3.1.3.14. ASDEmbeddedNode Element

This command is used to construct an ASDEmbeddedNodeElement object.

The ASDEmbeddedNodeElement is a constraint between **one constrained** node and **many retained** nodes. Since in OpenSees a Multi-Point constraint can have only one retained node, this constraint was implemented as an Element, thus imposing the constraint using the Penalty approach.

It constrains the displacements of the constrained node (NC) to be the **weighted average** of the displacements of the surrounding retained nodes (NRi). The same is done with the infinitesimal rotation, if the constrained node has rotational DOFs.

The constrained node should be inside (or on the boundary of) the domain defined by the retained nodes.

```
element ASDEmbeddedNodeElement $eleTag $Cnode $Rnode1 $Rnode2 $Rnode3 <$Rnode4> <-K $K> <-rot>
```

Argument	Type	Description
$eleTag	*integer*	unique integer tag identifying element object.
$Cnode	*integer*	the constrained node
$Rnode1 $Rnode2 $Rnode3 <$Rnode4>	3 or 4 *integer*	the 3 (or 4) retained nodes defining the surrounding triangle (or tetrahedron) domain.
-K	*string*	optional flag. if provided, the user should use a user-defined penalty stiffness.
$K	*float*	optional float, mandatory if -K flag is provided (default = 1.0e18). This is the penalty stiffness used to impose the constraint. This value is automatically multiplied by the area (or volume) of the surrounding domain, so that it is mesh-independent. You can estimate this value to be equal to, or slightly larger than the Young's modulus of the material used for the surrounding domain. Do not use extremely large values, otherwise the system will be ill-conditioned.
-rot	*string*	optional flag. if provided, and if the constrained node has rotational DOFs, its rotation will be constrained as well.

This element can be used in both 2D and 3D problems:

- **2D Problem:**
  - 3 retained nodes forming a triangle.
  - retained and constrained nodes can have 2 or 3 DOFs.
  - if the constrained node has 3 DOFs, its rotation can be constrained as well.

- **3D Problem:**
    - 3 retained nodes forming a triangle (if the surrounding domain is a Shell).
    - 4 retained nodes forming a tetrahedron (if the surrounding domain is a Solid).
    - retained and constrained nodes can have 3, 4 or 6 DOFs.
    - if the constrained node has 6 DOFs, its rotation can be constrained as well.

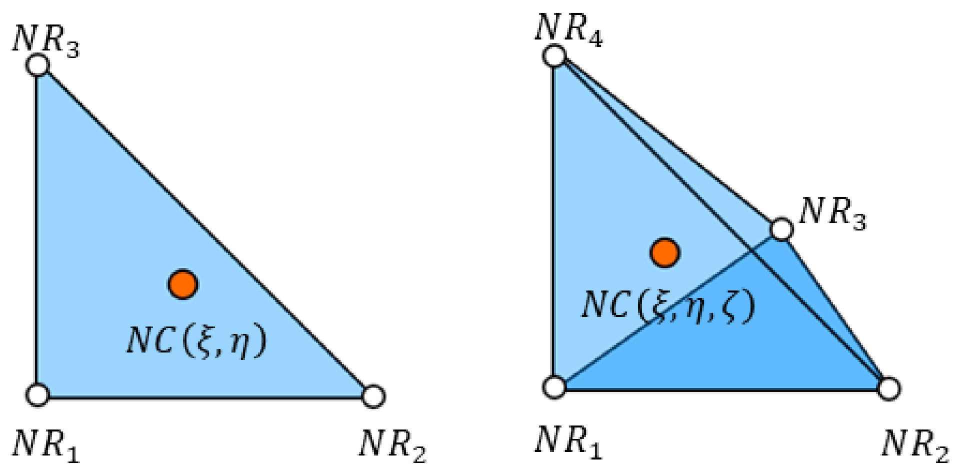

Fig. 3.1.3.11 Embedding geometry defined by the retained nodes. (Left) triangle for 2D continuum problems or 3D shell problems. (Right) tetrahedron for 3D continuum problems.

The constraint equations, imposed by this element, are the following for the 2D case. Similar equations can be easily derived for the 3D case:

$$UC_x = \sum_{i=1}^{3} N_i(\xi,\eta)\, UR_{x,i}$$
$$UC_y = \sum_{i=1}^{3} N_i(\xi,\eta)\, UR_{y,i}$$
$$\theta C_z = \sum_{i=1}^{3} \left(\frac{\partial N_i(\xi,\eta)}{\partial X} UR_{y,i} - \frac{\partial N_i(\xi,\eta)}{\partial Y} UR_{x,i} \right) /2$$

Note

- Even though the support domain can be only a triangle or a tetrahedron, it does not mean that you cannot embed a node in a quadrilateral or hexaedron. You just need to find the sub-triangle (for the quadrilateral) or the sub-tetrahedron (for the hexaedron) that contains the constrained node.

Example

1. **Tcl Code**

```tcl
# 2D problem with 2 DOFs on both the constrained node and the retained nodes
# The embedding domain is a 1x1 triangle, and the constrained node is placed at its centroid.
# Here we apply a random displacement on each retained node,
# and the displacement of the constrained node should be the weighted average
# of the displacements at the 3 retained nodes, with an equal weight = 1/3.

model basic -ndm 2 -ndf 2

# define the embedding domain (a piece of a soild domain)
node 1 0.0 0.0
node 2 1.0 0.0
node 3 0.0 1.0

# define the embedded node
node 4 [expr 1.0/3.0] [expr 1.0/3.0]

# define constraint element
element ASDEmbeddedNodeElement 1   4   1 2 3   -K 1.0e6

# apply random imposed displacement in range 0.1-1.0
set U1 [list [expr 0.1 + 0.9*rand()] [expr 0.1 + 0.9*rand()]]
set U2 [list [expr 0.1 + 0.9*rand()] [expr 0.1 + 0.9*rand()]]
set U3 [list [expr 0.1 + 0.9*rand()] [expr 0.1 + 0.9*rand()]]
puts "Applying random X displacement:\nU1: $U1\nU2: $U2\nU3: $U3\n\n"
timeSeries Constant 1
pattern Plain 1 1 {
   for {set i 1} {$i < 3} {incr i} {
      sp 1 $i [lindex $U1 [expr $i - 1]]
      sp 2 $i [lindex $U2 [expr $i - 1]]
      sp 3 $i [lindex $U3 [expr $i - 1]]
   }
}

# run analysis
constraints Transformation
numberer Plain
system FullGeneral
test NormUnbalance 1e-08 10 1
algorithm Linear
integrator LoadControl 1.0
analysis Static
analyze 1

# compute expected solution
set UCref [list [expr ([lindex $U1 0] + [lindex $U2 0] + [lindex $U3 0] )/3.0] [expr ([lindex $U1
1] + [lindex $U2 1] + [lindex $U3 1] )/3.0]]
puts "Expected displacement at constrained node is (U1+U2+U3)/3:\n$UCref\n\n"

# read results
set UC [list {*}[nodeDisp 4]]
puts "Obtained displacement at constrained node is UC:\n$UC\n\n"

# check error
set ER [list [expr abs([lindex $UC 0] - [lindex $UCref 0])/[lindex $UCref 0]] [expr abs([lindex $UC
1] - [lindex $UCref 1])/[lindex $UCref 1]]]
puts "Relative error is abs(UC-UCref)/UCref:\n$ER\n\n"
```

2. Python Code

```python
# 2D problem with 2 DOFs on both the constrained node and the retained nodes
# The embedding domain is a 1x1 triangle, and the constrained node is placed at its centroid.
# Here we apply a random displacement on each retained node,
# and the displacement of the constrained node should be the weighted average
# of the displacements at the 3 retained nodes, with an equal weight = 1/3.
from opensees import *
from random import random as rand

model('basic', '-ndm', 2, '-ndf', 2)

# define the embedding domain (a piece of a soild domain)
node(1, 0.0, 0.0)
node(2, 1.0, 0.0)
node(3, 0.0, 1.0)

# define the embedded node
node(4, 1.0/3.0, 1.0/3.0)

# define constraint element
element('ASDEmbeddedNodeElement', 1,   4,   1, 2, 3,   '-K', 1.0e6)

# apply random imposed displacement in range 0.1-1.0
U1 = [0.1 + 0.9*rand(), 0.1 + 0.9*rand()]
U2 = [0.1 + 0.9*rand(), 0.1 + 0.9*rand()]
U3 = [0.1 + 0.9*rand(), 0.1 + 0.9*rand()]
print('Applying random X displacement:\nU1: {}\nU2: {}\nU3: {}\n\n'.format(U1,U2,U3))
timeSeries('Constant', 1)
pattern('Plain', 1, 1)
for i in range(1, 3):
   sp(1, i, U1[i - 1])
   sp(2, i, U2[i - 1])
   sp(3, i, U3[i - 1])


# run analysis
constraints('Transformation')
numberer('Plain')
system('FullGeneral')
test('NormUnbalance', 1e-08, 10, 1)
algorithm('Linear')
integrator('LoadControl', 1.0)
analysis('Static')
analyze(1)

# compute expected solution
UCref = [
   (U1[0] + U2[0] + U3[0])/3.0,
   (U1[1] + U2[1] + U3[1])/3.0
   ]
print('Expected displacement at constrained node is (U1+U2+U3)/3:\n{}\n\n'.format(UCref))

# read results
UC = nodeDisp(4)
print('Obtained displacement at constrained node is UC:\n{}\n\n'.format(UC))

# check error
ER = [
   abs(UC[0] - UCref[0])/UCref[0],
   abs(UC[1] - UCref[1])/UCref[1]
   ]
print('Relative error is abs(UC-UCref)/UCref:\n{}\n\n'.format(ER))
```

Code Developed by: **Massimo Petracca** at ASDEA Software, Italy.