OpenSEES Model-Building Commands I

Silvia Mazzoni
University of California, Berkeley

OpenSees User Workshop

14 August 2006

ModelBuilder Objects

- model Command
- node Command
- mass Command
- Constraints objects
- uniaxialMaterial Command
- nDMaterial Command
- section Command
- element Command
- block Command
- region Command
- Geometric Transformation Command
- Time Series
- pattern Command

Tcl & OpenSEES commands

- Comand syntax:

command arg1 arg2 ...; # comment

example Tcl command:

set a 1; # assign value of 1 to a

set b [expr 2*$a];

example OpenSEES command:

node 1 10. 10. -mass $Mnode 0 0

Model Command

This command is used to construct the BasicBuilder object.

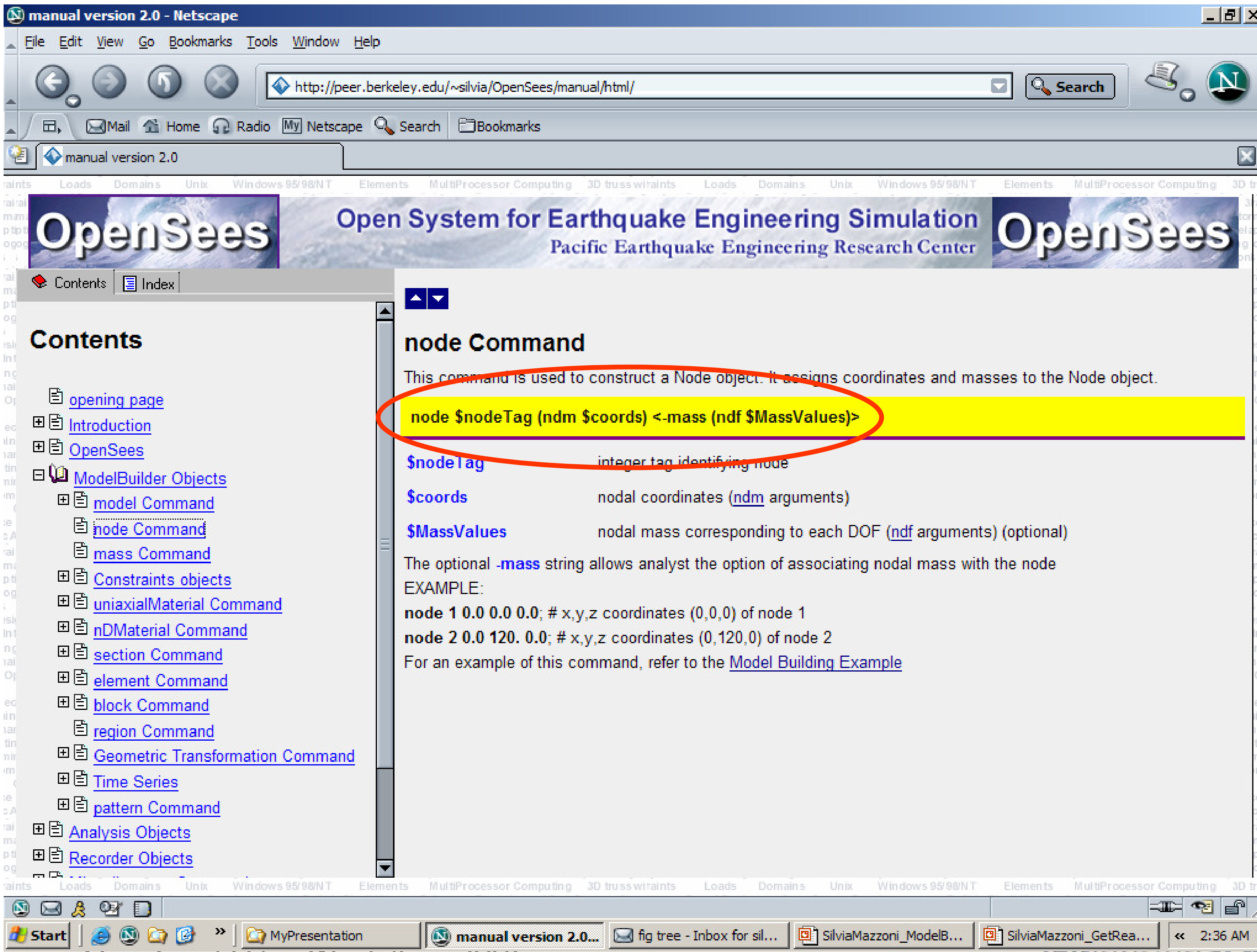
```
model BasicBuilder -ndm $ndm <-ndf $ndf>
```

$ndm dimension of problem (1,2 or 3)

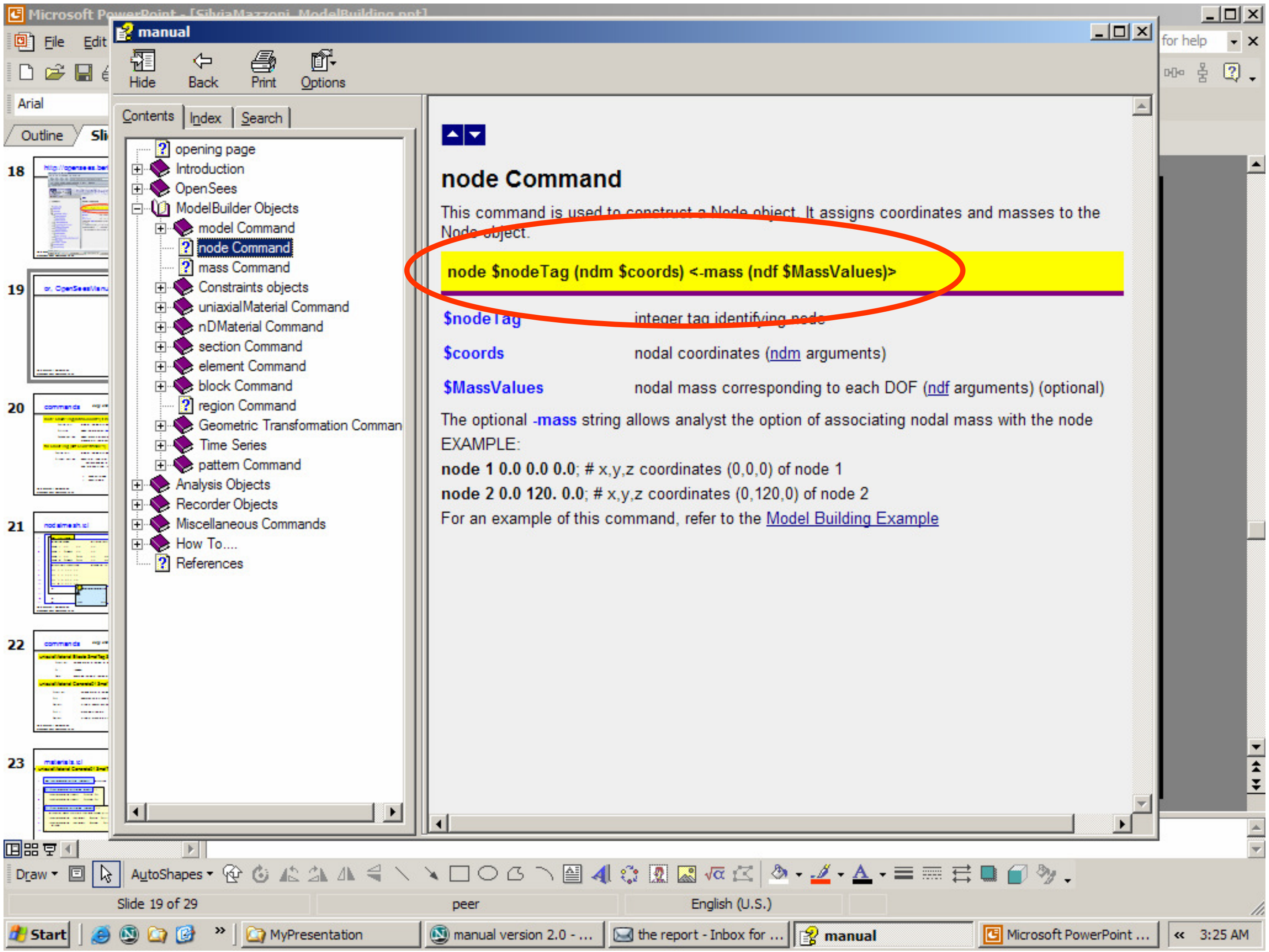
$ndf number of degrees of freedom at node (optional)
(default value depends on value of ndm:
ndm=1 -> ndf=1
ndm=2 -> ndf=3
ndm=3 -> ndf=6)

```
model BasicBuilder -ndm 3 -ndf 6
```

http://opensees.berkeley.edu



or, OpenSeesManual.chm



sample command

node Command

This command is used to construct a Node object. It assigns coordinates and masses to the Node object.

node $nodeTag (ndm $coords) <-mass (ndf $MassValues)>

$nodeTag	integer tag identifying node
$coords	nodal coordinates (ndm arguments)
$MassValues	nodal mass corresponding to each DOF (ndf arguments) (optional)

The optional **-mass** string allows analyst the option of associating nodal mass with the node

EXAMPLE:

node 1 0.0 0.0 0.0; # x,y,z coordinates (0,0,0) of node 1

node 2 0.0 120. 0.0; # x,y,z coordinates (0,120,0) of node 2

For an example of this command, refer to the Model Building Example

nodes and boundary conditions

copy and paste from manual:

node $nodeTag (ndm $coords) <-mass (ndf $MassValues)>

$nodeTag	integer tag identifying node
$coords	nodal coordinates (ndm arguments)
$MassValues	nodal mass corresponding to each DOF (ndf arguments) (optional)

fix $nodeTag (ndf $ConstrValues)

$nodeTag	integer tag identifying the node to be constrained
$ConstrValues	constraint type (0 or 1). ndf values are specified, corresponding to the ndf degrees-of-freedom. The two constraint types are: **0** unconstrained **1** constrained

Nodal coordinates and BC

```
# Define nodes;                  # frame is in X-Y plane
node  1   0.0       0.0          0.0
node  2   $Lbeam    0.0          0.0
node  3   0.0       $Lcol        0.0        -mass $Mnode 0.0 0.0 0.0 0.0 0.0
node  4   $Lbeam    $Lcol        0.0        -mass $Mnode 0.0 0.0 0.0 0.0 0.0
# Boundary conditions;           # node DX DY DZ RX RY RZ ! 1: fixed, 0: released
fix   1   1   1   1   1   1   1;
fix   2   1   1   1   1   1   1
fix   3   0   0   1   1   1   0
fix   4   0   0   1   1   1   0
#                                  3------------------------4
#                                   |                        |
#                                   |                        |
#                                 --1--                    --2--
```

coordinates & mass

boundary conditions

materials

copy and paste from manual:

uniaxialMaterial Elastic $matTag $E <$eta>

$matTag	unique material object integer tag
$E	tangent
$eta	damping tangent (optional, default=0.0)

uniaxialMaterial Concrete01 $matTag $fpc $epsc0 $fpcu $epsU

$matTag	unique material object integer tag
$fpc	compressive strength*
$epsc0	strain at compressive strength*
$fpcu	crushing strength*
$epsU	strain at crushing strength*

***NOTE:** Compressive concrete parameters should be input as negative values.

tcl if statement

```
if {logical statement} {
    ….series of commands…..
}
```

materials

concrete

uniaxialMaterial Concrete01 $matTag $fpc $epsc0 $fpcu $epsU

```
set ConcreteMaterialType "inelastic"  # options: "elastic","inelastic"

if {$ConcreteMaterialType == "elastic"} {
    uniaxialMaterial Elastic $Idcore $Ec
    uniaxialMaterial Elastic $Idcover $Ec
}
if {$ConcreteMaterialType == "inelastic"} {
# uniaxial Kent-Scott-Park concrete model w/ linear unload/reload, no T strength (-ve comp.)
    uniaxialMaterial Concrete01 $IDcore   $fc1C  $eps1C  $fc2C  $eps2C;   # Core
    uniaxialMaterial Concrete01 $IDcover  $fc1U  $eps1U  $fc2U $eps2U;   # Cover
}
```

uniaxialMaterial Hysteretic $matTag $s1p $e1p $s2p $e2p <$s3p $e3p> $s1n $e1n $s2n $e2n <$s3n $e3n> $pinchX $pinchY $damage1 $damage2 <$beta>

$matTag	unique material object integer tag
$s1p $e1p	stress and strain (or force & deformation) at *first* point of the envelope in the *positive* direction
$s2p $e2p	stress and strain (or force & deformation) at *second* point of the envelope in the *positive* direction
$s3p $e3p	stress and strain (or force & deformation) at *third* point of the envelope in the *positive* direction (optional)
$s1n $e1n	stress and strain (or force & deformation) at *first* point of the envelope in the *negative* direction*
$s2n $e2n	stress and strain (or force & deformation) at *second* point of the envelope in the *negative* direction*
$s3n $e3n	stress and strain (or force & deformation) at *third* point of the envelope in the *negative* direction (optional)*
$pinchX	pinching factor for strain (or deformation) during reloading
$pinchY	pinching factor for stress (or force) during reloading
$damage1	damage due to ductility: $D_1(mu-1)$
$damage2	damage due to energy: $D_2(E_{ii}/E_{ult})$
$beta	power used to determine the degraded unloading stiffness based on ductility, mu^{-beta} (optional, default=0.0)

materials

reinforcing steel

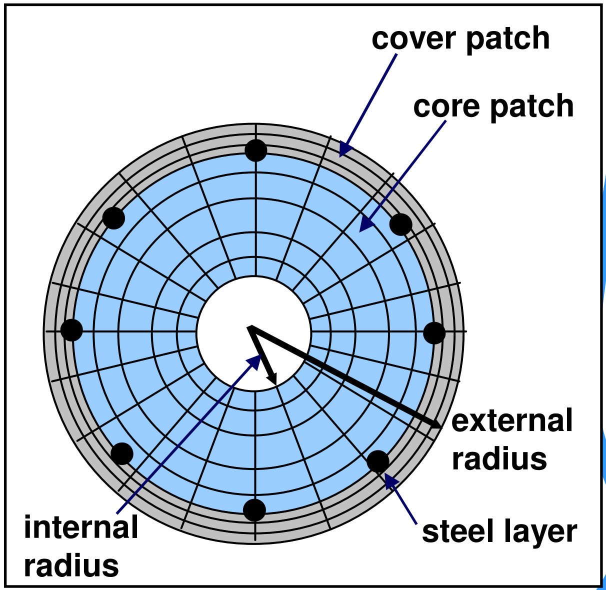
```
set SteelMaterialType "hysteretic";

if {$SteelMaterialType == "elastic"} {
   uniaxialMaterial Elastic $IDsteel $Es
}
if {$SteelMaterialType == "bilinear"} {
   uniaxialMaterial Steel01 $Idsteel $Fy $Es $Bs
}
if {$SteelMaterialType == "hysteretic"} {
   uniaxialMaterial Hysteretic $IDsteel $Fy $epsY $Fy1 $epsY1 $Fu $epsU -$Fy -$epsY -$Fy1 -$epsY1 -$Fu -$epsU $pinchX $pinchY $damage1 $damage2 $betaMUsteel
}
```

fiber section command

```
section Fiber $secTag {
        fiber <fiber arguments>
        patch <patch arguments>
        layer <layer arguments>
}
```

```
fiber $yLoc $zLoc $A $matTag
```

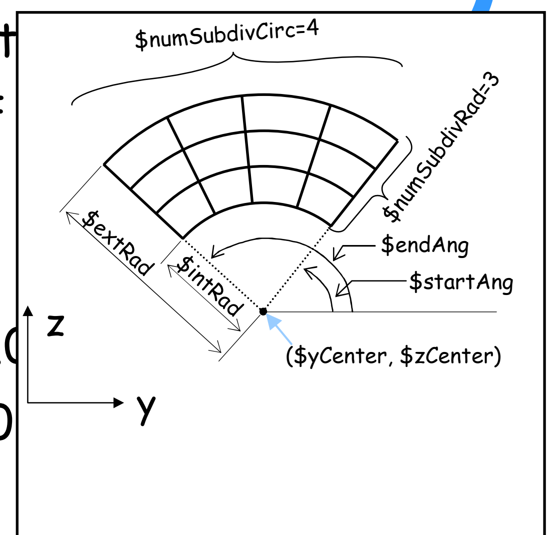


$yLoc	y coordinate of the fiber in the section (local coordinate system)
$zLoc	z coordinate of the fiber in the section (local coordinate system)
$A	area of fiber
$matTag	material tag of the pre-defined UniaxialMaterial object used to represent the stress-strain for the area of the fiber

section command (cont.)

patch circ $matTag $numSubdivCirc $numSubdivRad $yCenter $zCenter $intRad $extRad <$startAng $endAng>

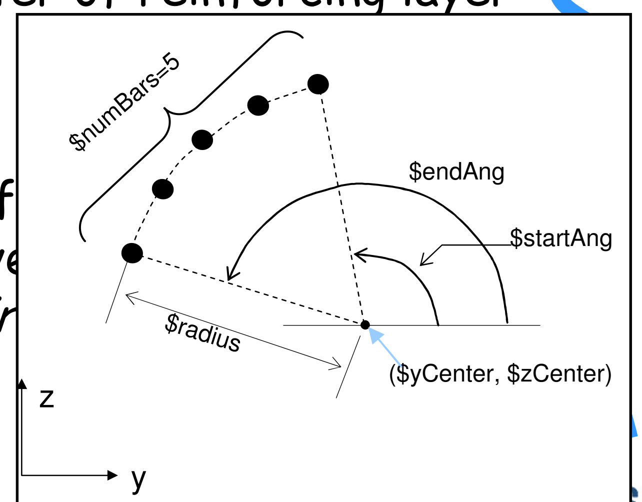
$matTag	material integer tag of the previously-defined UniaxialMaterial object used to represent the stress-strain for the area of the fiber
$numSubdivCirc	number of subdivisions (fibers) in the circumferential direction.
$numSubdivRad	number of subdivisions (fibers) in t
$yCenter $zCenter	y & z-coordinates of the center of
$intRad	internal radius
$extRad	external radius
$startAng	starting angle (optional. default=0.(
$endAng	ending angle (optional. default=360



section command (cont.)

layer circ $matTag $numBar $areaBar $yCenter $zCenter $radius <$startAng $endAng>

$matTag	material integer tag of the previously-defined UniaxialMaterial object used to represent the stress-strain for the area of the fiber
$numBar	number of reinforcing bars along layer
$areaBar	area of individual reinforcing bar
$yCenter $zCenter	y and z-coordinates of center of reinforcing layer (local coordinate system)
$radius	radius of reinforcing layer
$startAng $endAng	starting and ending angle of reinforcing layer, respective *(Optional, Default: a full cir is assumed 0-360)*



tcl procedure

```
proc procName {input variables} {
   … series of commands
}
```

to execute:

```
procName (input variables)
```

tcl proc: define fiber section

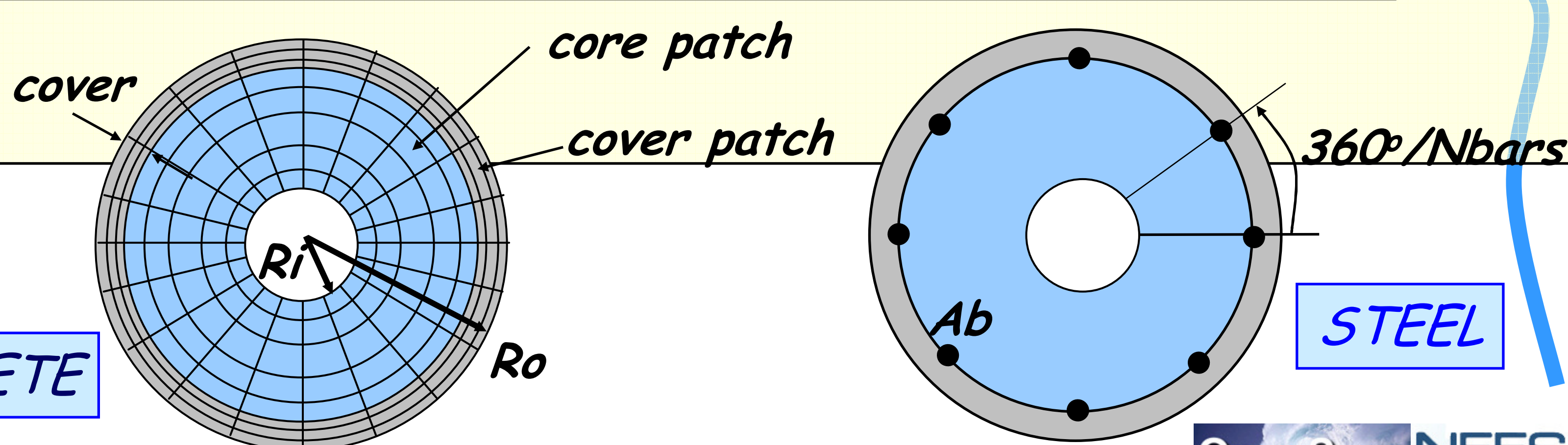
```
proc RCcircSection {id  Ri Ro cover coreID coverID steelID  Nbars Ab  nfCoreR nfCoreT nfCoverR nfCoverT} {
    section fiberSec $id {

        set Rc [expr $Ro-$cover];                                    # Core radius
        patch circ $coreID $nfCoreT $nfCoreR 0 0 $Ri $Rc 0 360;     # Define the core patch
        patch circ $coverID $nfCoverT $nfCoverR 0 0 $Rc $Ro 0 360;        # Define the cover patch

        if {$Nbars<= 0} { return }
        set theta [expr 360.0/$Nbars];                              # angle increment between bars
        layer circ $steelID $Nbars $Ab 0 0 $Rc $theta 360;          # Define the reinforcing layer
    }
}
```

CONCRETE

STEEL



CONCRETE

STEEL

section aggregator

- groups previously-defined UniaxialMaterial objects into a single section force-deformation model

section Aggregator $secTag $matTag1 $string1 $matTag2 $string2 .... <-section $sectionTag>

$secTag	unique section object integer tag
$matTag1, $matTag2 ...	previously-defined UniaxialMaterial objects
$string1, $string2	the force-deformation quantities corresponding to each section object. One of the following strings is used:

P	Axial force-deformation
Mz	Moment-curvature about section local z-axis
Vy	Shear force-deformation along section local y-axis
My	Moment-curvature about section local y-axis
Vz	Shear force-deformation along section local z-axis
T	Torsion Force-Deformation

<-section $sectionTag>	specifies a previously-defined Section object (identified by the argument $sectionTag) to which these UniaxialMaterial objects may be added to recursively define a new Section object

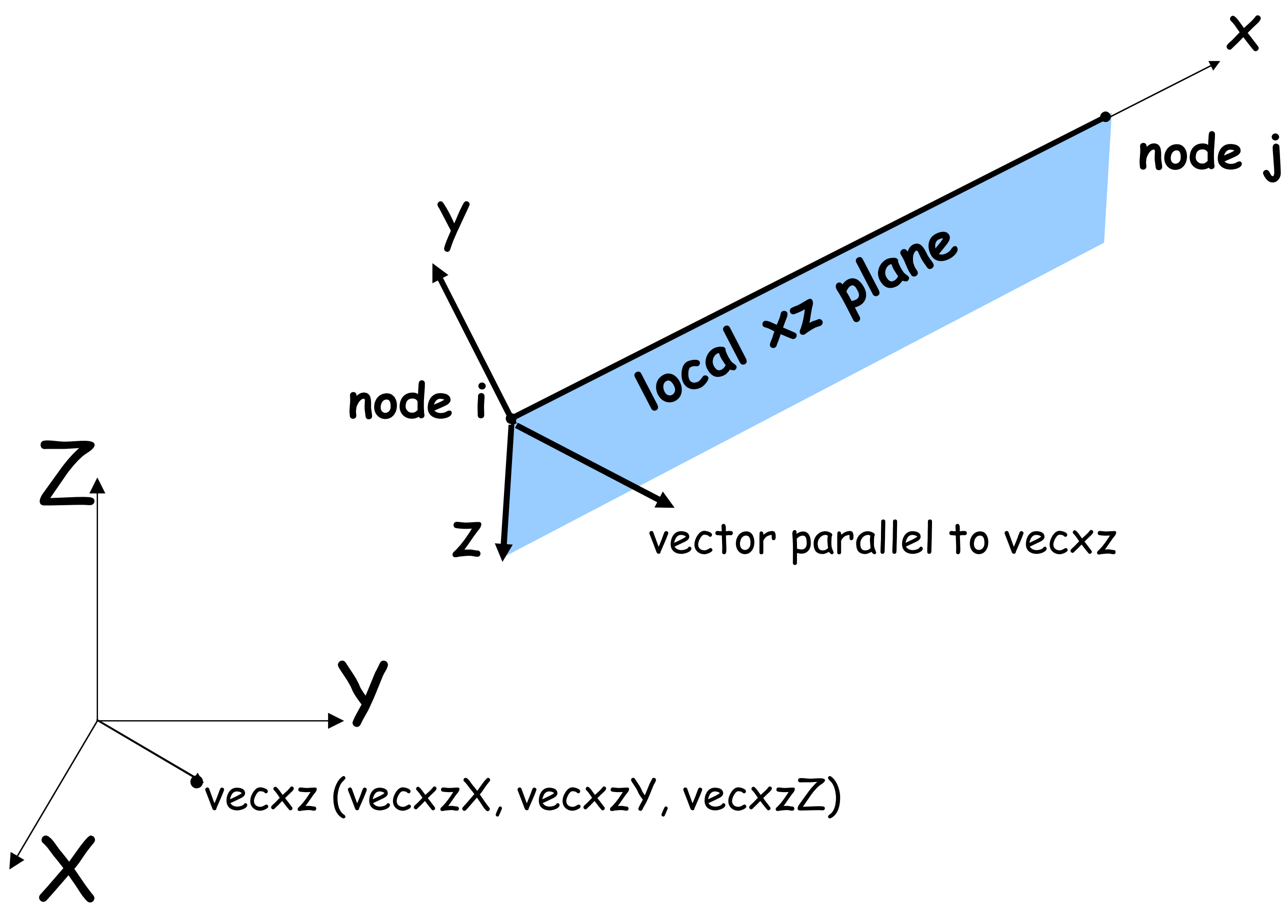
geometric transformation

- performs a linear geometric transformation of beam stiffness and resisting force from the basic system to the global-coordinate system

geomTransf Linear $transfTag $vecxzX $vecxzY $vecxzZ <-jntOffset $dXi $dYi $dZi $dXj $dYj $dZj>

$transfTag	unique identifier for CrdTransf object
$vecxzX $vecxzY $vecxzZ	X, Y, and Z components of vecxz, the vector used to define the local x-z plane of the local-coordinate system. The local y-axis is defined by taking the cross product of the x-axis and the vecxz vector. These components are specified in the global-coordinate system X,Y,Z and define a vector that is in a plane parallel to the x-z plane of the local-coordinate system. These items need to be specified for the three-dimensional problem.
$dXi $dYi $dZi	joint offset values -- absolute offsets specified with respect to the global coordinate system for element-end node i (the number of arguments depends on the dimensions of the current model) (optional)
$dXj $dYj $dZj	joint offset values -- absolute offsets specified with respect to the global coordinate system for element-end node j (the number of arguments depends on the dimensions of the current model) (optional)

local coordinate system



elements

- Truss Element
- Corotational Truss Element
- Elastic Beam Column Element
- NonLinear Beam-Column Elements
  - Nonlinear Beam Column Element
  - Beam With Hinges Element
  - Displacement-Based Beam-Column Element
- Zero-Length Elements
- Quadrilateral Elements
- Brick Elements
- FourNodeQuadUP Element
- BeamColumnJoint Element

Elastic Beam Column Element

- 2D:

element elasticBeamColumn $eleTag $iNode $jNode $A $E $Iz $transfTag

- 3D:

element elasticBeamColumn $eleTag $iNode $jNode $A $E $G $J $Iy $Iz $transfTag

Nonlinear Beam Column Element

```
element nonlinearBeamColumn $eleTag $iNode $jNode
$numIntgrPts $secTag $transfTag <-mass
$massDens> <-iter $maxIters $tol>
```

\$eleTag	unique element object tag
\$iNode \$jNode	end nodes
\$numIntgrPts	number of integration points along the element.
\$secTag	identifier for previously-defined section object
\$transfTag	identifier for previously-defined coordinate-transformation (CrdTransf) object
\$massDens	element mass density (per unit length), from which a lumped-mass matrix is formed *(optional, default=0.0)*
\$maxIters	maximum number of iterations to undertake to satisfy element compatibility *(optional, default=1)*
\$tol	tolerance for satisfaction of element compatibility *(optional, default=*10^{-16}*)*

elements

Tcl procedure

```
set ColumnType "inelastic";
source RCcircSection.tcl;                # proc to define circular fiber section- flexure
RCcircSection $IDcolFlex $riCol $roCol $cover $IDcore $IDcover $IDsteel $NbCol $AbCol $nfCoreR $nfCoreT $nfCoverR $nfCoverT
uniaxialMaterial Elastic $IDcolTors $GJ;     # Define torsion
section Aggregator $IDcolSec      $IDcolTors   T -section  $IDcolFlex;    # attach torsion & flex
geomTransf Linear $IDcolTrans 0 0 1;   # no 2nd-order effects, define element normal
   if {$ColumnType == "elastic"} {
      element elasticBeamColumn 1   1  3    $Acol    $Ec   $G   $J  $IyCol $IzCol $IDcolTrans
      element elasticBeamColumn 2   2  4    $Acol    $Ec   $G   $J  $IyCol $IzCol $IDcolTrans  }
   if {$ColumnType == "inelastic"} {
      # element element type    ID, node I, node J, no. int pts, section ID, transf. ID
      element nonlinearBeamColumn  1    1    3     $np $IDcolSec  $IDcolTrans
      element nonlinearBeamColumn  2    2    4     $np $IDcolSec  $IDcolTrans
   }
   geomTransf Linear    $IDbeamTrans      0 0 1;  #  BEAM transformation, element normal
   element elasticBeamColumn   3  3 4   $Abeam $Ec $G $J $IyBeam $IzBeam $IDbeamTrans
```

COLUMN

BEAM

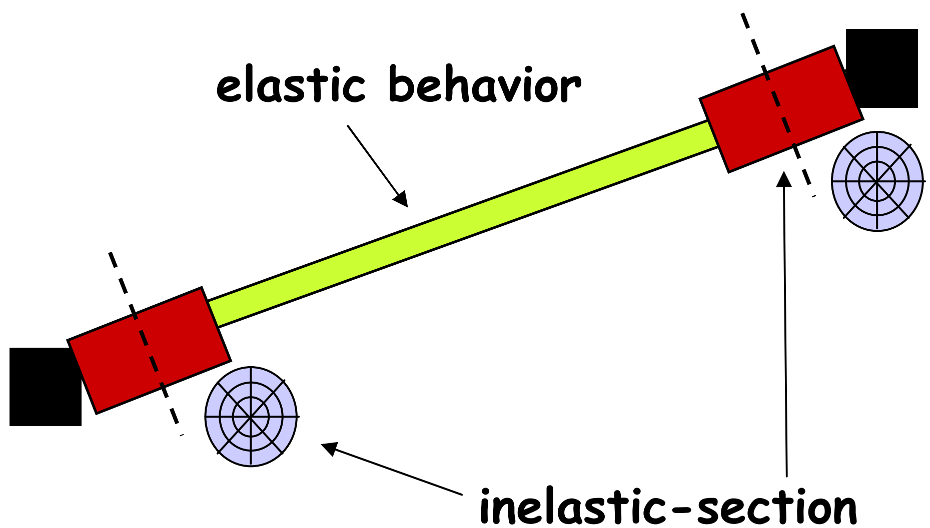
beamWithHinges Element

2D:

element beamWithHinges $eleTag $iNode $jNode $secTagI $HingeLengthI $secTagJ $HingeLengthJ $E $A $Iz $transfTag <-mass $massDens> <-iter $maxIters $tol>

3D:

element beamWithHinges $eleTag $iNode $jNode $secTagI $HingeLengthI $secTagJ $HingeLengthJ $E $A $Iz $Iy $G $J $transfTag <-mass $massDens> <-iter $maxIters $tol>



beamWithHinges Element

$eleTag	unique element object tag
$iNode $jNode	end nodes
$secTagI	identifier for previously-defined section object corresponding to node I
$HingeLengthI	hinge length at node I
$secTagJ	identifier for previously-defined section object corresponding to node J
$HingeLengthJ	hinge length at node J
$E	Young's Modulus
$A	area of element cross-section
$Iz	section moment of inertia about the section local z-axis
$Iy	section moment of inertia about the section local y-axis
$G	Shear Modulus
$J	torsional moment of inertia of cross section
$transfTag	identifier for previously-defined coordinate-transformation (CrdTransf) object
$massDens	element mass density (per unit length), from which a lumped-mass matrix is formed *(optional, default=0.0)*
$maxIters	maximum number of iterations to undertake to satisfy element compatibility *(optional, default=1)*
$tol	tolerance for satisfaction of element compatibility *(optional, default=*10^{-16}*)*

region command

- label a group of nodes and elements.
- This command is also used to assign rayleigh damping parameters to the nodes and elements in this region.
- The region is specified by either elements or nodes, not both. If elements are defined, the region includes these elements and the all connected nodes. If nodes are specified, the region includes these nodes and all elements whose external nodes are prescribed

region $regTag <-ele ($ele1 $ele2 ...)> <-eleRange $startEle $endEle> <-ele all> <-node ($node1 $node2 ...)> <-nodeRange $startNode $endNode> <-node all> <-rayleigh $alphaM $betaK $betaKinit $betaKcomm>

region command (element region)

region $regTag <-ele ($ele1 $ele2 ...)> <-eleRange $startEle $endEle> <-ele all> <-node ($node1 $node2 ...)> <-nodeRange $startNode $endNode> <-node all> <-rayleigh $alphaM $betaK $betaKinit $betaKcomm>

$regTag	unique integer tag
$ele1 $ele2 ...	tags of elements -- selected elements in domain *(optional, default: omitted)*
$startEle $endEle	tag for start and end elements -- range of selected elements in domain *(optional, default: all)*
all	all elements in domain *(optional & default)*
$alphaM $betaK $betaKinit $betaKcomm	Arguments to define Rayleigh damping matrix (optional, default: zero)

region command (node region)

```
region $regTag <-ele ($ele1 $ele2 ...)>  <-eleRange $startEle $endEle>
<-ele all> <-node ($node1 $node2 ...)> <-nodeRange $startNode
$endNode> <-node all> <-rayleigh $alphaM $betaK $betaKinit
$betaKcomm>
```

$regTag	unique integer tag
$node1 $node2 ...	node tags -- select nodes in domain *(optional, default: all)*
$startNode $endNode	tag for start and end nodes -- range of nodes in domain *(optional, default: all)*
all	all nodes in domain *(optional & default)*
$alphaM $betaK $betaKinit $betaKcomm	Arguments to define Rayleigh damping matrix (optional, default: zero

region command (damping)

```
region $regTag <-ele ($ele1 $ele2 ...)> <-eleRange $startEle $endEle> <-ele all> <-node ($node1 $node2 ...)> <-nodeRange $startNode $endNode> <-node all> <-rayleigh $alphaM $betaK $betaKinit $betaKcomm>
```

$$\mathbf{C} = \$alphaM^{*}\mathbf{M} + \$betaK^{*}\mathbf{K} + \$betaKinit^{*}\mathbf{K}_{\mathbf{initial}}\ \$betaKcomm^{*}\mathbf{K}_{\mathbf{committed}}$$

M	mass matrix used to calculate Rayleigh Damping
Kcurrent	stiffness matrix at current state determination used to calculate Rayleigh Damping
Kinit	stiffness matrix at initial state determination used to calculate Rayleigh Damping
KlastCommit	stiffness matrix at last-committed state determination used to calculate Rayleigh Damping