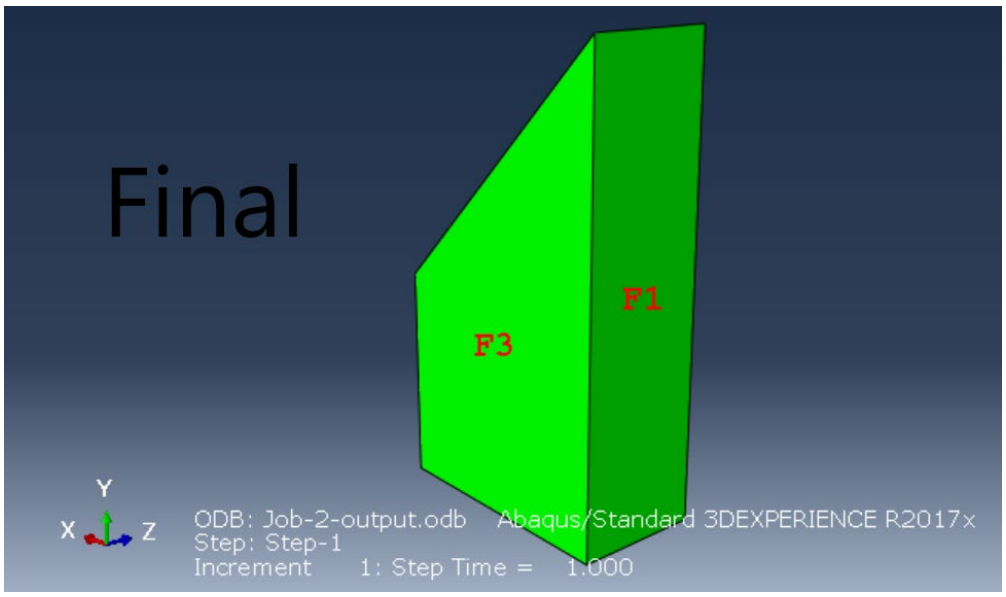
g) More Views 2: (Final) [User Input: Face-1= '-X'; Face-3='-Z']



h) Element 2609 (Initial and Final) [User Input: Face-1= '-X'; Face-3='-Z']



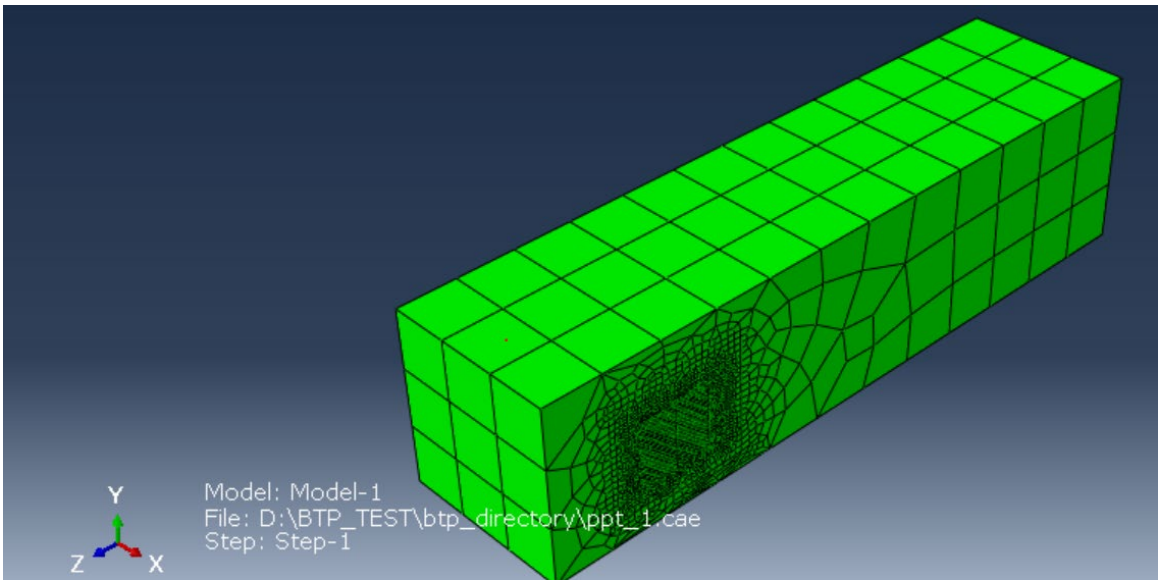




5.3 Part 3: (Distorted)

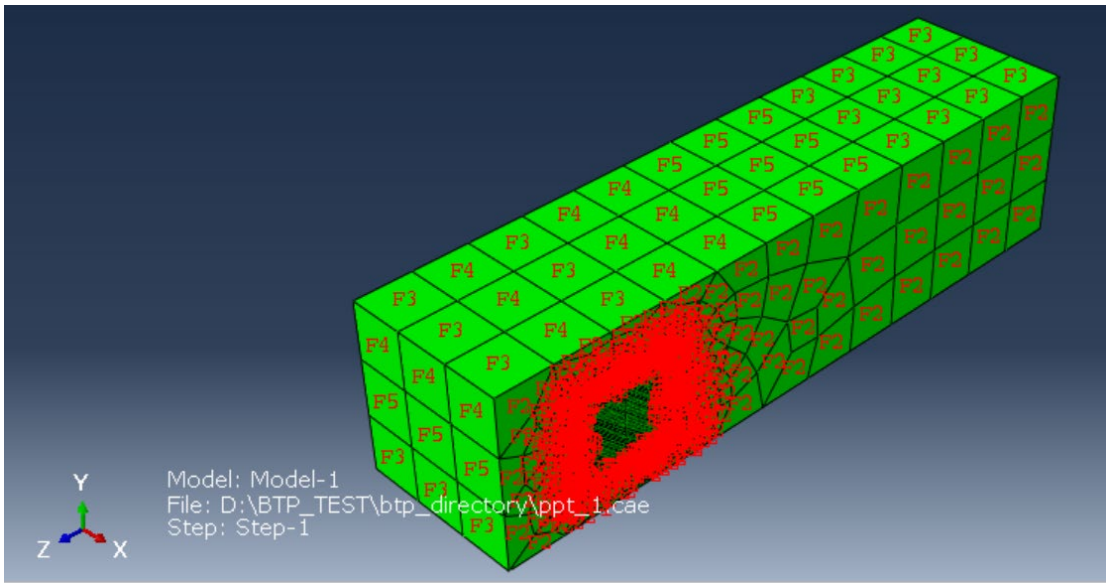
5.3.a) Input Geometry (Meshed):

Isometric View

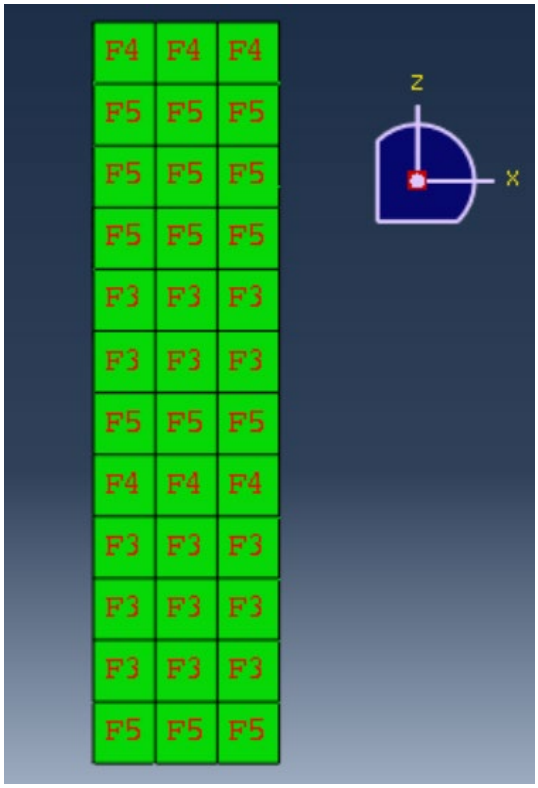


5.3.b) Input Geometry (Meshed and Non-Uniform):

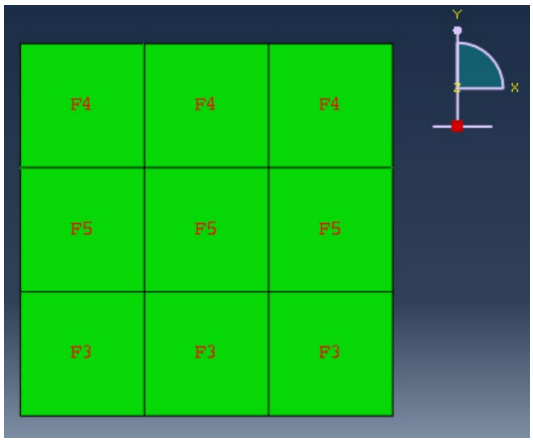
Isometric View



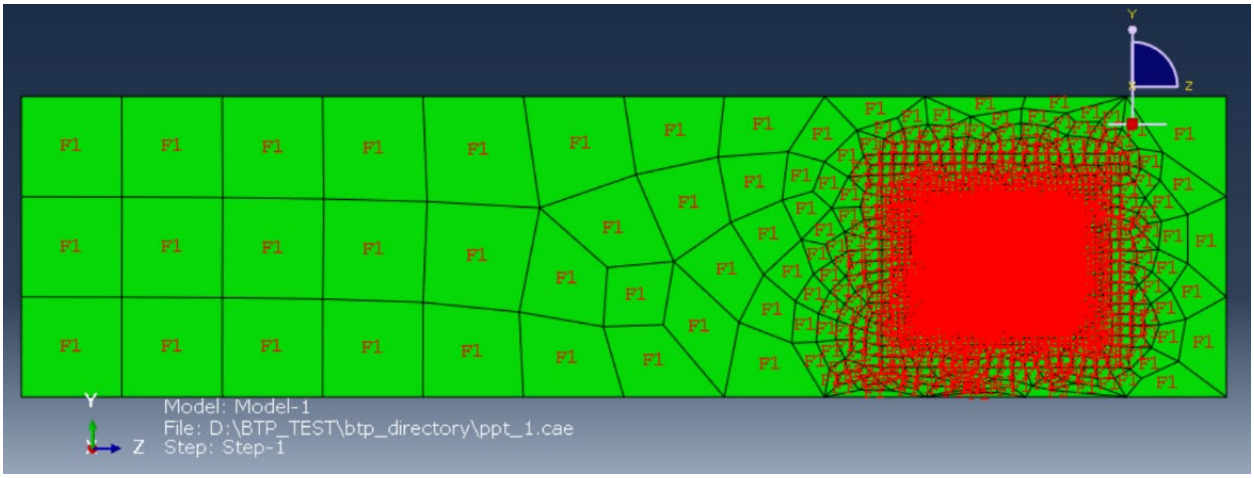
X-Z Plane:



X-Y Plane:

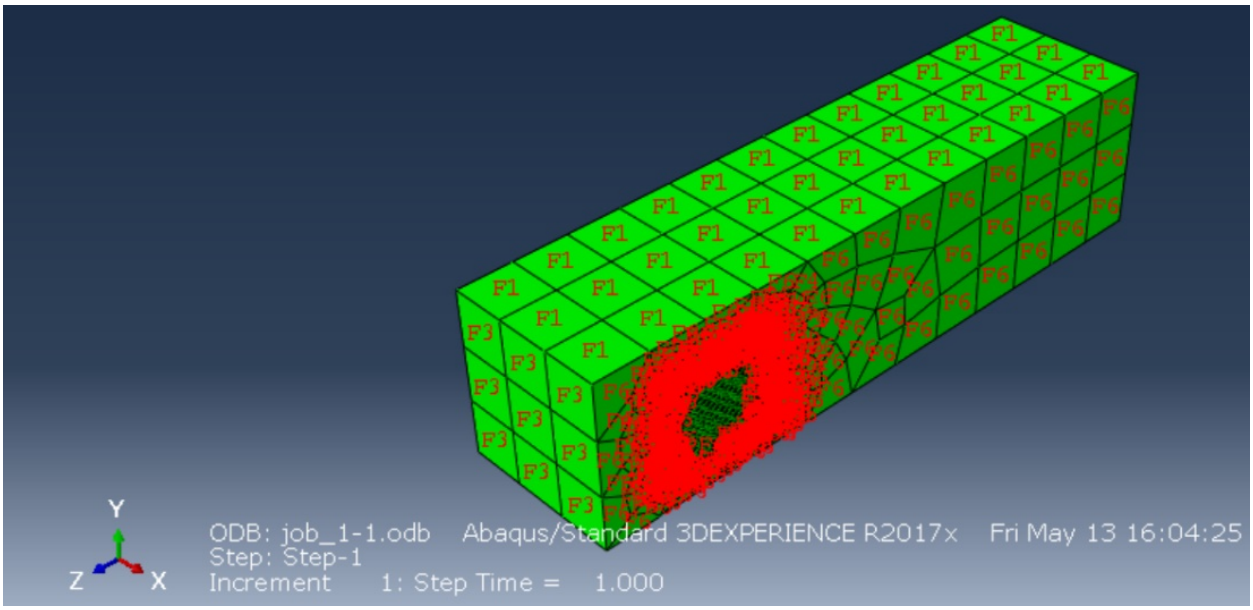


X-Z Plane: (This plane has uniform face numbering already)

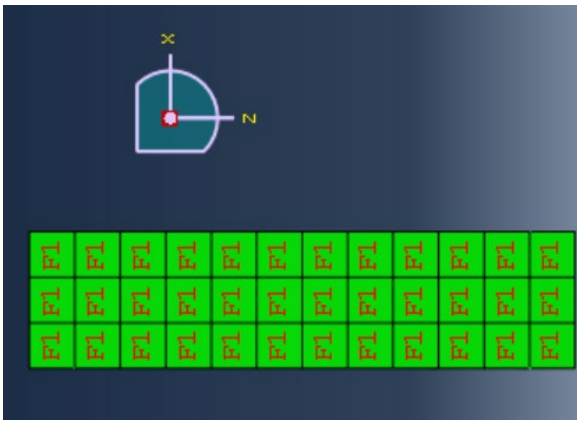


5.3.c) Output Geometry (Meshed and Uniform Face Numbering)

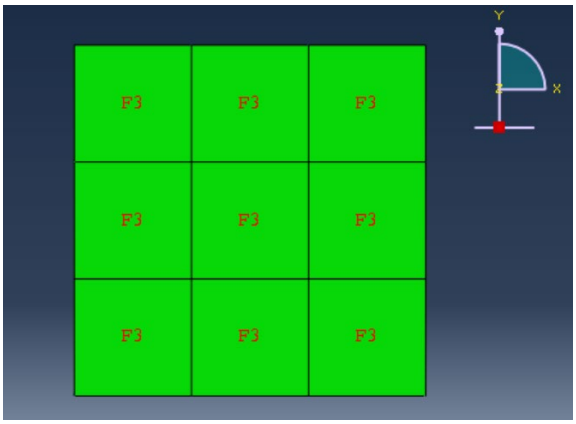
Isometric View:



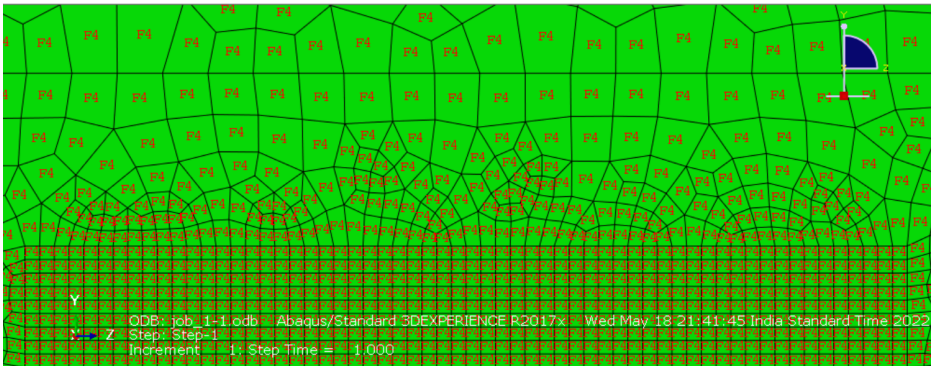
X-Z Plane



X-Y Plane:



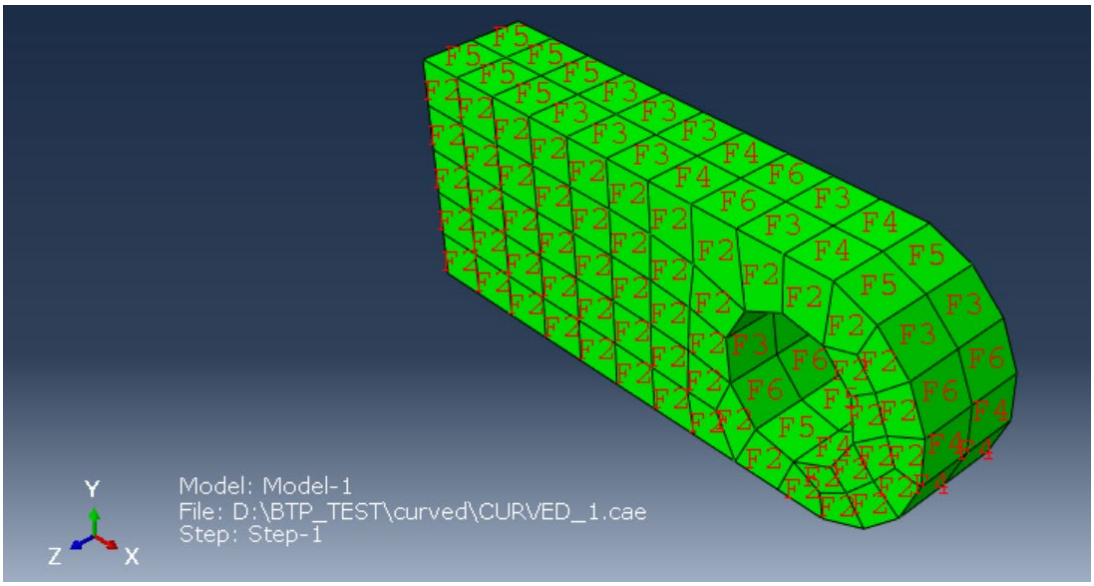
X-Z Plane



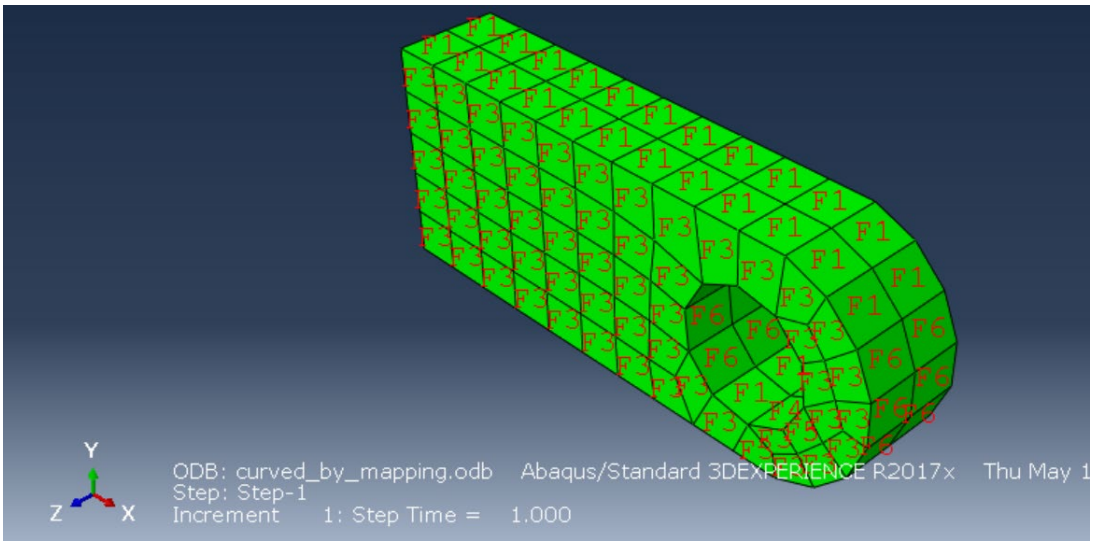
The above image is zoomed to the part where there is a transition in mesh size so, at its boundaries, we will find more distorted elements, and have seen the code works fine on it.

5.4 Part 4 (Distorted and Rotated):

5.4.a) Input Geometry (Meshed and Non-uniform):



5.4.b) Output Geometry (Meshed and Uniform) * Exceptions


Y
Z
X
ODB: curved_by_mapping.odb Abaqus/Standard 3DEXPERIENCE R2017x Thu May 1
Step: Step-1
Increment 1: Step Time = 1.000

6. Conclusions

1: Developed python script works for distorted and undistorted elements, it also works for rotating elements with some errors.

2: To avoid clashes Face-3, Face-4, and Face-1 are supposed to be perpendicular to each other so they can't have axes assigned that are opposite to each other such as "+ve X" and "-ve X" or "+ve Y" and "-ve Y".

3: The use of projections can be avoided by using local minimum again at the next step of face numbering where we have to pick amongst 4 combinations, the one combination which aligns Face-3 with the user's desired axis. We calculate in all 4 cases angles made by Face-3 and input axis and select the combination which makes a minimum angle

4: Many warnings systems alert if one of the elements is too distorted to determine the directions, in that case, the user will be asked to input axes that are aligned with the current element's Face-1 and Face-3, from that information 'Faces' array, will be created and further steps will be the same

5: Mapping used in FEM can't work for making uniform face numbering in the global coordinate system we have to calculate the initial array "Faces". Although face numbering remains uniform in a local coordinate system.

6: Instead of projection we can use mapping as well, we just need to figure out the homes in the code will are the basis vectors that form the respective systems and are mapped in local and global coordinates to each other. It'll be clearer when we discuss mapping in the future scope.

7. Future Scope

7.1 FEM: Coordinate Transformation

We all are familiar with the coordinate transformation used in FEM using shape functions. We map a distorted element into a uniform geometry in a local coordinate system and perform all our numerical integration and mathematical calculations on the uniform geometry which makes it easier for computation.

We will use the same method to change face numbering. First, let's explain in brief how the mapping is done.

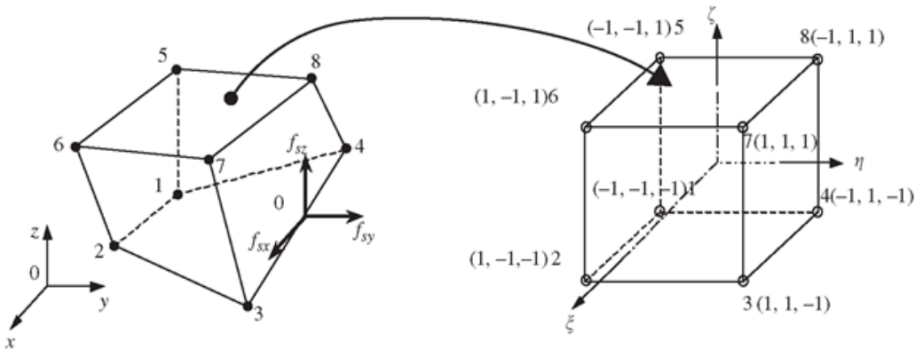


Fig. 7.1 Mapping : Distorted Element in cartesian coordinates to local coordinate system.

To develop this particular mapping, we need shape functions. The derivation of this particular mapping is cited in references I'll just pick some images containing the equations.

$$x = \sum_{i=1}^{8} N_i(\xi, \eta, \zeta) x_i$$

$$y = \sum_{i=1}^{8} N_i(\xi, \eta, \zeta) y_i$$

$$z = \sum_{i=1}^{8} N_i(\xi, \eta, \zeta) z_i$$

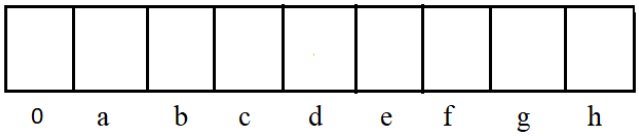
Fig. 7.2 Calculating global node coordinates corresponding to the local node coordinates

$$N_1 = \frac{1}{8}(1-\xi)(1-\eta)(1-\zeta)$$
$$N_2 = \frac{1}{8}(1+\xi)(1-\eta)(1-\zeta)$$
$$N_3 = \frac{1}{8}(1+\xi)(1+\eta)(1-\zeta)$$
$$N_4 = \frac{1}{8}(1-\xi)(1+\eta)(1-\zeta)$$
$$N_5 = \frac{1}{8}(1-\xi)(1-\eta)(1+\zeta)$$
$$N_6 = \frac{1}{8}(1+\xi)(1-\eta)(1+\zeta)$$
$$N_7 = \frac{1}{8}(1+\xi)(1+\eta)(1+\zeta)$$
$$N_8 = \frac{1}{8}(1-\xi)(1+\eta)(1+\zeta)$$

Fig. 7.3 Shape Functions

Now we don't need to use these equations for our purpose. You can easily notice that the choice of (x1, y1, z1) among our provided 8 nodes of your hexahedron element is arbitrary. Whatever we choose as (x1, y1, z1) will become Node-1 in the local coordinate system. Let's illustrate this clearly.

Fig. 7.4 Element Connectivity



Suppose we are given element connectivity in form of an array Indices have been named a little differently, because of the local node numbering that is always going to be present in the set {1,2,3,4,5,6,7,8}. As we don't have any other information about the orientation of the element currently, we are going to use the following convention for the selection:

- a – (x_1, y_1, z_1)
- b – (x_2, y_2, z_2)
- c – (x_3, y_3, z_3)
- d – (x_4, y_4, z_4)
- e – (x_5, y_5, z_5)
- f – (x_6, y_6, z_6)
- g – (x_7, y_7, z_7)

- h – (x_8, y_8, z_8)

Now this will always result in:

- Node a ↔ Node 1
- Node b ↔ Node 2
- Node c ↔ Node 3
- Node d ↔ Node 4
- Node e ↔ Node 5
- Node f ↔ Node 6
- Node g ↔ Node 7
- Node h ↔ Node 8

Coordinates of the nodes present in the local system are given as:

Node 1: (-1, -1, -1); Node 2: (1, -1, -1); Node 3: (1, 1, -1); Node 4: (-1, 1, -1)

Node 5: (-1, -1, 1); Node 6: (1, -1, 1); Node 7: (1, 1, 1); Node 8: (-1, 1, 1).

Generally, this would work great to change face numbering only given the input mesh is uniform, but when the face numbering is not uniform Face-1 and Face-2 directions are not the same for all elements, so when we change face numbering in the call system the reflections in the original system are different let's illustrate using an example.

Case :1

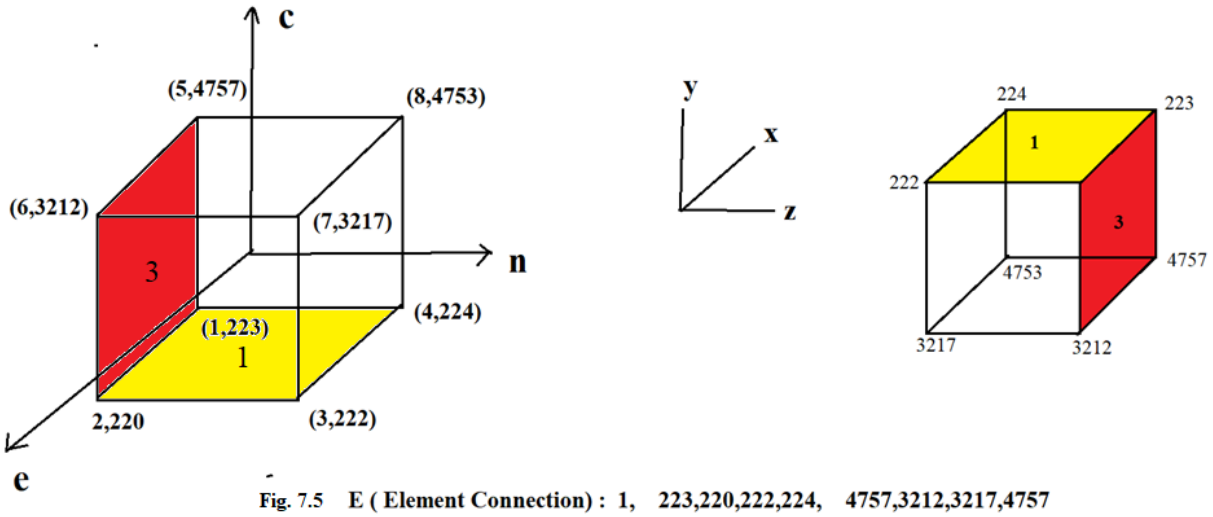
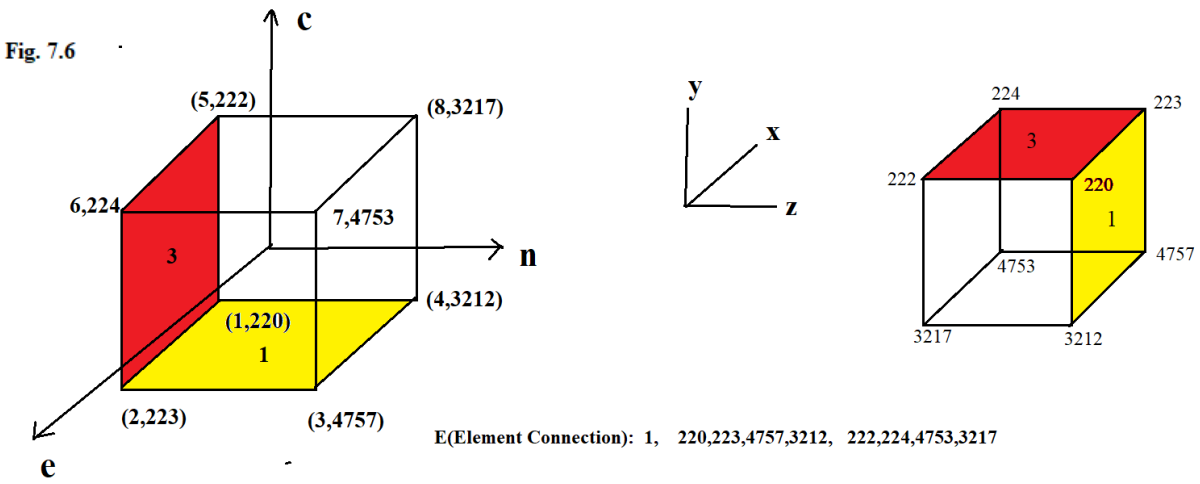


Fig. 7.5 E (Element Connection) : 1, 223,220,222,224, 4757,3212,3217,4757

Case:2



In the above example we can see how a similar element with different face numbering can result in uniform face numbering in the local system but different in the global one. This transformed hexahedron element formed will be used to change face numbering, we have taken input from the user in the form axes for Face-1 and Face-3 in a global coordinate system. But we need to know what changes should be made in the local system so that we could reflect the desired changes in the global system.

For that, we need the array "Faces" calculated accurately. We will need information about the directions of normal Face-1 and Face-3 in both global and local systems. We need to calculate the "Faces" array in both systems.

We can resolve it by making a relation between the basis spaces as stated. In case 2, the relation is

{z ↔ -c; -z ↔ c; y ↔ -n; x ↔ e; -y ↔ n; -x ↔ -e;}

Note: Originally, we just see which axis is Face-N pointing to in the original and local system

Now, all we need to do is see which axes correspond to the user's input and use it to change the face numbering in the local system and we are done. This causes concern for rotated elements; we need to know how an element is rotated concerning the reference element, we don't know how ABAQUS does node numbering, is there a local node numbering and connectivity? we don't know. But by using the method of Minimum First we can avoid clashes and still obtain uniform face numbering.

The problem of rotated elements remains.

7.2 Faster Code

Python was used especially because of its extremely useful libraries like Pandas, NumPy, Matplotlib, and hash tables that are already implemented in the form of dictionaries. Python is an example of a high-level language. It manages details of a program like memory allocation, memory deallocation, pointers, etc itself and this is what allows writing codes in Python effortless for users. The code is initially compiled into python byte code. Then Byte Code interpreter conversion happens internally and most of this conversion is hidden from the user. Byte code is platform-independent and lower-level programming. Compilation of byte code is to step up the execution of source code. The source code is then compiled to byte code which is executed in Python's virtual machine one by one, to carry out the all the necessary operations. One thing to note is that the virtual machine is an internal component of Python.

Internally a python code is interpreted during run time rather than being compiled to native code hence it is a slower than C/C++. We can use C++ to make our code run faster. Also an important thing to note is that some of the implementations of task are not done optimally due to time restrictions and brute force method is used , also some tasks were not necessary for changing the element connectivity but as the project was in development stage it was necessary for visualization purposes

8. References

1. MIT Information Systems and Technology (2017), ABAQUS Documentation, https://abaqus-docs.mit.edu
2. *Dassault Systems* (2017), ABAQUS Documentation, http://130.149.89.49:2080/v2016/index.html
3. Python 3.10.4 Documentation
4. The Finite Element Method in Engineering 6th edition, Singiresu S. Rao
5. What-when-how (In depth tutorials and information)

   FEM for 3D Solids (Finite Element Method) Part 2 (what-when-how.com)