

# OpenSees

## Model-Building Commands

Silvia Mazzoni

Frank McKenna, Gregory L. Fenves  
*University of California, Berkeley*

*nees@berkeley*

Hybrid Simulation Workshop  
April 2007



# Tcl scripting language

- Tcl is a string based scripting language
- enables variables and variable substitution (use variables to define units!!!)
- Expression evaluation
- Array management
- Basic control structures (if , while, for, foreach)
- Procedures
- File manipulation



# Tcl & OpenSees commands

- Command 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 10 0 0
```



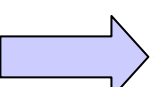
# numerical simulation

1. Set up preliminary Tcl variables to define units and constants
2. Build the numerical model of the physical object (nodes, material, elements, connectivity, etc.)
3. Define the loading configurations
4. Define the output
5. Prescribe the analysis procedure
6. Analyze



# Using Variables in Tcl -- UNITS

- set in 1.; # define basic units
- set sec 1.; # define basic units
- set kip 1.; # define basic units
- set ft [expr 12.\*\$in]; # define engineering units
- set ksi [expr \$kip/pow(\$in,2)];
- set psi [expr \$ksi/1000.];
- set in2 [expr \$in\*\$in]; # inch^2
- set in4 [expr \$in\*\$in\*\$in\*\$in]; # inch^4
- set PI [expr 2\*asin(1.0)]; # define constants
- set g [expr 32.2\*\$ft/pow(\$sec,2)]; # grav. acceleration

- 
- set Lcol [expr 36\*\$ft]; # column length
  - set Dcol [expr 6.5\*\$ft]; # circular-column Diameter
  - node 1 0 0
  - node 2 0 \$Lcol
  - set Weight [expr 1000\*\$kip]; # weight
  - set Mass [expr \$Weight/\$g]; # mass
  - mass 2 \$Mass 0 0; # assign mass to node



# ModelBuilding 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



# http://opensees.berkeley.edu



manual version 2.0 - Netscape

File Edit View Go Bookmarks Tools Window Help

http://peer.berkeley.edu/~silvia/OpenSees/manual/html/

OpenSees Open System for Earthquake Engineering Simulation Pacific Earthquake Engineering Research Center OpenSees

Contents Index

### Contents

- opening\_page
- Introduction
- OpenSees
- 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

- Analysis Objects
- Recorder Objects

### 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](#)

Start MyPresentation manual version 2.0... fig tree - Inbox for sil... SilviaMazzoni\_ModelB... SilviaMazzoni\_GetRea... 2:36 AM

# or, OpenSeesManual.chm

Microsoft PowerPoint - [SilviaMazzoni - ModelBuilding.ppt]

manual

File Edit

Hide Back Print Options

for help

Contents Index Search

opening page

Introduction

OpenSees

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

Analysis Objects

Recorder Objects

Miscellaneous Commands

How To....

References

## 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](#)

Slide 19 of 29 peer English (U.S.)

Start MyPresentation manual version 2.0 - ... the report - Inbox for ... manual Microsoft PowerPoint ... 3:25 AM



# 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)>
```

<b>\$nodeTag</b>	integer tag identifying node
<b>\$coords</b>	nodal coordinates ( <u>ndm</u> arguments)
<b>\$MassValues</b>	nodal mass corresponding to each DOF ( <u>ndf</u> 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)>
```

<b>\$nodeTag</b>	integer tag identifying node
<b>\$coords</b>	nodal coordinates ( <u>ndm</u> arguments)
<b>\$MassValues</b>	nodal mass corresponding to each DOF ( <u>ndf</u> arguments) (optional)

```
fix $nodeTag (ndf $ConstrValues)
```

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

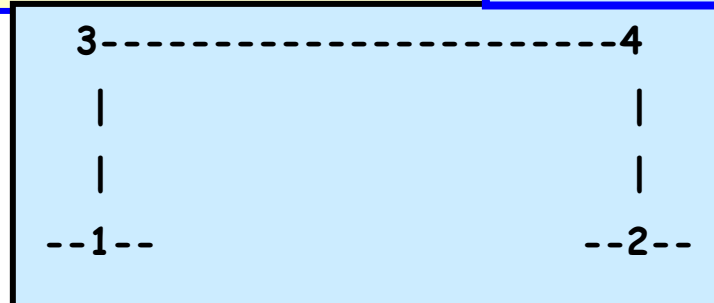


# nodalmesh.tcl

```
1. # Define nodes; # frame is in X-Y plane
2. node 1 0.0 0.0 0.0
3. node 2 $Lbeam 0.0 0.0
4. node 3 0.0 $Lcol 0.0 -mass $Mnode 0.0 0.0 0.0 0.0 0.0
5. node 4 $Lbeam $Lcol 0.0 -mass $Mnode 0.0 0.0 0.0 0.0 0.0
6. # Boundary conditions; # node DX DY DZ RX RY RZ ! 1: fixed, 0: released
7. fix 1 1 1 1 1 1 1;
8. fix 2 1 1 1 1 1 1
9. fix 3 0 0 1 1 1 0
10. fix 4 0 0 1 1 1 0
11. #
12. #
13. #
14. #
```

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\*



# tcl if statement

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



# materials.tcl

concrete

```
uniaxialMaterial Concrete01 $matTag $fpc $epsC0 $fpcu $epsU
```

1. 

```
set ConcreteMaterialType "inelastic" # options: "elastic","inelastic"
```
2. 

```
if {$ConcreteMaterialType == "elastic"} {
```
3. 

```
    uniaxialMaterial Elastic $IDcore $Ec
```
4. 

```
    uniaxialMaterial Elastic $IDcover $Ec
```
5. 

```
}
```
6. 

```
if {$ConcreteMaterialType == "inelastic"} {
```
7. 

```
    # uniaxial Kent-Scott-Park concrete model w/ linear unload/reload, no T strength (-ve comp.)
```
8. 

```
    uniaxialMaterial Concrete01 $IDcore $fpc1C $eps1C $fpc2C $eps2C; # Core
```
9. 

```
    uniaxialMaterial Concrete01 $IDcover $fpc1U $eps1U $fpc2U $eps2U;  
    # Cover
```
10. 

```
}
```



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 <i>first</i> point of the envelope in the <i>positive</i> direction
\$s2p \$e2p	stress and strain (or force & deformation) at <i>second</i> point of the envelope in the <i>positive</i> direction
\$s3p \$e3p	stress and strain (or force & deformation) at <i>third</i> point of the envelope in the <i>positive</i> direction (optional)
\$s1n \$e1n	stress and strain (or force & deformation) at <i>first</i> point of the envelope in the <i>negative</i> direction*
\$s2n \$e2n	stress and strain (or force & deformation) at <i>second</i> point of the envelope in the <i>negative</i> direction*
\$s3n \$e3n	stress and strain (or force & deformation) at <i>third</i> point of the envelope in the <i>negative</i> 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

# materials.tcl

reinforcing steel

```
1. set SteelMaterialType "hysteretic";
2. if {$SteelMaterialType == "elastic"} {
3.   uniaxialMaterial Elastic $IDsteel $Es
4. }
5. if {$SteelMaterialType == "bilinear"} {
6.   uniaxialMaterial Steel01 $IDsteel $Fy $Es $Bs
7. }
8. if {$SteelMaterialType == "hysteretic"} {
9.   uniaxialMaterial Hysteretic $IDsteel $Fy $epsY
   $Fy1 $epsY1 $Fu $epsU -$Fy -$epsY -$Fy1 -
   $epsY1 -$Fu -$epsU $pinchX $pinchY $damage1
   $damage2 $betaMUs steel
10. }
```



# tcl procedure

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

to execute:

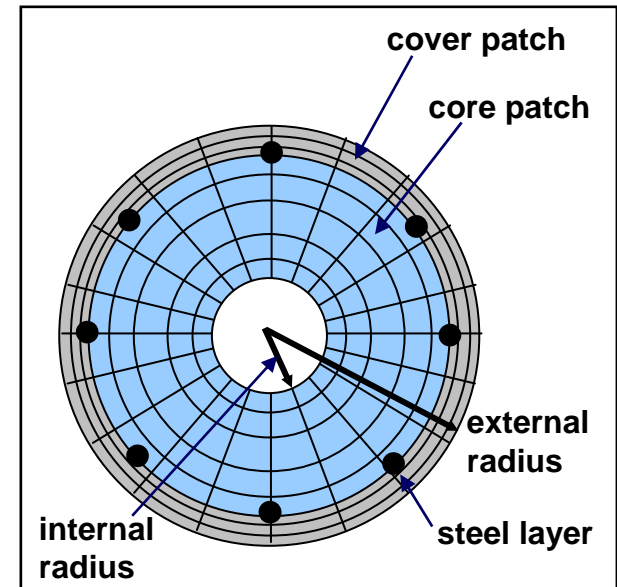
```
procName (input variables)
```



# fiber section command

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

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

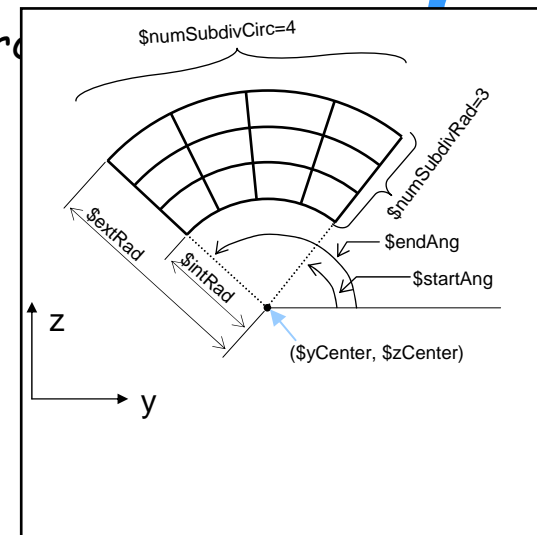


- \$yLoc** y coordinate of the fiber in the section (local coordinate system)
- \$zLoc** z coordinate of the fiber in the section (local coordinate system)
- \$A** area of fiber
- \$matTag** material tag of the pre-defined [UniaxialMaterial](#) object used to represent the stress-strain for the area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

# section command (cont.)

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

<b>\$matTag</b>	material integer tag of the previously-defined <u>UniaxialMaterial</u> object used to represent the stress-strain for the area of the fiber
<b>\$numSubdivCirc</b>	number of subdivisions (fibers) in the circumferential direction.
<b>\$numSubdivRad</b>	number of subdivisions (fibers) in the radial direction.
<b>\$yCenter</b> <b>\$zCenter</b>	y & z-coordinates of the center of the circle
<b>\$intRad</b>	internal radius
<b>\$extRad</b>	external radius
<b>\$startAng</b>	starting angle (optional. default=0.0)
<b>\$endAng</b>	ending angle (optional. default=360.0)



# 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

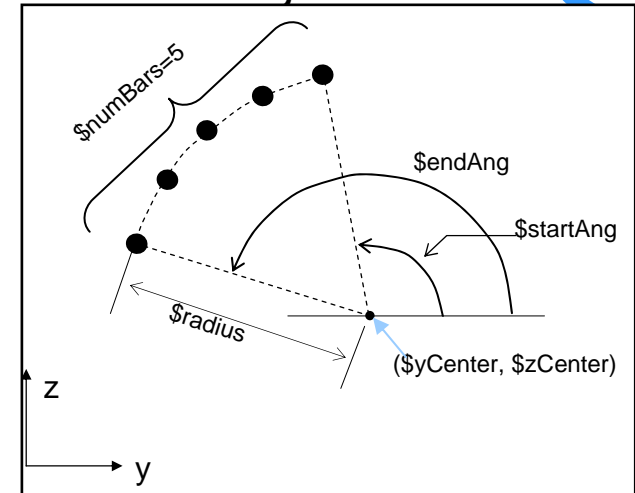
**\$numBar** number of reinforcing bars along layer

**\$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, respectively. *(Optional, Default: a full circle is assumed 0-360)*



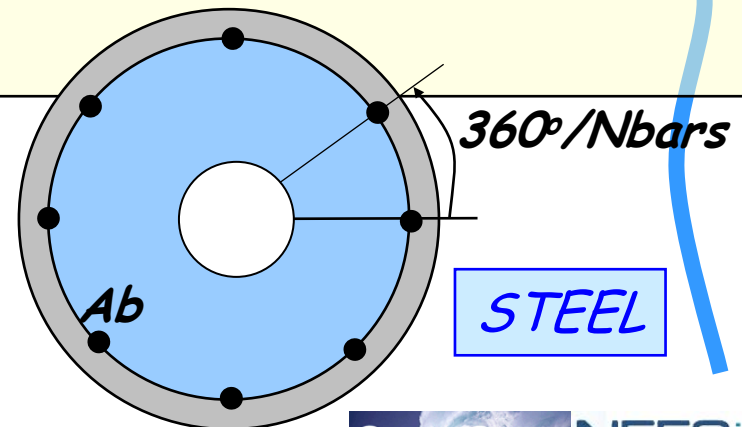
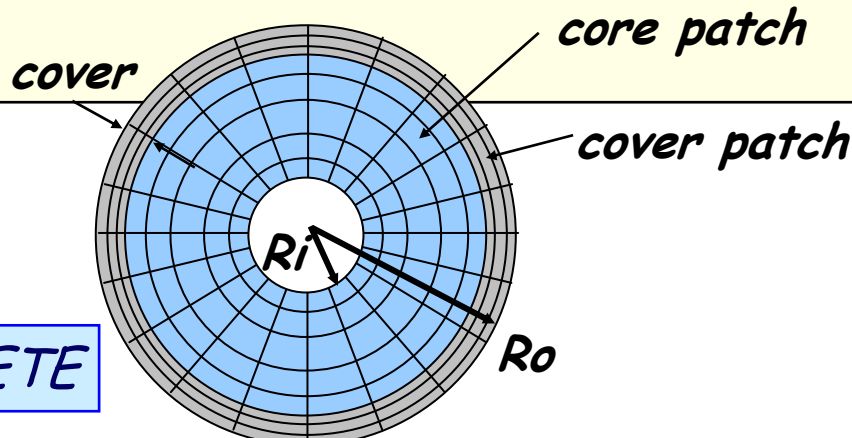
# tcl proc: define fiber section

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

```
    set Rc [expr $Ro-$cover]; CONCRETE # 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]; STEEL # angle increment between bars
    layer circ $steelID $Nbars $Ab 0 0 $Rc $theta 360; # Define the reinforcing
```

```
  layer
}
```



# section aggregator

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

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

<b>\$secTag</b>	unique section object integer tag
<b>\$matTag1,</b> <b>\$matTag2 ...</b>	previously-defined <u>UniaxialMaterial</u> objects
<b>\$string1, \$string2</b> ....	the force-deformation quantities corresponding to each section object. One of the following strings is used:
<b>P</b>	Axial force-deformation
<b>Mz</b>	Moment-curvature about section local z-axis
<b>Vy</b>	Shear force-deformation along section local y-axis
<b>My</b>	Moment-curvature about section local y-axis
<b>Vz</b>	Shear force-deformation along section local z-axis
<b>T</b>	Torsion Force-Deformation
<b>&lt;-section \$sectionTag&gt;</b>	specifies a previously-defined <u>Section</u> object (identified by the argument \$sectionTag) to which these <u>UniaxialMaterial</u> objects may be added to recursively define a new <u>Section</u> object



# geometric transformation

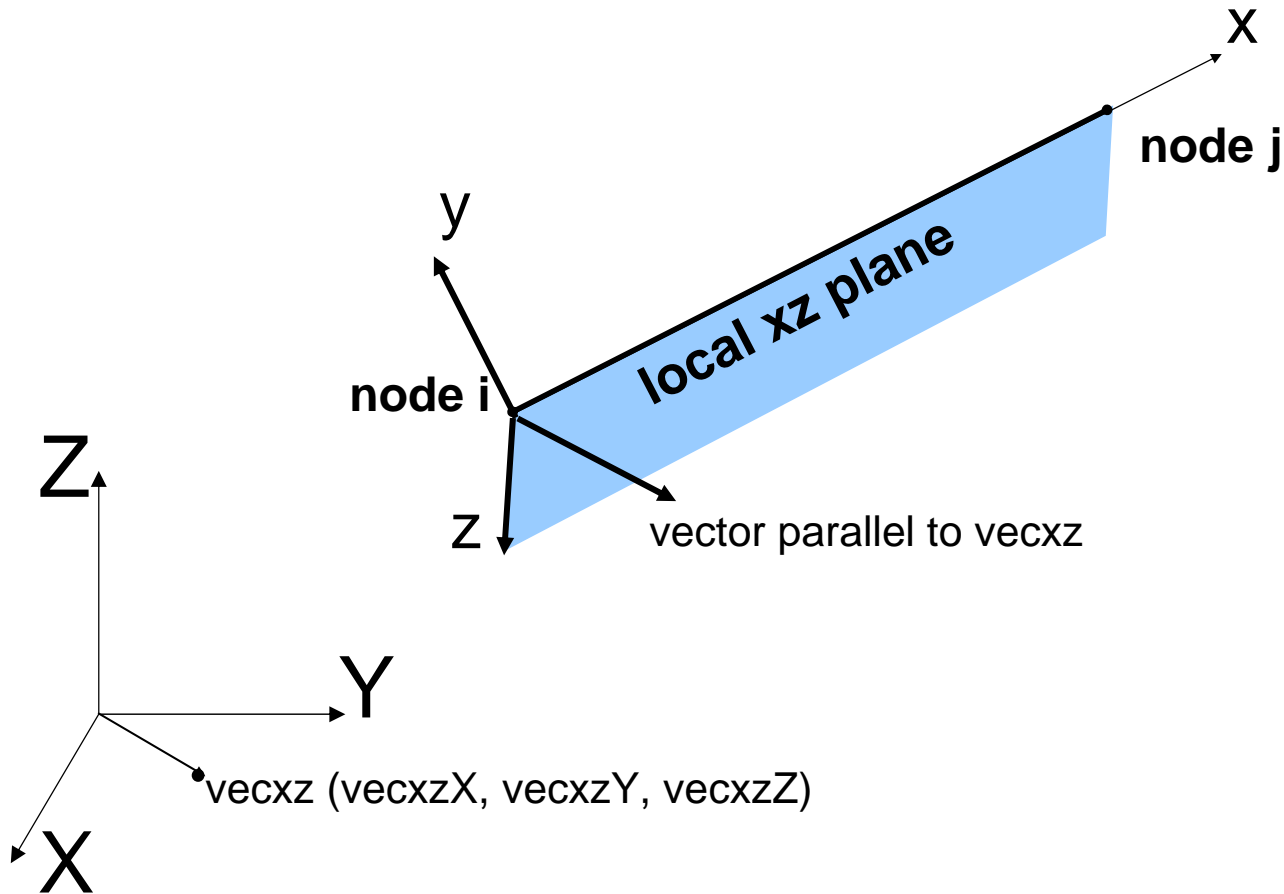
- performs a linear geometric transformation of beam stiffness and resisting force from the basic system to the global-coordinate system

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

<b>\$transfTag</b>	unique identifier for CrdTransf object
<b>\$vecxzX</b> <b>\$vecxzY</b> <b>\$vecxzZ</b>	X, Y, and Z components of $\text{vecxz}$ , 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 $\text{vecxz}$ 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.
<b>\$dXi \$dYi</b> <b>\$dZi</b>	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)
<b>\$dXj \$dYj</b> <b>\$dZj</b>	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>
```

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



# elements.tcl



```
1. set ColumnType "inelastic";
2. source RCcircSection.tcl; # proc to define circular fiber section– flexure
3. RCcircSection $IDcolFlex $riCol $roCol $cover $IDcore $IDcover $IDsteel $NbCol $AbCol $nfCoreR $nfCoreT $nfCoverR $nfCoverT
4. uniaxialMaterial Elastic $IDcolTors $GJ; # Define torsion
5. section Aggregator $IDcolSec $IDcolTors T -section $IDcolFlex; # attach torsion & flex
6. geomTransf Linear $IDcolTrans 0 0 1; # no 2nd-order effects, define element normal

7. if {$ColumnType == "elastic"} {
8.     element elasticBeamColumn 1 1 3 $Acol $Ec $G $J $IyCol $IzCol $IDcolTrans
9.     element elasticBeamColumn 2 2 4 $Acol $Ec $G $J $IyCol $IzCol $IDcolTrans }
10. if {$ColumnType == "inelastic"} {
11.     # element element type ID, node I, node J, no. int pts, section ID, transf. ID
12.     element nonlinearBeamColumn 1 1 3 $np $IDcolSec $IDcolTrans
13.     element nonlinearBeamColumn 2 2 4 $np $IDcolSec $IDcolTrans }

14. geomTransf Linear $IDbeamTrans 0 0 1; # BEAM transformation, define element normal
15. element elasticBeamColumn 3 3 4 $Abeam $Ec $G $J $IyBeam $IzBeam $IDbeamTrans
```

COLUMN

BEAM

# Recorder Objects

- Node Recorder
- EnvelopeNode Recorder
- MaxNodeDisp Recorder
- Drift Recorder
- Element Recorder
- EnvelopeElement Recorder
- Display Recorder
- Plot Recorder
- playback Command



```
recorder Node <-file $fileName> <-time> <-node ($node1
$node2 ...)> <-nodeRange $startNode $endNode> <-region
$RegionTag> <-node all> -dof ($dof1 $dof2 ...) $respType
```

**\$fileName** file where results are stored. Each line of the file contains the result for a committed state of the domain (*optional, default: screen output*)

**-time** this argument will place the pseudo time of the as the first entry in the line. (*optional, default: omitted*)

**\$node1**  
**\$node2 ...** tags nodes where response is being recorded -- select nodes in domain (*optional, default: all*)

**\$startNode**  
**\$endNode** tag for start and end nodes where response is being recorded -- range of nodes in domain (*optional, default: all*)

**\$RegionTag** tag for previously-defined selection of nodes defined using the Region command. (*optional*)

**all** where response is being recorded -- all nodes in domain (*optional & default*)

**\$dof1 \$dof2**  
**...** degrees of freedom of response being recorded. Valid range is from 1 through **ndf**, the number of nodal degrees-of-freedom.

**\$respType** defines response type to be recorded. The following response types are available:

<b>disp</b>	displacement
<b>vel</b>	velocity
<b>accel</b>	acceleration
<b>incrDisp</b>	incremental displacement
<b>eigen</b>	eigenvector
<b>reaction</b>	Nodal Reaction



```
recorder EnvelopeNode <-file $fileName> <-time> <-node  
($node1 $node2 ...)> <-nodeRange $startNode $endNode> <-  
region $RegionTag> <-node all> -dof ($dof1 $dof2 ...)  
$respType
```

records the envelope of displacement, velocity, acceleration and incremental displacement at the nodes (translational & rotational). The envelope consists of the following: minimum, maximum and maximum absolute value of specified response type

- \$fileName** file where results are stored. Each line of the file contains the result for a committed state of the domain (*optional, default: screen output*)
- time** this argument will place the pseudo time of the as the first entry in the line. (*optional, default: omitted*)
- \$node1** tags nodes where response is being recorded -- select nodes in domain
- \$node2 ...** (*optional, default: all*)
- \$startNode** tag for start and end nodes where response is being recorded -- range of
- \$endNode** nodes in domain (*optional, default: all*)
- \$RegionTag** tag for previously-defined selection of nodes defined using the Region command. (*optional*)
- all** where response is being recorded -- all nodes in domain (*optional & default*)
- \$dof1** degrees of freedom of response being recorded.
- \$dof2 ...** Valid range is from 1 through [ndf](#), the number of nodal degrees-of-freedom.

..... **same arguments as node recorder**



```
recorder MaxNodeDisp $dof $node1 $node2
```

```
...
```

records the values of the maximum absolute values of the displacement in the prescribed direction of a prescribed set of nodes

**\$dof** displacement degree-of-freedom direction.  
Valid range is from 1 through ndf, the number of nodal degrees-of-freedom.

**\$node1** nodes where maximum displacement is being  
**\$node2 ...** recorded



```
recorder Element <-file $fileName> <-time> <-ele ($ele1 $ele2
...)> <-eleRange $startEle $endEle> <-region $regTag> <-ele
all> ($arg1 $arg2 ...)
```

- \$fileName** file where results are stored. Each line of the file contains the result for a committed state of the domain (*optional, default: screen output*)
- time** this argument will place the pseudo time of the as the first entry in the line. (*optional, default: omitted*)
- \$ele1 \$ele2 ...** tags of elements whose response is being recorded -- selected elements in domain (*optional, default: omitted*)
- \$startEle \$endEle** tag for start and end elements whose response is being recorded -- range of selected elements in domain (*optional, default: all*)
- \$regTag** previously-defined tag of region of elements whose response is being recorded -- region of elements in domain (*optional*)
- all** elements whose response is being recorded -- all elements in domain (*optional & default*)
- \$arg1 \$arg2 ...** arguments which are passed to the setResponse() element method



# element recorder (output arguments)

## All:

**globalForce** - element resisting force in global coordinates (does not include inertial forces)

recorder Element -file ele1global.out -time -ele 1 globalForce

**localForce** - element resisting force in local coordinates (does not include inertial forces)

recorder Element -file ele1local.out -time -ele 1 localForce

## Section:

**section \$secNum** - request response quantities from a specific section along the element length

**\$secNum** refers to the integration point whose data is to be output

**force** - section forces

*example:* recorder Element -file ele1sec1Force.out -time -ele 1 section 1 force

**deformation** - section deformations

*example:* recorder Element -file ele1sec1Force.out -time -ele 1 section 1 deformation

**stiffness** - section stiffness

*example:* recorder Element -file ele1sec1Force.out -time -ele 1 section 1 stiffness

**stressStrain** - record stress-strain response.

*example:* recorder Element -file ele1sec1Force.out -time -ele 1 section 1 fiber \$y \$z stressStrain

**\$y** local y coordinate of fiber to be monitored\*  
**\$z** local z coordinate of fiber to be monitored\*



# output.tcl

# Record nodal displacements -NODAL DISPLACEMENTS

# ALL displacements at node 1

```
recorder Node -file Dnode1.out -time -node 1 -dof 1 2 3 disp;
```

# Record vertical-y displacement of ALL nodes

```
recorder Node -file DNodeALL.out -time -node all -dof 2 disp;
```

# Record REACTION FORCES - (=forces in element 1)

```
recorder Element -file Fel1.out -time -ele 1 localForce
```



# Loads - pattern command

```
pattern Plain $patternTag (TimeSeriesType arguments) {  
  load (load-command arguments)  
  sp (sp-command arguments)  
  eleLoad (eleLoad-command arguments)  
}
```

- |                                 |  |
|---------------------------------|--|
| <b>\$patternTag</b>             | unique pattern object tag  |
| <b>TimeSeriesType arguments</b> | list which is parsed to construct the <u>TimeSeries</u> object associated with the LoadPattern object. |
| <b>load ...</b>                 | list of commands to construct nodal loads -- the <u>NodalLoad</u> object                               |
| <b>sp ...</b>                   | list of commands to construct single-point constraints -- the <u>SP_Constraint</u> object              |
| <b>eleLoad ...</b>              | list of commands to construct element loads -- the <u>eleLoad</u> object                               |



# pattern command (cont.)

## load \$nodeTag (ndf \$LoadValues)

- \$nodeTag** node on which loads act
- \$LoadValues** load values that are to be applied to the node.  
Valid range is from 1 through **ndf**, the number of nodal degrees-of-freedom.

## sp \$nodeTag \$DOFtag \$DOFvalue

- \$nodeTag** node on which the single-point constraint acts
- \$DOFtag** degree-of-freedom at the node being constrained.  
Valid range is from 1 through **ndf**, the number of nodal degrees-of-freedom.
- \$DOFvalue** reference value of the constraint to be applied to the DOF at the node.

```
pattern Plain 1 Linear {
    #                Fx  Fy  Fz  Mx  My  Mz
    load 3           0.0 -$Pdl 0.0 0.0 0.0  -$Mdl
    load 4           0.0 -$Pdl 0.0 0.0 0.0  +$Mdl
}
```



# Questions, or statements!

The OpenSees Community Forum:

<http://opensees.berkeley.edu/community/index.php>

which can be accessed from:

<http://opensees.berkeley.edu>



thank you!!!

