Giraffe User's Manual v. 2.0.64

GIRAFFE

Generic Interface Readily Accessible for Finite Elements

Developed at University of São Paulo, Brazil

January 2020

Prof. Alfredo Gay Neto

Table of Contents

Introduction

Giraffe is the acronym of "Generic Interface Readily Accessible for Finite Elements". It is a platform coded using C++ language, with the objective of generating a base-interface to be used by researchers, to implement their own finite element formulations. Giraffe does not have the mission of being a completely generic platform, which would be too ambitious. Structural problems, however, which may include translational and rotational degrees of freedom, such as possible multiphysics applications, can be sketched in such a way that permits creating a platform to embrace new elements, new contact formulations, new constraint equations, among other features. With that aim, "Giraffe Project" was started on 2014 by Prof. Alfredo Gay Neto, at University of São Paulo, Brazil.

Giraffe has started as a generalization of a previous-developed finite element code, named "FemCable", which had the objective of simulating offshore structures: risers for oil exploitation. It had implementations of geometric nonlinear beam elements and classical node to surface contact formulation. Since a natural expansion required including new contact models, new structural elements and other resources, Giraffe was designed to have all the models included in "FemCable". Furthermore, it was thought to embrace easy inclusion of new resources, using object orientation programming. Giraffe is under continuous development by Prof. Alfredo Gay Neto and co-workers.

On 2018 Giraffe was completely re-structured to encompass a new broad of resources. The new code structure provides new possibilities for modeling, with a higher versatility. This includes the possibility of defining a sequence of solutions, possibly mixing static and dynamic methods, according to convenience. Furthermore, "BoolTable" keyword has replaced "Steps" keyword, no longer available. "BoolTable" provides an easier way to define in which solution step each resource will be included or not included in simulation. With that, one may straightforwardly switch on/off boundary conditions, loads, joints, contacts, etc. This leads to the possibility of creating scenarios where load sequence is an issue. Furthermore, it provides numerical strategies to achieve solution of challenging nonlinear problems. Also, post-processing possibilities have changed, with a more organized set of post files, for post-processing using Paraview™ environment. With respect to the user input file, there is a slight change between Giraffe 1.0 and Giraffe 2.0 syntax, since some commands have changed, there are new ones and others were discontinued. Users are invited to have a look at new tutorials, in order to get used to new resources.

This user's manual has some brief explanations on how to use Giraffe to simulate structural models using the available elements, contact algorithms and special constraints. The focus is to explain the syntax of the Giraffe input file. Examples may be found in Giraffe tutorials documentation.

Suggestions are always welcome and can be emailed to Prof. Alfredo Gay Neto: alfredo.gay@usp.br.

Enjoy!

Alfredo Gay Neto

São Paulo, Brazil, 2018.

General information

Installing Giraffe

Currently Giraffe is available only for Windows™ 64 bit. To install Giraffe, please follow the instructions depicted next:

1. Copy to your computer and install the patches "vcredist_x64" e "ww_icl_redist_intel64_2016.4.246", available in the folder "/Giraffe 2.0/Giraffe Install/".
2. Copy to your computer the folder with the desired Giraffe version, located in "/Giraffe 2.0/Giraffe Software/". For example, the folder "Giraffe 2.0.0 (beta)".
3. In your computer, inside the copied Giraffe version folder, copy all contents available in the folder "/Giraffe 2.0/Giraffe Install/GiraffeDLL/". Alternatively, one may copy such contents to another folder in the computer. In this case, it will be necessary to edit the environment variable "Path". In Windows this may be done by going on: Control Panel\All Control Panel Items\System. Enter the option "Advanced System Settings"->Environment Variables. Locate the Environment variable "Path" and edit it, including the desired directory where dll files are located. With that, the system automatically seeks for this location when executing Giraffe.
4. If desired, in your computer, inside the copied Giraffe version folder, copy also input file examples, available in the folder "Giraffe Input".
5. Execute Giraffe in your computer by double clicking the executable file.

Note: Giraffe usage is restricted. No user is permitted to supply third-part people with copies of Giraffe with no previous authorization of the developer.

Running Giraffe

To perform a simulation, just open Giraffe executable file. The instruction: "Enter the name of input file" will be given in the screen. Type the desired file name. Then, wait until the simulation finishes. In Figure 1 the file name typed is "beam01". Do not include the extension of the input file in this typing procedure. If you type "beam01.inp" Giraffe will not find the file.

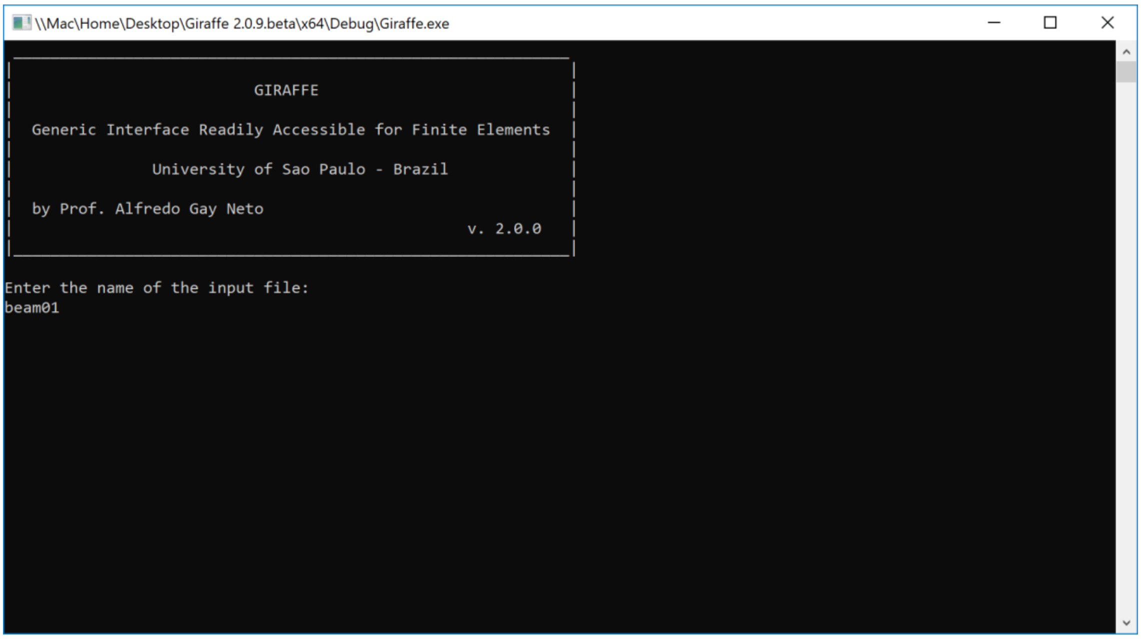


Figure 1 – Giraffe command window

Input file

Prior to perform a simulation, Giraffe creates a model database with all needed setup. This is done by reading a user input file containing all the commands to construct necessary data for the model, such as nodes, elements, loads, constraints, options for solution, etc. After reading and verifying if input data is consistent, the model is solved.

Giraffe reads a single input text file[1]. It must be located inside a folder with the same name of the input file. It is mandatory the usage of the file extension "*.inp" for the input file. Files with different extensions or with no extensions will result in error messages when Giraffe tries to read them.

Figure 2 shows Giraffe directory and some input files folders. For example, the folder "beam01". The input file named "beam01.inp" is located inside the "beam01" folder (Figure 3).

The folder with the input file can be located in two possible directories:

- The directory of "Giraffe.exe" executable file or
- The public "/Documents/Giraffe/" folder.

When trying to read an input file, Giraffe first seeks for it in the same directory of the "Giraffe.exe" executable file. If not succeed, it tries to read a file located in public "Documents/Giraffe" folder. If not succeed in this second try, an error message is prompted to the user.

[1] Some exceptions are treated locally in this manual, when extra inputs are to be provided by the user.

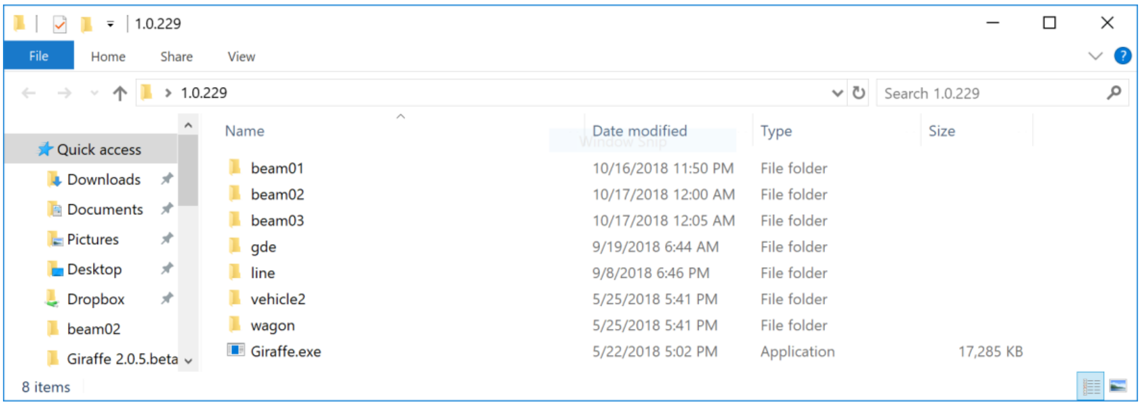


Figure 2 – Giraffe executable file directory example

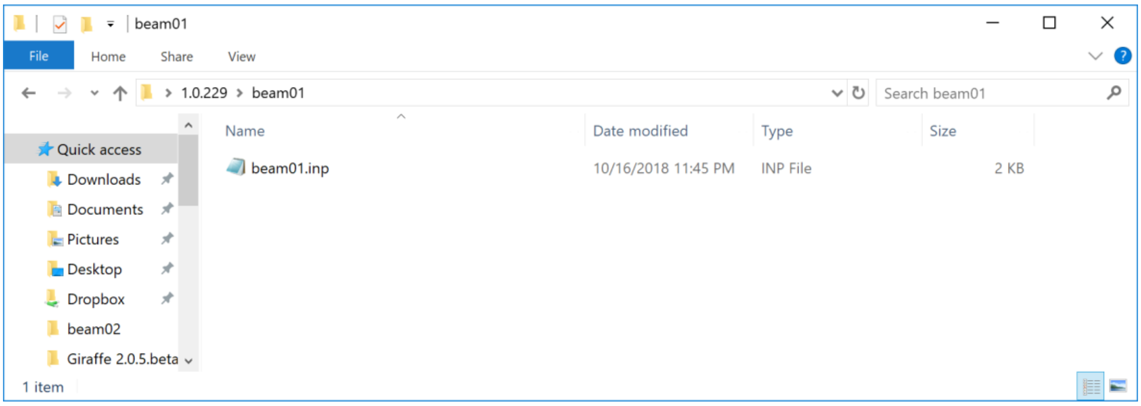


Figure 3 – Input file folder example

This user's manual presents and gives examples of each keyword to be used as a part of Giraffe input file. Giraffe is prepared to read the input file keywords independently of any pre-defined sequence. For example, one may first define the finite element nodes and, after, define elements. Alternatively, one may first define the elements and, after, the nodes. Therefore, the sequence of commands here provided is not mandatory for compounding the input file structure.

To interrupt Giraffe reading process of an input file, the user optionally is allowed to introduce the keyword EOF, indicating "end of file". This causes Giraffe to read the input file only up to that position. If this keyword is not included in input file, Giraffe will read the whole input file contents.

Explanatory user-comments can be included in some parts of the input file. Comments syntax for Giraffe input file is analogous to C and C++ language, as follows. Single line comments can be included by starting a new line with "//…". Comments can also span multiple lines, for this purpose one may use "/*…*/". An example of a commented input file is shown below:

```
//Comment 0
Nodes  5
//Comment 1
Node   1      0      2.5    0
Node   2      0.1    2.5    0
Node   3      0.2    2.5    0
//Comment 2
Node   4      0.3    2.5    0
```

```
Node    5       0.4     2.5     0

//Comment 3
CoordinateSystems       2
CS      1       E1      0       1       0       E3      1       0       0
CS      2       E1      1       0       0       E3      0       1       0

Materials       1
Hooke   1       E       2e9     Nu      0.30    Rho     8000

//Comment 4
Sections        2
SuperEllipse    1       A       0.1     B       0.06    N       2       AMeshFDM        100
SuperEllipse    2       A       0.06    B       0.1     N       2       AMeshFDM        100

/*
Large comment
with multiple lines
*/

Elements        2
//Comment 5
Beam_1          1       Mat     1       Sec     1       CS      1       Nodes   1 2 3
//Comment 6
Beam_1          2       Mat     1       Sec     1       CS      1       Nodes   3 4 5
```

Note that it is possible to make comments between first level keywords (e.g.: Nodes, Elements, Sections). It is also possible to make comments between second-level keywords (e.g.: Node, CS, Hooke, Beam_1).

Note: the user cannot introduce comments between lower-level keywords, for example:

```
Node  1  X  /*not allowed*/  0.1  Y  2.5  Z  0  CONSTR  0
```

Next, an example of Giraffe input file is shown. The reader finds on it a basic structure to establish a simple finite element model of a cantilever beam initially aligned in global Z direction (tutorial01).

```
/* Example of an input file for Giraffe */
//Creation of nodes
Nodes   11
//Number        X       Y       Z
Node    1       0       0       0.0
Node    2       0       0       0.1
Node    3       0       0       0.2
Node    4       0       0       0.3
Node    5       0       0       0.4
Node    6       0       0       0.5
Node    7       0       0       0.6
Node    8       0       0       0.7
```

```
Node    9       0       0       0.8
Node    10      0       0       0.9
Node    11      0       0       1.0
//Creation of node sets
NodeSets        2
NodeSet         1       Nodes   1       List    1
NodeSet         2       Nodes   1       List    11
//Creation of elements
Elements        5
Beam_1          1       Mat     1       Sec     1       CS      1       Nodes   1 2 3
Beam_1          2       Mat     1       Sec     1       CS      1       Nodes   3 4 5
Beam_1          3       Mat     1       Sec     1       CS      1       Nodes   5 6 7
Beam_1          4       Mat     1       Sec     1       CS      1       Nodes   7 8 9
Beam_1          5       Mat     1       Sec     1       CS      1       Nodes   9 10 11
//Creation of materials
Materials       1
Hooke   1       E       1e7     Nu      0.3     Rho     2000
//Creation of sections
Sections        1
Rectangle       1       B       0.1     H       0.1
//Creation of coordinate systems
CoordinateSystems       1
CS      1       E1      1       0       0       E3      0       0       1
//Creation of the solution steps
SolutionSteps   1
Static  1
EndTime         1
TimeStep        0.1
MaxTimeStep     0.1
MinTimeStep     0.01
MaxIt   20
MinIt   3
ConvIncrease    4
IncFactor       1.0
Sample 2
//Creation of loads
Loads   1
NodalLoad       1       NodeSet         2       CS      1       NTimes 2
//Time  FX      FY      FZ      MX      MY      MZ
0       0       0       0       0       0       0
1       1000    0       0       0       0       0
//Creation of constraints
Constraints     1
NodalConstraint         1       NodeSet         1
        UX              BoolTable       1
        UY              BoolTable       1
        UZ              BoolTable       1
        ROTX    BoolTable       1
        ROTY    BoolTable       1
```

```
	ROTZ	BoolTable	1
//Creation of solver options
SolverOptions
Processors	4	LinSys	Direct
//Creation of monitors
Monitor		Sample 1
MonitorNodes	1	11
//Creation of post files
PostFiles
MagFactor	1.0
WriteMesh	1
WriteRenderMesh		1
WriteRigidContactSurfaces	0
WriteFlexibleContactSurfaces	0
WriteForces	0
WriteConstraints	0
WriteSpecialConstraints		0
WriteContactForces	0
WriteRenderRigidBodies		0
WriteRenderParticles	0
```

Output files

During solution process Giraffe automatically saves all requested output files. They are saved inside the folder where the input file was read.

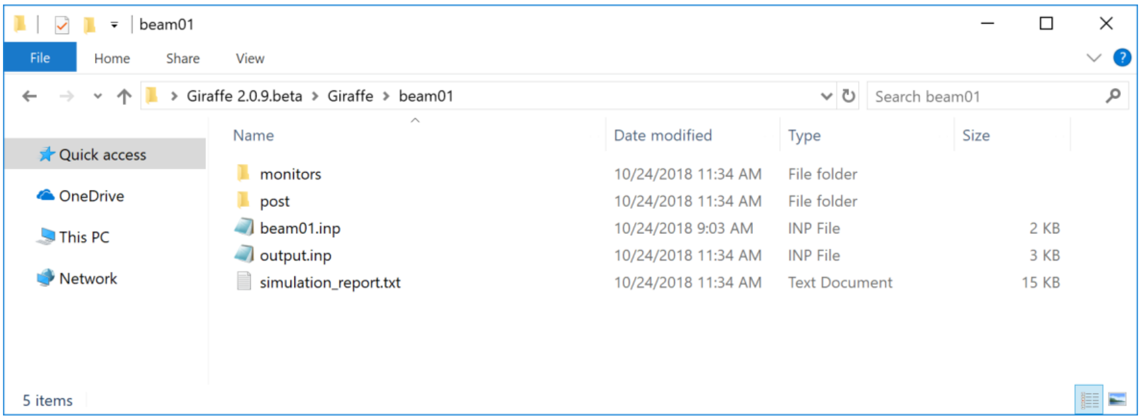


Figure 4 – Example of a simulation folder (from file "beam01.inp")

Each output file type is described next.

File "output.inp"

It is a text file that simply reflects the read information from the input file and may be used to check if Giraffe read input file correctly in some cases.

File "simulation_report.txt"

It is a text file that simply reflects the Giraffe screen output, showing convergence history of all time-steps of simulation.

Post files

Folder "/post" is always created after solving a simulation. Inside it, Giraffe creates sub-folders with outputs for each solution step. Figure 5 shows an example of "/post" folder contents for the example "beam01.inp". Note that in this case a single folder "/post/solution_1" was created. This is because a single solution step was requested in the input file. In case the user requests more solution steps, more folders will be automatically created inside the "/post" folder. A concomitant solution also creates a sub-folder on "/post".

Note that there are ".pvd" files inside the "/post" folder. These are to be used together with Paraview™ post processor. Paraview™ ".pvd" files creates links to other files located inside solution folders, which contain the simulation results established by PostFiles keyword. Depending on which results are requested, more or less ".pvd" files will be saved. For this example, only mesh and render mesh were requested, which lead for only two types of ".pvd" files. Paraview™ ".pvd" files are very useful for creating animations or high-quality images. These files links Paraview™ to read the whole time series of subsequent node positions, node and element results. Inside PARAVIEW™, when referring to element results meaning, one may look at each element results sequence list, contained in each element presented on this user's manual.

The "whole_solution_mesh.pvd" and "whole_solution_rendermesh.pvd" files contains the same contents of the solutions ".pvd" files, but encompassing all solution steps. These are useful for a visualization in Paraview™ of the whole time-history of the simulation. For current example, since a single solution step was requested, ".pvd" files for solution 1 and whole ".pvd" files will be the same.

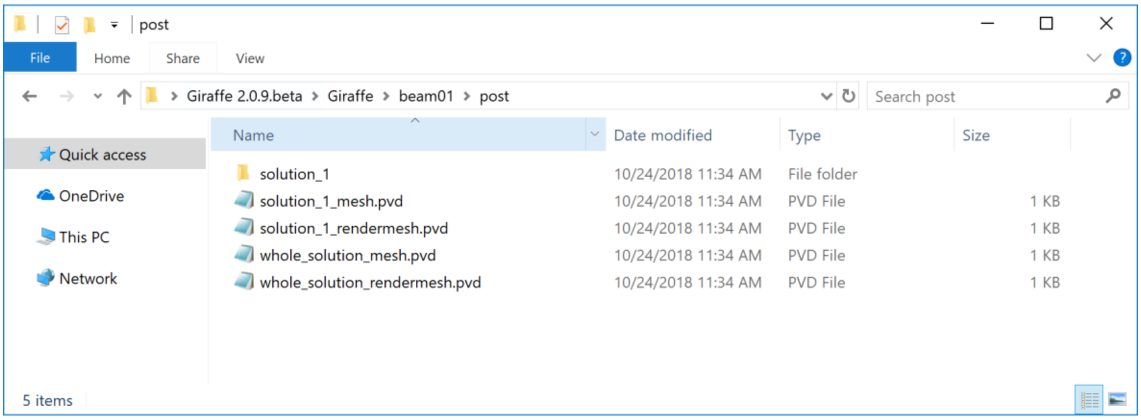


Figure 5 – "/post" folder example

Also, inside the "/post/solution_i" folder (for i = 1,...,n – solution steps), Giraffe writes configuration text files, with nodes and elements results, adopting the same sampling employed for saving Paraview™ post-processing files. An example of inside contents of "/post/solution_1" is shown in Figure 6.

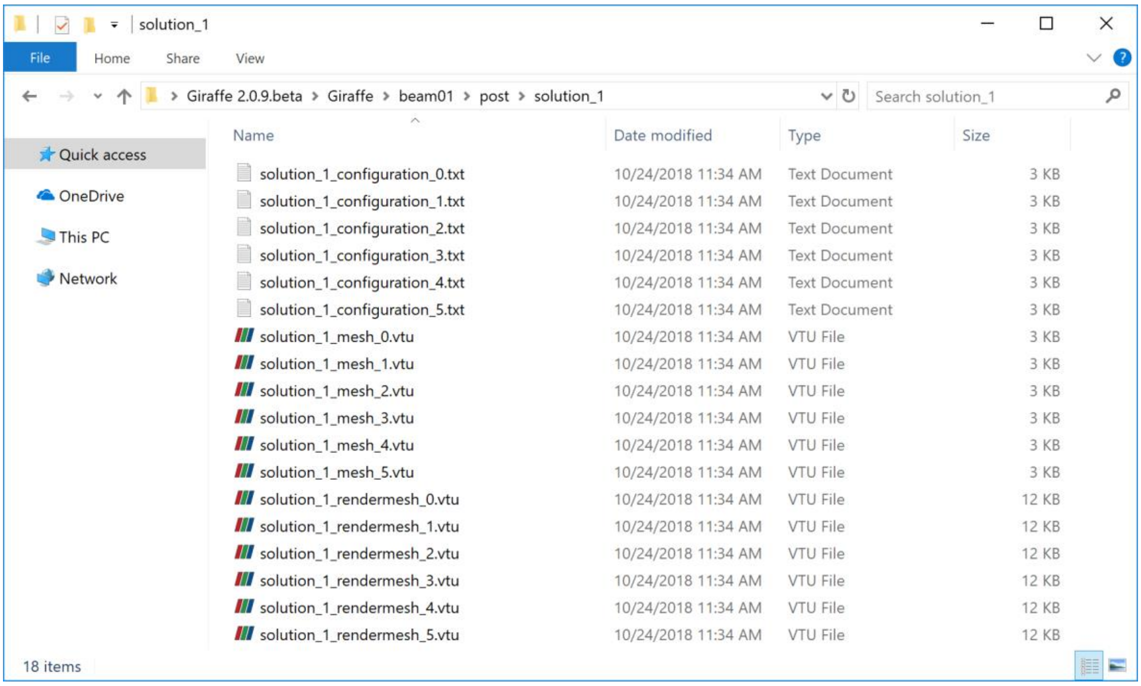


Figure 6 – "/post/solution_1" folder

Note that in "/post/solution_1" there are also ".vtu" files, which are to be read by Paraview™, being referred in already mentioned ".pvd" files.

Monitor files

When requested, monitors are very useful for analyzing and creating time series with information about a given node, element, contact region or node set of interest. Monitors are created from the beginning of the simulation. They are updated until the simulation finishes, during the whole solution process. The Figure 7 shows an example of "monitors" folder.

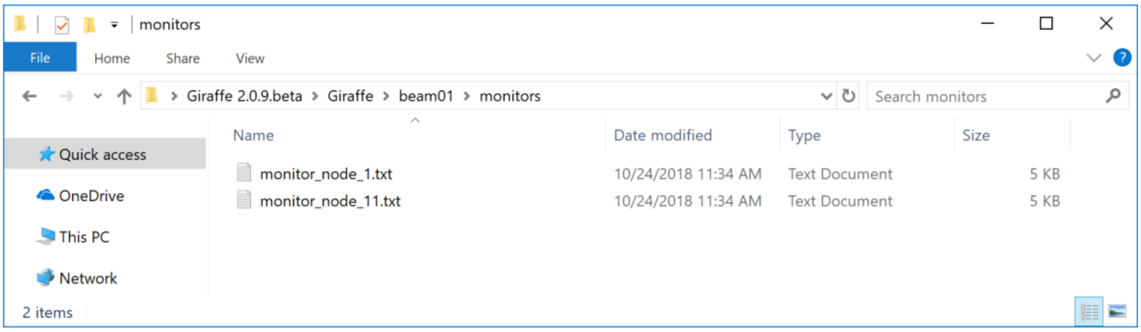


Figure 7 – Monitors folder example

Tutorials

This document has no tutorials. A specific tutorials document containing examples of Giraffe input files is available.

Nodes

Starts a command block for creation of nodes to be used to compound a finite element mesh or particles positions.

Syntax:

```
Nodes   N
Node    ID      X       Y       Z
```

- N: number of nodes
- ID: current node identification number
- X: current node X coordinate (on a global coordinate system)
- Y: current node Y coordinate (on a global coordinate system)
- Z: current node Z coordinate (on a global coordinate system)

Example:

```
Nodes   3
Node    1       1.0     0.0     3.0
Node    2       0.0     2.5     -5.1
Node    3       0.0     0.0     -10.0
```

Additional information:

Each node is defined by the keyword Node followed by the node identification number (must be an ascending sequence starting from number one), coordinates X, Y and Z.

After reading the input file, Giraffe checks all nodes, elements and particles connectivity. Based on this check, it evaluates how many degrees of freedom (DOFs) and which nature of DOFs have to be assigned for each node.

Elements

Starts a command block for creation of elements to be used to compound a finite element mesh.

Syntax:

```
Elements        N
Name   ID       data
```

- N: number of elements
- Name: current element name
- ID: current element identification number
- data: current element data (depends on element resources and requirements)

Example:

```
Elements    2
Beam_1      1     Mat   1     Sec   1     CS    1     Nodes  1 2 3
Shell_1     2     Mat   1     Sec   1     Nodes  1 2 3 4 5 6
```

Additional information:

Each element is defined by a specific keyword followed by the element identification number (must be an ascending sequence starting from number one) and additional data. Each element available and its input data is explained next.

Beam_1

Creates an initially straight beam finite element defined by three nodes.

Syntax:

Beam_1	EID	Mat	MID	Sec	SID	CS	CSID	Nodes	ID1 ID2 ID3

- EID: current element identification number
- MID: material identification number
- SID: element cross-section identification number
- CSID: coordinate system identification number
- ID1, ID2 and ID3: identification number of nodes defining the element

Example:

Beam_1	1	Mat	1	Sec	1	CS	1	Nodes	1 2 3

Additional information:

This is a 3D beam element, with three equally spaced nodes. Three displacement and three rotation DOFs are defined for each node. Then, each element has eighteen DOFs. The Beam_1 element uses two Gauss points for integration. The nodes used to create the Beam_1 element have to be established by the keyword Nodes, followed by the node numbers corresponding to nodes 1, 2 and 3 in a local reference (Figure 8). The only environmental field loading that can be used with this element is the self-weight induced by the gravitational field, defined by BoolTable environment data.

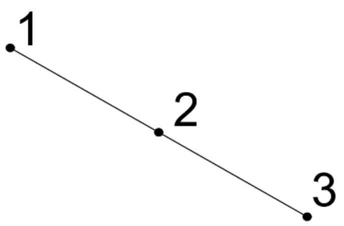


Figure 8 – Beam_1 element local nodes numbering reference

This element internally assumes its local framework containing $\mathbf{e}_3$ axis aligned with axial direction of the beam. Then, local directions $\mathbf{e}_1$ and $\mathbf{e}_2$ are orthogonal to the beam direction. For establishing the cross section correct alignment one has to choose the element coordinate system such that $\mathbf{e}_3$ lies at the element axial direction and $\mathbf{e}_1$ is aligned with the direction used to define the cross section (see Sections). More details about theoretical details of this beam formulation can be found in [1] and [2]. For post-processing Beam_1 element results, one has the following sequence (to be chosen in Paraview™ post-processing):

Table 1 – Beam_1 element results

Element result index	Meaning
0	Shear force in direction $\mathbf{e}_1$
1	Shear force in direction $\mathbf{e}_2$
2	Axial force (in direction $\mathbf{e}_3$)
3	Bending moment around direction $\mathbf{e}_1$
4	Bending moment around direction $\mathbf{e}_2$
5	Torsion moment (around direction $\mathbf{e}_3$)

Pipe_1

Creates an initially straight pipe finite element defined by three nodes.

Syntax:

Pipe_1 EID	PipeSec	PSID	CS	CSID	Nodes	ID1 ID2 ID3

- EID: current element identification number
- PSID: pipe cross-section identification number
- CSID: coordinate system identification number
- ID1, ID2 and ID3: identification number of nodes defining the element

Example:

Pipe_1 1	PipeSec	1	CS	1	Nodes	1 2 3

Additional information:

This is a 3D pipe element. The structural behavior is the same as Beam_1 element. However, the input attributes are different. It is possible to make use of Pipe_1 with environmental loading, such as weight, Morison sea current drag loading, internal and external pressure loading.

For post-processing Pipe_1 element results, one has the following sequence (to be chosen in Paraview™ post-processing):

Table 2 – Pipe_1 element results

Element result index	Meaning
0	Shear force in direction $\mathbf{e}_1$
1	Shear force in direction $\mathbf{e}_2$
2	Axial force (in direction $\mathbf{e}_3$)
3	Bending moment around direction $\mathbf{e}_1$
4	Bending moment around direction $\mathbf{e}_2$
5	Torsion moment (around direction $\mathbf{e}_3$)

Shell_1

Creates an initially planar shell element defined by six nodes.

Syntax:

Shell_1	EID	Mat	MID	Sec	SID	CS	CSID	Nodes ID1 ID2 ID3 ID4 ID5 ID6

- EID: current element identification number
- MID: material identification number
- SID: shell section identification number
- CSID: optional coordinate system identification number. Necessary for composite shell structures
- ID1, ID2, ID3, ID4, ID5 and ID6: identification number of nodes defining the element

Example:

Shell_1	1	Mat	1	Sec	1	Nodes 1 2 3 4 5 6

Example for composite shell structures:

Shell_1	1	Mat	1	Sec	1	CS	1	Nodes 1 2 3 4 5 6

Additional information:

This is a triangular shell element with six nodes. It uses three points to integrate along the element area. The sequence of nodes must be provided according to the numbering sequence shown in Figure 9. Note that the direction chosen to increase the number of nodes implicitly defines the normal of the shell element, according to the right-hand rule. The user can establish the external normal direction $\boldsymbol{n}$ by performing the cross product between the vectors $\boldsymbol{v}_1 = (\boldsymbol{P}_2 - \boldsymbol{P}_1)$ and $\boldsymbol{v}_2 = (\boldsymbol{P}_3 - \boldsymbol{P}_1)$, such that $\boldsymbol{n} = \frac{\boldsymbol{v}_1 \times \boldsymbol{v}_2}{\|\boldsymbol{v}_1 \times \boldsymbol{v}_2\|}$.This external normal direction is used to define pressure loading on shell elements.

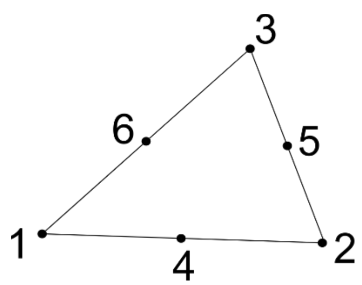


Figure 9 – Shell_1 element local nodes numbering reference

More details about theoretical details of this shell formulation can be found in [3].

To establish a local coordinate system, Giraffe uses the reference configuration of the shell element. The direction $\mathbf{e}_3^{\mathrm{r}}$ is the normal direction of the shell, at reference configuration. The local $\mathbf{e}_1^{\mathrm{r}}$ is defined by the global x direction projection on the shell reference plane. If this projection is null, then $\mathbf{e}_1^{\mathrm{r}}$ is defined by the global y direction projection on the shell plane. Finally, $\mathbf{e}_2^{\mathrm{r}} = \mathbf{e}_3^{\mathrm{r}} \times \mathbf{e}_1^{\mathrm{r}}$.

When dealing with composite shell structures it is necessary to set a local coordinate system. Each local coordinate system will define the principal material axes orientation and the stacking order of the laminas in the shell element.

For post-processing Shell_1 element results, one has the following sequence (to be chosen in Paraview™ post-processing):

Table 3 – Shell_1 element results

Element result index	Meaning
0	Force in direction $\mathbf{e}_1$ (cutting plane with normal direction $\mathbf{e}_1$)
1	Force in direction $\mathbf{e}_2$ (cutting plane with normal direction $\mathbf{e}_1$)
2	Force in direction $\mathbf{e}_3$ (cutting plane with normal direction $\mathbf{e}_1$)
3	Moment in direction $\mathbf{e}_1$ (cutting plane with normal direction $\mathbf{e}_1$)
4	Moment in direction $\mathbf{e}_2$ (cutting plane with normal direction $\mathbf{e}_1$)
5	Moment in direction $\mathbf{e}_3$ (cutting plane with normal direction $\mathbf{e}_1$)
6	Force in direction $\mathbf{e}_1$ (cutting plane with normal direction $\mathbf{e}_2$)
7	Force in direction $\mathbf{e}_2$ (cutting plane with normal direction $\mathbf{e}_2$)
8	Force in direction $\mathbf{e}_3$ (cutting plane with normal direction $\mathbf{e}_2$)
9	Moment in direction $\mathbf{e}_1$ (cutting plane with normal direction $\mathbf{e}_2$)
10	Moment in direction $\mathbf{e}_2$ (cutting plane with normal direction $\mathbf{e}_2$)
11	Moment in direction $\mathbf{e}_3$ (cutting plane with normal direction $\mathbf{e}_2$)

Mass_1

Creates a single-node lumped mass element.

Syntax:

Mass_1	EID	Mass	MV	Node	NID

- EID: current element identification number
- MV: mass value
- NID: identification number of the node defining the element

Example:

Mass_1	1	Mass	150.3	Node	1

Additional information:

This is a lumped mass element. It can be used to model a portion of mass not included in the finite element model, but that may affect the system response due to gravitational field loads and/or inertial loads.

This element has no direct influence in system's stiffness, but only in the mass matrix coefficients related to translational DOFs and external loads, corresponding to inertial and gravitational field.

This element has no specific results for post-processing.

SpringDashpot_1

Creates a two-node spring and dashpot element.

Syntax:

SpringDashpot_1	EID	Stiffness	SV	Damping	DV	Nodes ID1 ID2

- EID: current element identification number
- SV: stiffness value
- DV: damping value
- ID1 and ID2: identification number of the nodes defining the element

Example:

SpringDashpot_1	1	Stiffness	200.2	Damping	1.34	Nodes 1 2

Additional information:

This is a spring and dashpot element. It can be used in applications were the stiffness/damping coefficients are known a priori.

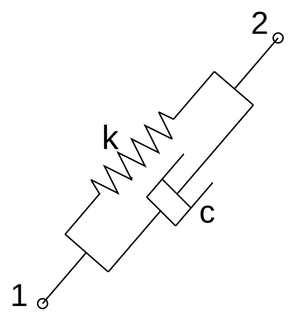


Figure 10 – SpringDashpot_1 element local nodes numbering reference

It is a two-node element (see local nodes numbering in Figure 10), with a linear stiffness "k" – proportional to the relative displacement of the nodes – and a linear damping "c" – proportional to the relative velocity of the nodes, projected on the direction of the line that connects the two nodes. The element is geometrically nonlinear. Then, the direction affected by the stiffness/damping follows the current position of the element nodes. It can handle large rigid body rotations and translations.

Table 4 – SpringDashpot_1 element results

Element result index	Meaning
0	Spring elongation (positive value: augmenting the length, negative value: diminishing the length)
1	Elastic force from spring (positive value: tension, negative value: compression)
2	Damping force from dashpot (positive value: relative velocity increasing the damper length, negative value: relative velocity decreasing the damper length)

RigidBody_1

Creates a single-node rigid body element.

Syntax:

RigidBody_1	EID	RigidBodyData	RBID	CS	CID	Node	NID

- EID: current element identification number
- RBID: rigid body data identification number
- CID: coordinate system identification number
- NID: identification number of the node defining the element

Example:

RigidBody_1	1	RigidBodyData	1	CS	1	Node	1

Additional information:

This is a rigid body element with mass and inertia properties. It can be used in multibody systems to represent relatively stiff components.

It is a single-node element that accounts for mass and inertia properties from a 3D solid body (provided within RigidBodyData). Each rigid body element has a unique identification number that follows the keyword "RigidBody_1". Each element receives its properties and therefore it is necessary to indicate the identification number of the RigidBodyData.

Rigid bodies are oriented in Giraffe according to local Coordinate System (CS). The objective of this local CS is to orient the global axes from the CAD file in the Giraffe three-dimensional space. The CAD origin is placed at the Rigid Body node. This is illustrated in Figure 11.

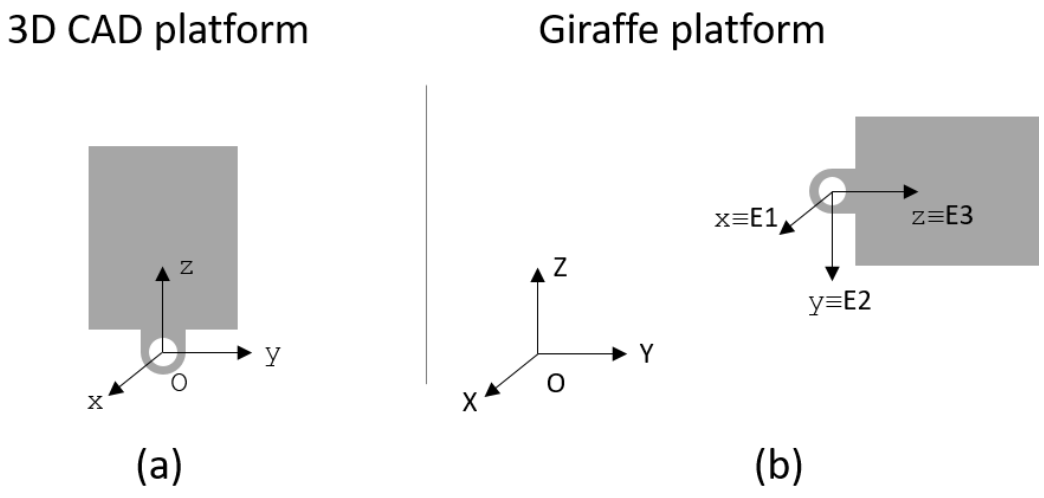


Figure 11 – Orientation of RigidBody_1 elements

In Figure 11 (a) a generic CAD model is represented. It was modelled in the yz plane. Suppose the orientation shown in Figure 11 (b) is desired in Giraffe, then the local CS represented by E1, E2 and E3 has to be specified. For example, in Figure 11 (b) the axes were oriented according to the following CS:

```
CS      1
CSYS    1       E1      1       0       0       E3      0       1       0
```

This means that the x axis of the CAD file was aligned with the X axis in Giraffe, and the z axis was aligned with the Y axis. Note that the main objective of this CS is to orient the geometry in Giraffe. It is a very important feature since it is directly related to the inertia properties and it is used for postprocessing purposes (to visualize geometry in ParaView™).

The Node used to create RigidBody_1 can be placed anywhere. Therefore, the CAD model has to be generated in a way that its origin coincides with the position of this node. It is very convenient to create the RigidBody_1 node at the center of mass or at some place that will be constrained during the simulation, Figure 12 and Figure 13 clarify this question.

Example 1:

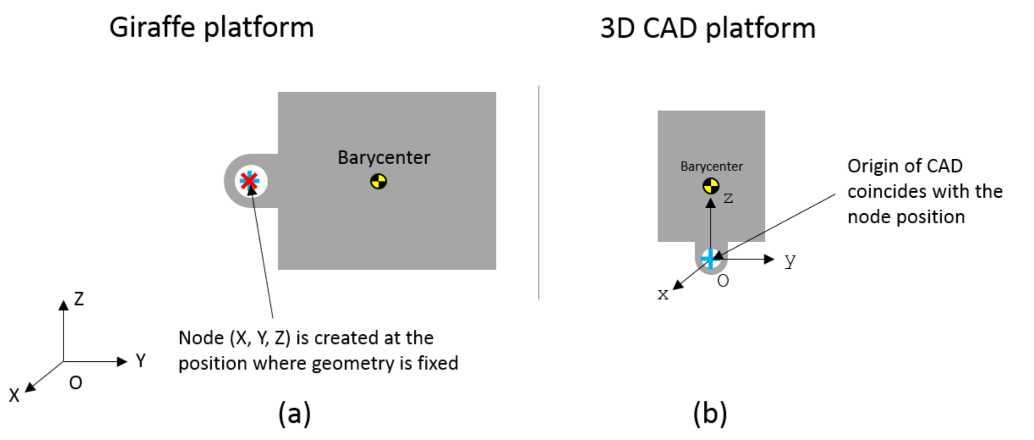


Figure 12 – RigidBody_1 positioning in Giraffe platform: Example 1

Example 2:

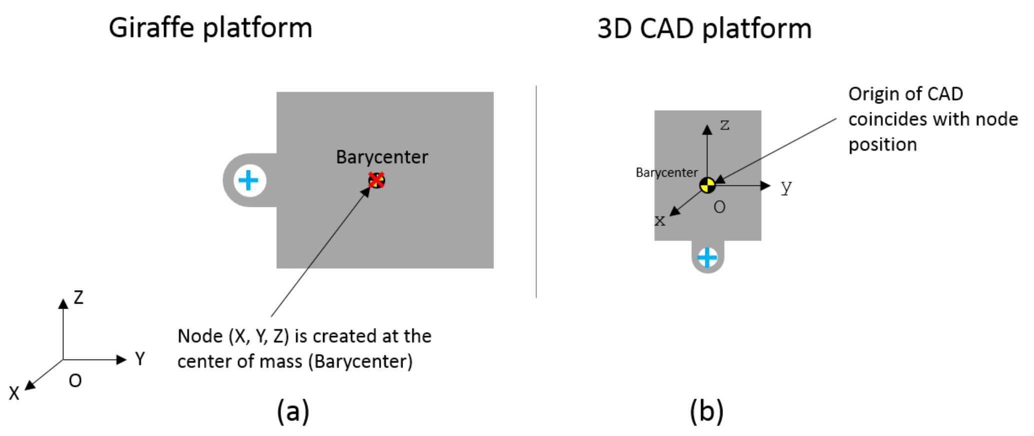


Figure 13 – RigidBody_1 positioning in Giraffe platform: Example 2

RigidBody_1 is usually used in conjunction with special constraints, especially rigid node set. For example, one may be interested in monitoring kinematic quantities of a point located anywhere in the body. In order to do that, it is necessary to create a node at the position of interest and then to define a rigid node set from the RigidBody_1 Node (pilot node) to the node of interest (slave node). Nodes at different locations of the body can be added to rigid node set. The kinematic quantities can be obtained via Monitors.

To obtain quantities such as Kinetic Energy (T), Linear Momentum (L) and Angular Momentum about the center of mass (HG) one can request element Monitors.

This element has no specific results for post-processing using Paraview™.

Truss_1

Creates a two-node truss element.

Syntax:

This element type has two possible syntax entries shown below:

Truss_1	EID	Mat	MID	Sec	SID	Nodes	ID1 ID2
Truss_1	EID	PipeSec		PSID	Nodes	ID1 ID2	

- EID: current element identification number
- MID: material identification number
- CSID: cross-section data identification number
- PSID: pipe cross-section data identification number
- ID1 and ID2: identification number of the nodes defining the element

Example:

Truss_1	1	Mat	1	Sec	1	Nodes	1 2
Truss_1	2	PipeSec		1	Nodes	2 3	

Additional information:

This is a 3D truss element. There are two possible entries for establishing such element: by material and cross-section data or by pipe cross-section data. When establishing material data, this element can handle large strain. Material model may be chosen between:

(i) linear-elastic (Hooke)
(ii) elastic-plastic with isotropic hardening

Alternatively, if pipe cross-section data is defined, the element employs only axial stiffness and adopts is as constant. Additionally, it may be used to define environmental loading such as weight and Morison sea current drag and added mass loading.

For post-processing Truss_1 element results one has the following sequence (to be chosen in Paraview™ post-processing):

Table 5 – Truss_1 element results

Element result index	Meaning
0	Element axial tension
1	Cross-section area
2	Element length
3	Plastic deformed length
4	Kirchhoff stress

Particles

Starts a command block for creation of particles.

Syntax:

Particles		N
Name	ID	data

- N: number of particles
- Name: current particle name
- ID: current particle identification number
- data: current particle data (depends on particle resources and requirements)

Example:

Particles		2							
Sphere	1	Mat	1	CS	1	Radius	2.5	Node	1
Sphere	2	Mat	1	CS	1	Radius	1.5	Node	1

Additional information:

Each particle is defined by a specific keyword followed by the particle identification number (must be an ascending sequence starting from number one) and additional data. Each particle available and its input data is explained next.

Sphere

Creates a spherical particle.

Syntax:

Sphere	PID	Mat	MID	CS	CSID	Radius	RV	Node	NID

- PID: current particle identification number
- MID: material identification number
- CSID: coordinate system identification number
- RV: sphere radius value
- NID: identification number of the node defining the particle

Example:

Sphere	1	Mat	1	CS	1	Radius	2.5	Node	1

Additional information:

This is a spherical particle centered at a given nodal position. Three displacement and three rotation DOFs are defined for the node. In order to provide Giraffe the necessary data to evaluate the sphere mass and its moment of inertia, one has to choose a material identification number and a radius value for the particle. Furthermore, a coordinate system has to be chosen, as the reference orientation of the sphere. This CS is used for rendering plot of the sphere. The particle follows a rigid body kinematics behavior.

Materials

Starts a command block for creation of materials.

Syntax:

```
Materials       N
Name   ID       data
```

- N: number of materials
- Name: current material name
- ID: current material identification number
- data: current material data (depends on material resources and requirements)

Example:

```
Materials       1
Hooke  1        E       210E9   Nu      0.3     Rho     7800
```

Additional information:

Each material is defined by a specific keyword followed by the material identification number (must be an ascending sequence starting from number one) and additional data. Each material available and its input data is explained next.

Hooke

Creates a linear-elastic material (Hooke's law)

Syntax:

```
Hooke  MID    E      EV     Nu     NV     Rho    RV
```

- MID: current material identification number
- EV: Young's Modulus value
- NV: Poisson's ratio value
- RV: specific mass value

Example:

```
Hooke  1      E      210E9  Nu     0.3    Rho    7800
```

Additional information:

Even defining a Hooke material behavior, each element formulation makes use of different techniques to mount the constitutive equation. Different strain energy functions can be used. More details on the constitutive equation assumed for each element formulation can be found in the papers referenced in this manual, such as [3] and [1].

ElasticPlasticIsoHardening

Creates an elastic-plastic material with isotropic hardening rule.

Syntax:

```
ElasticPlasticIsoHardening   MID   E   EV   Nu   NV   Rho   RV   H   HV   YieldingStrength   YSV
```

- MID: current material identification number
- EV: Young's Modulus value
- NV: Poisson's ratio value
- RV: specific mass value
- HV: linear hardening slope value
- YSV: yielding strength value

Example:

```
ElasticPlasticIsoHardening	1	E	210000 Nu	0.3	Rho	8E-9	H
	10000	YieldingStrength	250
```

Additional information:

This material model is not available for all elements. Please, check availability for the element of interest prior to usage.

Orthotropic

Creates an orthotropic material

Syntax:

Orthotropic	MID	E1	E1V	E2	E2V	G12	G12V	G23	G23V	Nu12
	N12V	Rho	RV							

- MID: current material identification number
- E1V: Young's Modulus value at direction principal direction 1
- E2V: Young's Modulus value at direction principal direction 2
- G12V: Shear Modulus value associated with directions 1 and 2
- G23V: Shear Modulus value associated with directions 2 and 3
- N12V: Poisson's ratio value associated with directions 1 and 2
- RV: specific mass value

Example:

Orthotropic 1	E1	41e9	E2	10.4e9	G12	4.3e9	G23	4.3e9	Nu12	0.28
	Rho	1970								

Additional information:

This material model is not available for all elements. Please, check availability for the element of interest prior to usage. Orthotropic material constants orientation as defined in Figure 14.

(a) Directions 1 and 2 (b) Direction 3

Figure 14 – Orthotropic material orientation

CoordinateSystems

Starts a command block for creation of coordinate systems.

Syntax:

CoordinateSystems			N						
CS	ID	E1	E1XV	E1YV	E1ZV	E3	E3XV	E3YV	E3ZV

- N: number of coordinate systems
- ID: current coordinate system identification number
- E1XV, E1YV and E1ZV: components of direction E1
- E3XV, E3YV and E3ZV: components of direction E3

Example:

CoordinateSystems			2						
CS	1	E1	1	0	0	E3	0	1	0
CS	2	E1	0	1	0	E3	0	0	1

Additional information:

Each coordinate system is defined by the keyword CS followed by an identification number (must be an ascending sequence starting from number one). The coordinate systems are Cartesian and are defined by three unit-vectors, named E1, E2 and E3. Only E1 and E3 components have to be defined. The orientation E2 is internally calculated using the cross product.

CADData

Starts a command block for creation of Computer Aided Design (CAD) data.

Syntax:

CADData	N	
Name ID	data	

- N: number of CAD data inputs
- Name: current CAD data type name
- ID: current CAD data identification number
- data: current CAD data information (depends on CAD data resources and requirements)

Example:

CADData	2	
NURBSSurface	1	box.txt
STLSurface	2	body.stl

Additional information:

Each CAD data is defined by a specific keyword followed by the CAD data identification number (must be an ascending sequence starting from number one) and additional data. Each CAD data available and its input data is explained next.

STLSurface

Creates a STL (stereolithography) CAD information for usage with other Giraffe resources.

Syntax:

STLSurface	CID	file

- CID: current CAD data identification number
- file: *STL* file name (the file must be located inside a folder named "CAD" located in the same directory of Giraffe input file)

Example:

STLSurface	2	body.stl

Additional information:

The stl file must be input using the ASCII syntax.

(see https://en.wikipedia.org/wiki/STL_(file_format) for details)

NURBSSurface

Creates a NURBS (non uniform rational basis spline) surface CAD information for usage with other Giraffe resources.

Syntax:

NURBSSurface	CID	file

- CID: current CAD data identification number
- file: NURBS file name (the file must be located inside a folder named "CAD" located in the same directory of Giraffe input file)

Example:

STLSurface	2	body.stl

Additional information:

The input file for a NURBS surface in Giraffe has to follow a specific syntax. It is a text file with information provided as explained next. More information on the theory of NURBS surfaces is well-presented in [4], but the basic idea is herein presented next.

First, we have to define a net of control points $\mathbf{P}_{\mathrm{i,j}}$, where $0 \leq i \leq n$ and $0 \leq j \leq m$. Thus, we consider a total of $(n+1)(m+1)$ control points. Each one is associated with a weight, given by $w_{\mathrm{i,j}}$.

A polynomial basis is constructed to represent points on the surface. Each point is given by a mapping procedure, starting from a parametric plane with coordinates u and v. The basis is independent for u and v. The polynomial degrees are also independent and are p and q, respectively for u and v. The so-called knot-vectors are non-decreasing sequences of real numbers, organized as $\mathbf{U}$ and $\mathbf{V}$ by:

$$\begin{aligned} \mathbf{U} &= \left[u_0, u_1, \dots, u_{n+p+1}\right], \\ \mathbf{V} &= \left[v_0, v_1, \dots, v_{m+q+1}\right]. \end{aligned} \tag{1}$$

The knot inputs in $\mathbf{U}$ and $\mathbf{V}$ are used to establish the polynomial functions $N_{i,p}(u)$ and $N_{j,q}(v)$ by de Cox de Boor recursive formula. At the end, the NURBS surface is given by the parameterization:

$$\mathbf{s}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{i,j} \mathbf{P}_{\mathrm{i,j}}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{i,j}} \tag{2}$$

The range for u and v is defined by the coordinates organized in each knot vector. The control points and weights have direct influence on the surface location in space. NURBS surfaces are not usually interpolatory on control points.

The Giraffe NURBS input file contains all the information needed to establish the parameterization (2). Therefore, one needs to input keywords followed by numeric input data. The keywords are defined as follows:

- UDim: dimension $n + 1$ (for direction u);
- VDim: dimension $m + 1$ (for direction m);
- UOrder: polynomial degree for the basis along direction u;
- VOrder: polynomial degree for the basis along direction v;
- UKnotVector: $\mathbf{U}$ knot vector;
- VKnotVector: $\mathbf{V}$ knot vector;
- Weights: weights;
- ControlPoints: control points.

Weights $w_{\mathrm{i,j}}$ and control points $\mathbf{P}_{\mathrm{i,j}} = (x_{ij}, y_{ij}, z_{ij})$ are read assuming that they are in a sequence associated with (u, v) as follows:

$$(v_0, u_0), (v_0, u_1) \dots (v_0, u_{n+1})$$

$$(v_1, u_0), (v_1, u_1) \dots (v_1, u_{n+1})$$

$$\dots$$

$$(v_{m+1}, u_0), (v_{m+1}, u_1) \dots (v_{m+1}, u_{n+1})$$

An example of a simple NURBS input file for Giraffe is:

```
UDim
2
UOrder
1
UKnotVector
0.0000000000000000e+00
0.0000000000000000e+00
1.0000000000000000e+00
1.0000000000000000e+00
VDim
2
VOrder
1
VKnotVector
0.0000000000000000e+00
0.0000000000000000e+00
1.0000000000000000e+00
1.0000000000000000e+00
Weights
1.0000000000000000e+00
1.0000000000000000e+00
```

```
1.0000000000000000e+00
1.0000000000000000e+00
ControlPoints
-1.0000000000000000e+00    -1.0000000000000000e+00    0.0000000000000000e+00
1.0000000000000000e+00     -1.0000000000000000e+00    0.0000000000000000e+00
-1.0000000000000000e+00    1.0000000000000000e+00     0.0000000000000000e+00
1.0000000000000000e+00     1.0000000000000000e+00     0.0000000000000000e+00
```

Sections

Starts a command block for creation of cross-sections (here referenced as "sections").

Syntax:

Sections		N
Name	ID	data

- N: number of sections
- Name: current section name
- ID: current section identification number
- data: current section data (depends on section resources and requirements)

Example:

Sections	1				
Rectangle	1	B	1.0	H	2.5

Additional information:

Each section is defined by a specific keyword followed by the section identification number (must be an ascending sequence starting from number one) and additional data. Each section available and its input data is explained next.

Sections are to be used as parameters for elements Beam_1 and Truss_1. Note that the local coordinate system used to define the cross section has origin on the cross-section intersection with the beam axis position, defined by the nodes of the mesh. The plane of the cross-section is parallel to the plane formed by $\mathbf{e}_1$ and $\mathbf{e}_2$, defined in the beam element coordinate system (see Beam_1 input data).

General

Creates a general cross-section for usage with beam and truss elements.

Syntax:

```
General   SID   A   AV  I11   I11V   I22   I22V   I12   I12V   JT  JTV
```

- SID: current section identification number
- AV: cross-section area value
- I11V: moment of inertia around E1 axis
- I22V: moment of inertia around E2 axis
- I12V: product of inertia related to axis E1 and E2
- JTV: cross-section moment of torsion value

Example:

```
General   1   A  0.1  I11   0.01   I22   0.01   I12   0.0   JT  0.02
```

Additional information:

It is assumed that the element axis passes through cross-section centroids and shear centers. Thus, if this is not the case, use UserDefined cross-section, instead.

Rectangle

Creates a rectangular cross-section for usage with beam and truss elements.

Syntax:

Rectangle	SID	B	BV	H	HV

- SID: current section identification number
- BV: base dimension value of the cross-section lying in direction E1
- HV: height dimension value of the cross-section lying in direction E2

Example:

Rectangle	1	B	1.0	H	2.5

Additional information:

It is assumed that the element axis passes through cross-section centroids and shear centers. Thus, if this is not the case, use UserDefined cross-section, instead.

SuperEllipse

Creates a super elliptical cross-section for usage with beam and truss elements.

Syntax:

SuperEllipse	SID	A	AV	B	BV	N	NV	AMeshFDM	MV

- SID: current section identification number
- AV: semi-axis value lying in direction E1
- BV: semi-axis value lying in direction E2
- NV: super ellipse exponent value
- MV: number of divisions in a Finite Difference Method mesh discretization in the direction of radius A (performed to calculate Saint-Venant moment of torsion, prior to FEM simulation)

Example:

SuperEllipse	1	A	1.0	B	2.0	N	3	AMeshFDM	200

Additional information:

It is assumed that the element axis passes through cross-section centroids and shear centers. Thus, if this is not the case, use UserDefined cross-section, instead.

Tube

Creates a tubular cross-section for usage with beam and truss elements.

Syntax:

```
Tube    SID    De    DEV    Di    DIV
```

- SID: current section identification number
- DEV: external diameter value
- DIV: internal diameter value

Example:

```
Tube    1    De    0.2    Di    0.1
```

Additional information:

It is assumed that the element axis passes through cross-section centroids and shear centers. Thus, if this is not the case, use UserDefined cross-section, instead.

A null internal diameter can be used, and leads to a solid cylinder.

UserDefined

Creates a user defined cross-section for usage with beam and truss elements.

Syntax:

```
UserDefined   SID
GA       GAV
EA       EAV
ES1      ES1V
ES2      ES2V
EI11     EI11V
EI22     EI22V
EI12     EI12V
GS1      GS1V
GS2      GS2V
GS1S     GS1SV
GS2S     GS2SV
GJT      GJTV
J11      J11V
J22      J22V
J12      J12V
A        AV
SC       SC1V    SC2V
BC       BC1V    BC2V
Rho      RV
SD    SDID
//Optional keywords block below:
AD       ADID
AC       AC1V    AC2V
AeroLength     ALV
```

- SID: current section identification number
- GAV: equivalent shear stiffness product
- EAV: equivalent axial stiffness product
- ES1V: equivalent ES_1
- ES2V: equivalent ES_2
- EI11V: equivalent bending stiffness EI_{11}
- EI22V: equivalent bending stiffness EI_{22}
- EI12V: equivalent bending stiffness EI_{12}
- GS1V: equivalent GS_1
- GS2V: equivalent GS_2
- GS1SV: equivalent $\mathrm{GS}_1^{\mathrm{S}}$
- GS2SV: equivalent $\mathrm{GS}_2^{\mathrm{S}}$
- GJTV: equivalent torsion stiffness with respect to origin O (GJ_{t})
- J11V: mass Moment of inertia per unit reference length J_{11}
- J22V: mass Moment of inertia per unit reference length J_{22}
- J12V: mass Product of inertia per unit reference length J_{12}
- AV: Cross-section area

- SC1V and SC2V: shear center coordinates (s_1, s_2)
- BC1V and BCV2: barycenter coordinates (g_1, g_2)
- RV: mass per unit reference length ($\bar{\rho}$)
- SDID: section details (defines the external contour of the cross section – to be used for rendering purposes)

  Optional keywords block:
- ADID: aerodynamic data identification number (defines aerodynamic curves to be used to evaluate environment wind forces)
- AC1V and AC2V: aerodynamic center position (c_1, c_2)
- ALV: aerodynamic reference length value (usually cross-section profile chord is employed)

Example:

```
Sections 1
UserDefined 1
GA        656713326.769231
EA        1707454649.6
ES1       144398244.5
ES2       0.0
EI11      20026109.28137880
EI22      1200819.338227490
EI12      0.0
GS1       55537786.35
GS2       0.0
GS1S      27768893.1727725
GS2S      0.0
GJT       1373989.013
J11       0.787026095
J22       0.0471921999923403
J12       0.0
A         0.008537273248
SC        0.0 0.04228465
BC        0.0 0.08456930
Rho       67.10296773
SD        1
```

Additional information:

For this cross-section, no assumptions are made with respect to the beam axis position. On 3D space, the beam element is defined by its nodes, normally, which are given in section Nodes. The properties to define the beam constitutive behavior will vary according to the chosen position of the axis. Accordingly, the results of internal loads have to be re-interpreted.

Many properties are required to use this cross section, but it permits a myriad of applications, such as composite beams, thin-walled cross-sections and complex-shape cross-sections.

Important: when the UserDefined cross section is used, the material data assigned to the beam element is ignored.

Figure 15 shows an example of UserDefined cross section. The beam axis intersection with the cross-section occurs at point O, origin of a local coordinate system that has to be used to define all quantities defined from now on. As in other Giraffe's cross-sections for beams, directions $\mathbf{e}_1$ and $\mathbf{e}_2$ define the cross-section plane. A general material point on the cross-section may be described by coordinates (x_1, x_2) using the system $(O, \mathbf{e}_1, \mathbf{e}_2)$. Let ρ be the material specific mass function $\rho = \hat{\rho}(x_1, x_2)$. The cross-section area domain is given by A.

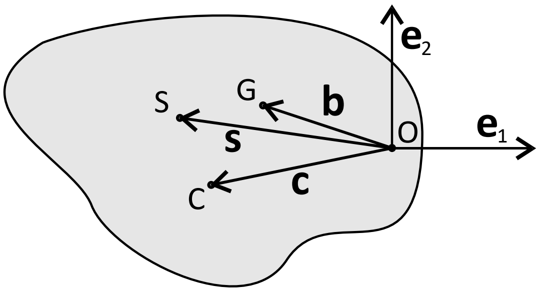


Figure 15 – UserDefined cross-section

The cross-section barycenter is located at material point G, such that

$$\mathbf{b} = (G - O) = g_1\mathbf{e}_1 + g_2\mathbf{e}_2, \quad (1)$$

with

$$g_1 = \frac{\int_A \rho x_1 dA}{\int_A \rho dA}; \; g_2 = \frac{\int_A \rho x_2 dA}{\int_A \rho dA}. \quad (2)$$

The shear center is located at material point S, such that

$$\mathbf{s} = (S - O) = s_1\mathbf{e}_1 + s_2\mathbf{e}_2. \quad (3)$$

For some applications involving fluid-structure interaction, it is also necessary to define the aerodynamic center, located at material point C, such that

$$\mathbf{c} = (C - O) = c_1\mathbf{e}_1 + c_2\mathbf{e}_2. \quad (4)$$

Next, one finds the convention used for all geometric properties evaluation for Giraffe. (see more details in [5]).

- Cross-section area:

$$A = \int_A dA \quad (5)$$

- Moments of inertia with respect to area:

$$I_{11} = \int_A x_2^2 dA \quad (6)$$

$$I_{22} = \int_A x_1^2 dA \quad (7)$$

- Product of inertia with respect to area:

$$I_{12} = -\int_A x_1 x_2 dA \tag{8}$$

- First-order moments (static moments) with respect to area:

$$S_1 = \int_A x_2 dA \tag{9}$$

$$S_2 = -\int_A x_1 dA \tag{10}$$

Note: if the specific mass function ρ is constant on A, one may write:

$$S_1 = Ag_2 \tag{11}$$

$$S_2 = -Ag_1 \tag{12}$$

- First-order moments (static moments) with respect to the shear center position:

$$S_1^s = S_1 - As_2 \tag{13}$$

$$S_2^s = S_2 + As_1 \tag{14}$$

Note: if the specific mass function ρ is constant on A, one may write:

$$S_1^s = A(g_2 - s_2) \tag{15}$$

$$S_2^s = -A(g_1 - s_1) \tag{16}$$

One may also define properties that depend on mass distribution:

- Mass per unit length

$$\bar{\rho} = \int_A \rho dA \tag{17}$$

- Moments of inertia with respect to mass (per unit reference length of beam, since they are integrated in the area and not in the volume):

$$J_{11} = \int_A \rho x_2^2 dA \tag{18}$$

$$J_{22} = \int_A \rho x_1^2 dA \tag{19}$$

- Product of inertia with respect to mass (per unit reference length of beam, since it is integrated in the area and not in the volume):

$$J_{12} = -\int_A \rho x_1 x_2 dA \tag{20}$$

The constitutive equation that Giraffe uses to evaluate the beam generalized stress (internal loads) $\boldsymbol{\sigma}$ relation with generalized strains $\boldsymbol{\epsilon}$ is given by

$$\boldsymbol{\sigma} = \begin{bmatrix} GA & 0 & 0 & 0 & 0 & G(S_1^S - S_1) \\ 0 & GA & 0 & 0 & 0 & G(S_2^S - S_2) \\ 0 & 0 & EA & ES_1 & ES_2 & 0 \\ 0 & 0 & ES_1 & EI_{11} & EI_{12} & 0 \\ 0 & 0 & ES_2 & EI_{12} & EI_{22} & 0 \\ G(S_1^S - S_1) & G(S_2^S - S_2) & 0 & 0 & 0 & GJ_t \end{bmatrix} \boldsymbol{\epsilon}. \quad (21)$$

All equivalent stiffness coefficients in this relation have to be input for using UserDefined cross-section. These values represent equivalent quantities already integrated on cross-section area. If handling a single material homogeneous cross-section, such stiffness coefficients may be evaluated using all geometric quantities previously defined, multiplied by material data (Young Modulus E and Shear Modulus G). If handling a composite material cross-section, equivalent stiffness coefficients evaluation may be non-straightforward and may be obtained by using a third-part software.

A coefficient needs a special attention: the torsion stiffness GJ_t. When disregarding cross-section warping, the moment of torsion is given by $J_t = I_{11} + I_{22}$. This gives exact results for circular or tubular cross sections, that do not experience warping under torsion. For all other shapes of cross section, such approximation gives larger values for J_t than expected, when considering warping properly.

If one uses Saint-Venant torsion theory, it is possible to enhance evaluation of J_t by defining a warping function. With that, one may define the shear-center and the moment of torsion, with respect to the shear-center, named J_t^S. This value usually may be obtained by a CAD software or other third-part software. However, since the beam axis here considered is general, the moment of torsion for input in Giraffe is not J_t^S, but may be obtained by considering a proper transport such that

$$J_t = J_t^S + A(s_1^2 + s_2^2). \quad (22)$$

Table 6 shows the SI units for all quantities needed for input when using UserDefined cross-section.

Table 6 – SI units for all quantities needed for input in UserDefined cross-section

Coefficient	Unit	Coefficient	Unit	Coefficient	Unit
GA	N	GS_1 , GS_2	N.m	A	m²
EA	N	GS_1^S, GS_2^S	N.m	s_1, s_2	m
ES_1, ES_2	N.m	GJ_t	N.m²	g_1, g_2	m
EI_{11}, EI_{22}, EI_{12}	N.m²	J_{11}, J_{22}, J_{12}	Kg.m	$\bar{\rho}$	kg/m

SectionDetails

Starts a command block for creation of section details (cross-section details).

Syntax:

SectionDetails		N
Name	ID	data

- N: number of section details
- Name: current section detail name
- ID: current section detail identification number
- data: current section detail data (depends on section detail resources and requirements)

Example:

SectionDetails		1					
SolidSection		1	AxisPosition	0.1	0.12	NPoints	4
Point	1	0.4	0				
Point	2	0	0.2				
Point	3	-0.4	0				
Point	4	0	-0.2				

Additional information:

Each section detail is defined by a specific keyword followed by the section detail identification number (must be an ascending sequence starting from number one) and additional data. Each section detail available and its input data is explained next.

SolidSection

Creates a solid section details (for post-processing and visualization purposes).

Syntax:

SolidSection	SDID	AxisPosition	XAV	YAV	NPoints	NP
Point PID	XPV	YPV				

- SDID: current section detail identification number
- XAV and YAV: coordinates of the axis position in element local coordinate system (according to the properties employed to establish the element cross-section)
- NP: number of points employed to define the cross-section
- PID: identification number of each point employed to define the cross-section external boundary
- XPV and YPV: coordinates of each point employed to define the cross-section external boundary

Example:

SolidSection	1	AxisPosition	0.1	0.12	NPoints	4
Point 1	0.4	0				
Point 2	0	0.2				
Point 3	-0.4	0				
Point 4	0	-0.2				

Additional information:

The objective of the section details is to be used only for post-processing purposes. This data is not used by Giraffe mathematical model. When using beam elements with UserDefined cross-sections, the section details are then addressed by the PostFiles keyword.

The cross-section points are defined in a local coordinate system, associated with the beam element, as presented in UserDefined cross-section explanation. The cross-section local plane is given by directions E1 and E2 assigned to the beam element (see CoordinateSystems keyword). The coordinates of the beam axis in this local plane are defined after the keyword AxisPosition (origin for evaluating properties for UserDefined cross sections). Then, the number of points that will be used to define the cross section external boundary is defined by the keyword NPoints. Finally, each point is defined in a list by the keyword Point, followed by an ascending identification number and the coordinates of the point.

MultiCellSection

Creates a multi-cell section details (for post-processing and visualization purposes).

Syntax:

```
MultiCellSection.  SDID  AxisPosition   XAV    YAV    NPoints    NP    NWebs NW
Point    PID    XPV    YPV
Web      WID    W1     W2
```

- SDID: current section detail identification number
- XAV and YAV: coordinates of the axis position in element local coordinate system (according to the properties employed to establish the element cross-section)
- NP: number of points employed to define the cross-section
- NW: number of webs employed to define the cross-section
- PID: identification number of each point employed to define the cross-section external boundary
- XPV and YPV: coordinates of each point employed to define the cross-section external boundary
- WID: identification number of each web
- W1 and W2: identification number of the points connecting web connections with the external cross-section boundary

Example:

```
MultiCellSection  1    AxisPosition   0.0    0.0    NPoints    6   NWebs 1
Point    1    0.4     +0.2
Point    2    0.4     -0.2
Point    3    0       -0.2
Point    4    -0.4    -0.2
Point    5    -0.4    +0.2
Point    6    0       +0.2
Web   1   3   6
```

Additional information:

This type of section detail is similar to the solid section details. The only difference is the presence of webs in visualization. Rendering also considers the structure as thin-walled and not solid cross-section.

PipeSections

Starts a command block for creation of pipe sections.

Syntax:

PipeSections		N									
PS	ID	EA	EAV	EI	EIV	GJ	GJV	GA	GAV	Rho	RV
	CDt	CDTV	CDn	CDNV	CAt	CATV	CAn	CANV	De	DEV	Di
	DIV										

- N: number of pipe sections
- ID: current pipe section identification number
- EAV : equivalent axial stiffness product value
- EIV: equivalent bending stiffness product value
- GJV: equivalent torsional stiffness product value
- GAV: equivalent shearing stiffness product value
- RV: equivalent mass per unit reference length value
- CDTV: drag coefficient in tangential direction
- CDNV: drag coefficient in normal direction
- CATV: added mass coefficient in tangential direction
- CANV: added mass coefficient in normal direction
- DEV: external diameter value
- DIV: internal diameter value

Example:

PipeSections		2									
PS	1	EA	400000000		EI	15000	GJ	2250000		GA	
	200000000		Rho	120	CDt	0.0	CDn	1.0	CAt	0	CAn
	1.0	De	0.25	Di	0.2						
PS	2	EA	6080489749		EI	110821607.7		GJ	85247390.5		GA
	2338649904		Rho	237	CDt	0.0	CDn	1.0	CAt	0	CAn
	1.0	De	0.4	Di	0.2						

Additional information:

Each pipe cross section centroid must lie at the pipe axis, defined by the nodes of the mesh. More details about theoretical details of the pipe section attributes can be found in [2], together with applications for offshore risers simulations.

This cross-section is to be used with Pipe_1 element type. Alternatively, the user may also use it with Truss_1 element type. In this case, only axial stiffness and hydrodynamic coefficients are considered in the model.

ShellSections

Starts a command block for creation of shell sections.

Syntax:

```
ShellSections     N
Name   ID         data
```

- N: number of shell sections
- Name: current shell section name
- ID: current shell section identification number
- data: current shell section data (depends on shell section resources and requirements)

Example:

```
ShellSections     1
Homogeneous  1         Thickness        0.25
```

Additional information:

Each shell section is defined by a specific keyword followed by the shell section identification number (must be an ascending sequence starting from number one) and additional data. Each shell section available and its input data is explained next.

Shell sections are to be used as parameters for the element Shell_1.

Homogeneous

Creates a homogeneous shell section.

Syntax:

```
Homogeneous    SID  Thickness  TV
```

- SID: current shell section identification number
- TV: shell section thickness value

Example:

```
Homogeneous    1    Thickness  0.25
```

Additional information:

It is assumed that the shell surface is located on the middle of the thickness height.

Composite

Creates a composite shell section.

Syntax:

```
Composite      SID      Laminas  LN
//Material ID  Thickness         Orientation
table data
```

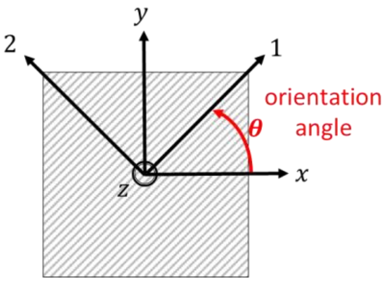
- SID: current shell section identification number
- LN: number of laminas that compose the composite shell section
- table data: table containing each lamina information
  column 1: material identification number
  column 2: lamina thickness
  column 3: lamina principal angle in degrees with respect to the local coordinate system

Example:

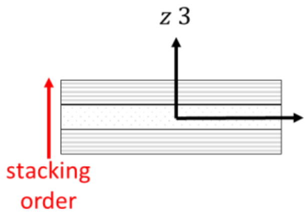
```
Composite   1      Laminas 3
1     0.006  0
2     0.012  90
1     0.006  0
```

Additional information:

It is assumed that the shell surface is located on the middle of the total thickness height. The principal angle orientation of each lamina with respect to the local coordinate system and the stacking order are defined as seen in Figure 16.



(a) Principal angle orientation



(b) Laminas stacking order

Figure 16 – Composite shell section orientations

RigidBodyData

Starts a command block for creation of rigid body data, to be used together with RigidBody_1 element type.

Syntax:

```
RigidBodyData  N
RBData RBID
Mass     MV
J11      J11V     J22      J22V     J33      J33V     J12      J12V     J13      J13V     J23      J23V
Barycenter        XGV      YGV      ZGV
//Optional keywords block below:
CADData                    CADN
```

- N: number of rigid body data
- RBID: current rigid body data identification number
- MV: rigid body mass value
- J11V, J22V, J33V, J12V, J13V and J23V: rigid body inertia tensor components values
- XGV, YGV, and ZGV: coordinates of the barycenter position
- CADN: number of the CAD data ID for post-processing purposes (see CADData keyword)

Example:

```
RigidBodyData  1
RBData 1
Mass     3.63e-5
J11      0.00663           J22      0.00480           J33      0.00663           J12      0.0      J13
         0.0      J23      0.0
Barycenter        0.0      0.0      0.0
CADData           1
```

Additional information:

Each rigid body data is defined by the keyword RBData followed by an identification number (must be an ascending sequence starting from number one).

The mass value has to be provided according to the unit system adopted. In the example, length is defined in millimeters, force in Newton and time in seconds, so that mass has to be provided in tonnes. Users are allowed to choose any other consistent unit system for the whole model.

Inertia properties (J11, J22, J33, J12, J13, J23) must be provided with respect to barycentric axes, parallel to de CAD coordinate system. This is illustrated in Figure 17.

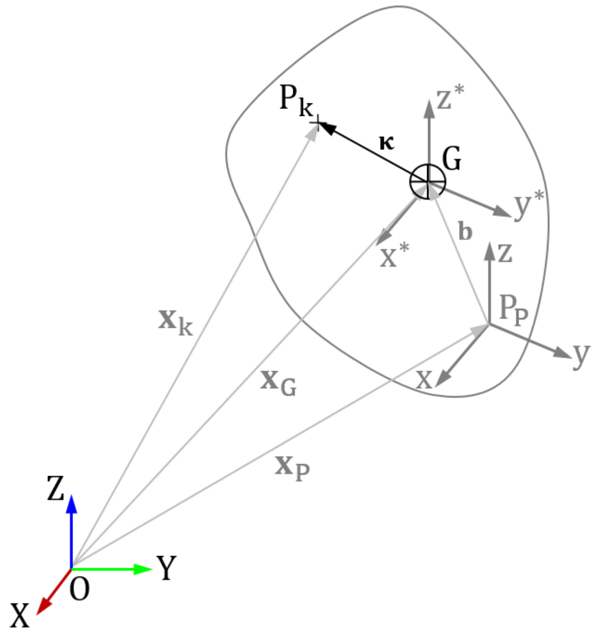


Figure 17 – Inertia properties for Rigid Bodies

In Figure 17 we have the CAD coordinate system (P_P xyz) and the body is already positioned in Giraffe (system OXYZ). Inertia properties should be provided with respect to axes x*, y* and z* illustrated in the same picture, these axes are parallel to x, y and z, but have their origin located at the center of mass of the body (G). Inertia values have units of Mass x Length^2 and they are easily obtained from any 3D CAD software.

It is important to check how your CAD system computes inertia properties[2]. Considering Figure 17, the input necessary to Giraffe could be computed using the following expressions:

$$J_{11} = \int_V \rho(\kappa_2^2 + \kappa_3^2)dV \tag{23}$$

$$J_{22} = \int_V \rho(\kappa_3^2 + \kappa_1^2)dV \tag{24}$$

$$J_{33} = \int_V \rho(\kappa_1^2 + \kappa_2^2)dV \tag{25}$$

$$J_{12} = \int_V \rho(\kappa_1 \kappa_2)dV \tag{26}$$

[2] Some of the main CAD packages provide the inertia tensor as an output, which means that the products of inertia are shown as -J12, -J13 and -J23 (with negative signs). This is not the correct input for Giraffe. Instead of this, Giraffe expects the values directly obtained by the expressions in this page (without changing signs).

$$J_{13} = \int_V \rho(\kappa_1 \kappa_3) dV \tag{27}$$

$$J_{23} = \int_V \rho(\kappa_2 \kappa_3) dV \tag{28}$$

With κ_1, κ_2 and κ_3 being the components of vector $\boldsymbol{\kappa}$ illustrated in Figure 17 and ρ is volumetric mass density function of the material considered for the body.

The barycenter position (G in Figure 17) has to be provided in the CAD coordinate system (O xyz). This information is also easily obtained from the 3D CAD software. The origin of the CAD must coincide with the RigidBody_1 node position.

The last parameter (optional) to define the CADData is the graphic file for postprocessing purposes. It is not used for computing the system physics, which is provided via parameters (mass and inertia tensor entries).

ElementSets

Starts a command block for creation of element sets.

Syntax:

```
ElementSets    N
//Input method 1:
ElementSet    ESID    Elements    NE    List    E1  E2  ...
//Input method 2:
ElementSet    ESID    Elements    NE    Sequence    Initial   EIN   Increment IN
```

- N: number of element sets
- ESID: current element set identification number
- NE: number of elements defined in the current element set
- E1, E2, …, : list with NE element identification numbers
- EIN: initial element identification number
- IN: increment for the element identification number

Example:

```
ElementSets    2
//Input method 1:
ElementSet    1    Elements    3    List    12    27    21
//Input method 2:
ElementSet    2    Elements    4    Sequence    Initial   3   Increment    2
```

Additional information:

Each element set is defined by the keyword ElementSet followed by an identification number (must be an ascending sequence starting from number one).

There are two input methods to define the element sets. The user must choose one of the following options:

- List: it indicates to Giraffe that a list of elements will be provided as input. For example, the ElementSet 1 has three elements which are listed after the keyword List;
- Sequence: it indicates to Giraffe that a sequence of elements will be provided. In the example, ElementSet 2 has 4 elements. The sequence generated automatically will be 3, 5, 7, 9.

NodeSets

Starts a command block for creation of node sets.

Syntax:

```
NodeSets    N
//Input method 1:
NodeSet     NSID    Nodes  NN    List      N1  N2  ...
//Input method 2:
NodeSet     NSID    Nodes  NN    Sequence     Initial   NIN  Increment     IN
```

- N: number of node sets
- NSID: current node set identification number
- NN: number of nodes defined in the current node set
- N1, N2, …: list with NN node identification numbers
- NIN: initial node identification number
- IN: increment for the node identification number

Example:

```
NodeSets    2
//Input method 1:
NodeSet     1       Nodes  3     List      12     27     21
//Input method 2:
NodeSet     2       Nodes  4     Sequence     Initial   3  Increment   2
```

Additional information:

Each node set is defined by the keyword NodeSet followed by an identification number (must be an ascending sequence starting from number one).

There are two input methods to define the node sets. The user must choose one of the following options:

- List: it indicates to Giraffe that a list of nodes will be provided as input. For example, the NodeSet 1 has three nodes which are listed after the keyword List;
- Sequence: it indicates to Giraffe that a sequence of nodes will be provided. In the example, NodeSet 2 has 4 nodes. The sequence generated automatically will be 3, 5, 7, 9.

SurfaceSets

Starts a command block for creation of surface sets.

Syntax:

```
SurfaceSets    N
//Input method 1:
SurfaceSet    SSID    Surfaces    NS    List    S1  S2  …
//Input method 2:
SurfaceSet    SSID    Surfaces    NS    Sequence    Initial  NIS  Increment  IS
```

- N: number of surface sets
- SSID: current surface set identification number
- NS: number of surfaces defined in the current surface set
- S1, S2, …: list with NS surface identification numbers
- NIS: initial surface identification number
- IS: increment for the surface identification number

Example:

```
SurfaceSets    2
//Input method 1:
SurfaceSet    1    Surfaces    3    List    12    27    21
//Input method 2:
SurfaceSet    2    Surfaces    4    Sequence    Initial  3  Increment  2
```

Additional information:

Each surface set is defined by the keyword SurfaceSet followed by an identification number (must be an ascending sequence starting from number one).

There are two input methods to define the surface sets. The user must choose one of the following options:

- List: it indicates to Giraffe that a list of surfaces will be provided as input. For example, the SurfaceSet 1 has three surfaces which are listed after the keyword List;
- Sequence: it indicates to Giraffe that a sequence of surfaces will be provided. In the example, SurfaceSet 2 has 4 surfaces. The sequence generated automatically will be 3, 5, 7, 9.

BoolTable

Creates a Boolean table to rule the behavior of some loads, displacements, contacts and other resources along solution steps. This keyword is used as parameter for many Giraffe resources.

Syntax:

```
BoolTable    B1    B2    B3    ...    BN
```

- B1, B2, B3, ..., BN: sequence of 1 or 0 values composing a Boolean table

Example:

```
BoolTable    1 0 0 1
```

Additional information:

Bootable command is used to provide to Giraffe a sequence of numbers 1 or 0. It is used to define if a given resource is active (1) or inactive (0) in a sequence of solution steps. It can be used with many Giraffe resources, such as environment data, constraints data, contact and special constraints.

As example, in a scenario of 3 sequential solution steps requested by the user, one may be interested in including environment data to define gravity field and also define some constraints, as depicted below:

```
Constraints    1
NodalConstraint        1    NodeSet    1
        UX        BoolTable    1 1 1
        UY        BoolTable    0 1 1
        UZ        BoolTable    1 0 0

Environment
GravityData
G    0    0    -9.81    BoolTable    0 1 1
```

Giraffe would interpret nodal constraints as:

- UX constraint would be considered during the first, second and third solution steps;
- UY constraint would be considered during the second and third solution steps, but would be disregarded during first solution step;
- UZ constraint would be considered only during the first solution step and disregarded during second and third solution steps.

Gravity data would be interpreted to be not applied during the first solution step, but to be applied during the second solution step by a linear ramp function along time evolution. During the third solution step, gravity data would be kept.

Note that BoolTable can be defined with less data than the number of solution steps requested by the user. In this case, Giraffe uses the last provided value as the same for subsequent undefined data. For example, Giraffe interprets "BoolTable 1 1 1" data as the same as "BoolTable 1 1" and also "BoolTable 1". Thus, if the user wants to include some resource using BoolTable from beginning and just keeping it defined along arbitrary solution sequence, it is enough to provide simply: "BoolTable 1". Another example: "BoolTable 1 0 0" data is interpreted as the same as "BoolTable 1 0".

Gravity and ocean data employs BoolTable, as provided in the example. Always the insertion of such loads within a given solution step is done by a linear ramp function along time evolution, both increasing or decreasing the load. For example, if one defines "BoolTable 0 1 0" for GravityData, during the first solution step gravity loads would not be applied, during the second solution step gravity data would be included in the model (by a linear ramp increasing function along time) and during the third solution step gravity data would be, again, switched off (by a linear ramp decreasing function along time).

When using BoolTable with another resources, such as contact or special constraints, the interpretation is straightforward, as one may see in following example:

```
SameDisplacement    1    Nodes  1    2    BoolTable    1 0 1
```

The SameDisplacement special constraint would be considered during the first and third solution steps, but not during the second solution step. Thus, BoolTable permits to switch on/off constraints, special constraints and contacts along solution, when changing between solution steps.

Environment

Creates environment data.

Syntax:

```
Environment
//Optional block to define gravity data:
GravityData
G         GXV     GYV     GZV     BoolTable       BDG
//Optional block to define ocean data:
OceanData
RhoFluid        RV      SurfacePosition         XSV     YSV     ZSV
SeaCurrent      N       NSC     BoolTable       BDO
Depth   DV      Speed   SP      Angle   AV
```

- GXV, GYV and GZV: components of gravity field vector
- BDG: bool table data for gravity loads (see BoolTable)
- RV: specific mass of ocean water
- XSV, YSV and ZSV: coordinates of an arbitrary point located on ocean surface
- NSC: number of points employed to define the sea current velocity field
- BDO: bool table data for sea current velocity field (see BoolTable)

  To define each depth data for sea current:
- DV: depth value
- SP: water speed value
- AV: water speed azimuth angle orientation value (in degrees)

Example:

```
Environment
//Optional block to define gravity data:
GravityData
G         0       0       -9.81   BoolTable       1
//Optional block to define ocean data:
OceanData
RhoFluid        1024    SurfacePosition         0       0       1200
SeaCurrent      N       5       BoolTable       0 0 1
Depth   0       Speed   1.3     Angle   30
Depth   100     Speed   1.2     Angle   10
Depth   250     Speed   0.5     Angle   -30
Depth   500     Speed   0.3     Angle   -20
Depth   1200    Speed   0.1     Angle   45
```

Additional information:

The Environment keyword is used to define environment data, which is given in blocks: GravityData and OceanData. Each block may be defined in arbitrary sequence after Environment keyword. The presence of all blocks in not mandatory. For example, it is possible to define only the GravityData block if one wants only the effect of gravity in the model. For OceanData block, the sea current is defined using a table with N points, which has to be assigned. OceanData employs buoyancy effect when the element is inside the water (applicable for Beam_1, Pipe_1 and Truss_1 elements)

Both GravityData and OceanData are considered in the simulation according to the BoolTable data (see BoolTable).

Loads

Starts a command block for creation of loads.

Syntax:

```
Loads   N
Name    ID      data
```

- N: number of loads
- Name: current load name
- ID: current load identification number
- data: current load data (depends on load resources and requirements)

Example:

```
Loads   1
NodalLoad   1   NodeSet   2   CS   1   NTimes   2
//Time FX FY FZ MX MY MZ
0   0      0   0   0   0   0
1   1000   0   0   0   0   0
```

Additional information:

Each load is defined by a specific keyword followed by the load identification number (must be an ascending sequence starting from number one) and additional data. Each load available and its input data is explained next.

NodalLoad

Creates a nodal load.

Syntax:

```
NodalLoad    ID     NodeSet     NSID   CS     CSID   NTimes   N
//Time FX FY FZ MX MY MZ
table data
```

- ID: current load identification number
- NSID: node set identification number
- CSID: coordinate system identification number
- N: number of lines to be input in table data
- table data: table of nodal loads data following the rule:
  column 1: time
  columns 2-4: components of force vector
  columns 5-7: components of the moment vector

Example:

```
NodalLoad    1      NodeSet     1      CS     1      NTimes   2
//Time FX FY FZ MX MY MZ
0      0      0      0      0      0      0
1      1000   0      0      0      0      0
```

Additional information:

A nodal load employs a table to define a time-varying force and a time-varying moment. Both are applied to the nodes of the node set. Force and moment components are to be defined using any defined coordinate system CSID. The values of forces and moments components are divided between the nodes of the node set and each node receives the same amount of force/moment, such that the total magnitude of force and moment fulfills the user input.

Note: Division of forces/moments on nodes of the node set is done checking if the DOFs are active /inactive in each node. Thus, for Shell_1 element, for example, only mid-side nodes receive moment loads. Corners have only translational DOFs active. This checking is automatically done by Giraffe.

Note: Giraffe performs a linear interpolation between data provided as a tabular time-series. In case of a need of data outside the defined time-range in a table, Giraffe considers: (i) the initially defined value as constant for time-values prior to it and (ii) the lastly defined value as constant for time-values after it.

NodalFollowerLoad

Creates a nodal follower load.

Syntax:

```
NodalFollowerLoad   ID   NodeSet   NSID   CS   CSID   NTimes   N
//Time FX FY FZ MX MY MZ
table data
```

- ID: current load identification number
- NSID: node set identification number
- CSID: coordinate system identification number
- N: number of lines to be input in table data
- table data: table of nodal loads data following the rule:
  column 1: time
  columns 2-4: components of force vector
  columns 5-7: components of the moment vector

Example:

```
NodalFollowerLoad   1   NodeSet   1   CS   1   NTimes   2
//Time FX FY FZ MX MY MZ
0   0      0   0   0   0   0
1   1000   0   0   0   0   0
```

Additional information:

A nodal follower load employs a table to define a time-varying force and a time-varying moment. Both are applied to the nodes of the node set. Force and moment components are to be defined using any defined coordinate system CSID. The values of forces and moments components are divided between the nodes of the node set and each node receives the same amount of force/moment, such that the total magnitude of force and moment fulfills the user input.

Note: Division of forces/moments on nodes of the node set is done checking if the DOFs are active /inactive in each node. Thus, for Shell_1 element, for example, only mid-side nodes receive moment loads. Corners have only translational DOFs active. This checking is automatically done by Giraffe.

The components of force/moment are kept in a local coordinate system that follows the rotations of each node. Then, the follower load updates according to the movement experienced by the node.

Note: Giraffe performs a linear interpolation between data provided as a tabular time-series. In case of a need of data outside the defined time-range in a table, Giraffe considers: (i) the initially defined value as constant for time-values prior to it and (ii) the lastly defined value as constant for time-values after it.

PipeLoad

Creates a pipe load (to be used together with Pipe_1 elements).

Syntax:

```
PipeLoad        ID        ElementSet      ESID      NTimes  N
//Time  POI  POE  RhoI  RhoE
table data
```

- ID: current load identification number
- ESID: element set identification number
- N: number of lines to be input in table data
- table data: table of pipe loads data following the rule:
  column 1: time
  column 2: internal pressure of the pipe
  column 3: external pressure of the pipe
  column 4: internal fluid specific mass (currently not used by Giraffe)
  column 5: external fluid specific mass (currently not used by Giraffe)

Example:

```
PipeLoad        1         ElementSet      1         NTimes 2
//Time POI      POE       RhoI RhoE
2        0        0        0        0
3        1200000           0        0        0
```

Additional information:

A pipe load employs a table to define time-varying internal/external pressures in a pipe. Both are applied to each element of the element set. All elements of the element set must be of the type Pipe_1.

Note: Giraffe performs a linear interpolation between data provided as a tabular time-series. In case of a need of data outside the defined time-range in a table, Giraffe considers: (i) the initially defined value as constant for time-values prior to it and (ii) the lastly defined value as constant for time-values after it.

ShellLoad

Creates a shell load (to be used together with Shell_1 elements).

Syntax:

```
ShellLoad    ID    ElementSet    ESID    AreaUpdate    AUB    NTimes N
//Time P
table data
```

- ID: current load identification number
- ESID: element set identification number
- AUB: Boolean variable to define if the area is to be updated for recalculation of the resultant due to pressure integration (1) or not (0)
- N: number of lines to be input in table data
- table data: table of shell loads data following the rule:
  column 1: time
  column 2: pressure applied on shell element surface

Example:

```
ShellLoad    1    ElementSet    1    AreaUpdate    1    NTimes 2
//Time P
0            0.0
2            +10.0
```

Additional information:

A shell load employs a table to define a time-varying pressure in a shell element. It is applied to each element of the element set. All elements of the element set must be of the type Shell_1. A positive value of pressure acts opposite to the external normal direction of the element surface, which is associated with the sequence of nodes numbering in the element. The AreaUpdate keyword establishes if the resultant of pressure integrated along the element area should consider or not the area update, according to element deformations.

Note: Giraffe performs a linear interpolation between data provided as a tabular time-series. In case of a need of data outside the defined time-range in a table, Giraffe considers: (i) the initially defined value as constant for time-values prior to it and (ii) the lastly defined value as constant for time-values after it.

Displacements

Starts a command block for creation of displacements (to be prescribed).

Syntax:

```
Displacements  N
Name   ID        data
```

- N: number of displacements
- Name: current displacement name
- ID: current displacement identification number
- data: current displacement data (depends on displacement resources and requirements)

Example:

```
Displacements  1
NodalDisplacement    1    NodeSet    1    CS    1    NTimes 3
//Time UX UY UZ ROTX ROTY ROTZ
0    0    0       0    0    0    0
2    0    1.25    0    0    0    0
3    0    0       0    0    0    6.28
```

Additional information:

Each displacement is defined by a specific keyword followed by the displacement identification number (must be an ascending sequence starting from number one) and additional data. Each displacement available and its input data is explained next.

Note: displacement prescription in Giraffe has to be carefully carried out by the user. In case of simultaneous prescription of displacements/rotations to a given node, the last-defined value only will be considered, following the sequence of displacements input data.

NodalDisplacement

Creates a nodal displacement (to be prescribed).

Syntax:

```
NodalDisplacement    ID    NodeSet    NSID    CS    CSID    NTimes    N
//Time UX UY UZ ROTX ROTY ROTZ
table data
```

- ID: current displacement identification number
- NSID: node set identification number
- CSID: coordinate system identification number
- N: number of lines to be input in table data
- table data: table of nodal displacement data following the rule:
  column 1: time
  columns 2-4: components of displacement vector
  columns 5-7: components of the rotation vector (Euler rotation vector)

Example:

```
NodalDisplacement    1    NodeSet    1    CS    1    NTimes    3
//Time UX UY UZ ROTX ROTY ROTZ
0    0    0       0    0    0    0
2    0    1.25    0    0    0    0
3    0    0       0    0    0    6.28
```

Additional information:

A nodal displacement employs a table to define a time-varying displacement and a time-varying rotation vector. Both are applied (prescribed) to each node of the node set. Displacement and rotation vector components have to be defined using the desirable coordinate system CSID. The rotation input is made in **radians** by a Euler rotation vector.

Displacement/rotation imposition is done in an incremental way. Thus, each time-step of solution will prescribe an increment of displacement/rotation, following the table of Nodal displacement entry. Giraffe picks in nodal displacement table the difference between the displacement/rotation value at the beginning/end of the time-step. This increment is prescribed within the time-step. The procedure continues until the end of the simulation. Note that the value of prescribed displacement/rotation in the table is not necessarily imposed during the simulation. It depends on how the user sets constraints on nodes. To prescribe displacements/rotations the user has to define both displacements and constraints inputs, in order to choose if each DOF is free or fixed (which may vary along solution sequence). If the DOF is free (default), the corresponding nodal displacement is ignored.

Note: Giraffe performs a linear interpolation between data provided as a tabular time-series. In case of a need of data outside the defined time-range in a table, Giraffe considers: (i) the initially defined value as constant for time-values prior to it and (ii) the lastly defined value as constant for time-values after it.

DisplacementField

Creates a linear time-varying displacement field at nodes during a given solution step.

Syntax:

```
DisplacementField     ID      NNodes        NN      CS CSID SolutionStep    SS
//Node UX       UY      UZ      ROTX    ROTY    ROTZ
table data
```

- ID: current displacement identification number
- NN: number of nodes that will have an assigned displacement data
- CSID: coordinate system identification number
- SS: solution step number associated with the displacement field prescription
- table data: table of nodal displacement data following the rule:
  column 1: node
  columns 2-4: components of displacement vector
  columns 5-7: components of the rotation vector (Euler rotation vector)

Example:

```
DisplacementField     1       NNodes        5       CS 1    SolutionStep    2
//Node UX       UY      UZ      ROTX    ROTY    ROTZ
1       0       1.0     0       0       0       0
2       0       0.5     0       0       0       0
3       0       0.0     0       0       0       0
6       0       -0.5    0       0       0       0
7       0       -1.0    0       0       0       0
```

Additional information:

A nodal displacement field employs a table to define a field of generally distinct linear time-varying displacement/rotation for a set of nodes. All nodes assigned have independent displacement/rotation. Displacement and rotation vector components have to be defined using the desirable coordinate system CSID. The rotation input is made in **radians** by a Euler rotation vector.

Displacement/rotation prescription is done in an incremental way. Thus, at each time-step of solution an increment of displacement/rotation is prescribed following a linear increment along time, such that along the solution step chosen the whole amount of displacement/rotation is prescribed.

Note that the value of prescribed displacement/rotation in the table is not necessarily imposed during the simulation. It depends on how the user sets constraints on nodes. To prescribe displacements/rotations the user has to define both displacements and constraints inputs, in order to choose if each DOF is free or fixed (which may vary along solution sequence). If the DOF is free (default), the corresponding nodal displacement is ignored.

Constraints

Starts a command block for creation of constraints.

Syntax:

```
Constraints     N
Name   ID       data
```

- N: number of constraints
- Name: current constraint name
- ID: current constraint identification number
- data: current constraint data (depends on constraint resources and requirements)

Example:

```
Constraints     2
NodalConstraint         1       NodeSet         1
        UX      BoolTable       1
        UY      BoolTable       1
        UZ      BoolTable       1
        ROTX    BoolTable       1 1
        ROTY    BoolTable       1 1 0 1
        ROTZ    BoolTable       1
NodalConstraint         2       NodeSet         2
        UX      BoolTable       0
        UY      BoolTable       0 0
        UZ      BoolTable       0 0
        ROTX    BoolTable       0 0
        ROTY    BoolTable       0 0
        ROTZ    BoolTable       0 0
```

Additional information:

Each constraint is defined by a specific keyword followed by the constraint identification number (must be an ascending sequence starting from number one) and additional data. Each constraint available and its input data is explained next.

Constraints imposition follow the increasing sequence defined in the input file. In case the user establishes conflicting input data, the last defined will be assumed by Giraffe.

NodalConstraint

Creates a nodal constraint.

Syntax:

```
NodalConstraint     ID     NodeSet     NSID
constraint data
```

- ID: current constraint identification number
- NSID: node set identification number
- constraint data: specific keywords UX, UY, UZ, ROTX, ROTY and ROTZ, each followed by a BoolTable keyword to define nodal constraint solution sequence data

Example:

```
Constraints     2
NodalConstraint     1     NodeSet     1
        UX      BoolTable     1
        UY      BoolTable     1
        UZ      BoolTable     1
        ROTX    BoolTable     1 1
        ROTY    BoolTable     1 1 0 1
        ROTZ    BoolTable     1
NodalConstraint     2     NodeSet     2
        UX      BoolTable     1
        UY      BoolTable     0 1
        ROTZ    BoolTable     1 0
```

Additional information:

A nodal constraint is employed to define fixed DOFs in the model. When establishing a mesh, all DOFs are free. To establish Dirichlet boundary conditions in model regions, it is necessary to establish node sets and, on that regions, apply desirable constraints. For that, the user may separately operate on distinct DOFs of each node, within a chosen node set. These are:

- UX: displacement in global X direction
- UY: displacement in global Y direction
- UZ: displacement in global Z direction
- ROTX: rotation in global X direction
- ROTY: rotation in global Y direction
- ROTZ: rotation in global Z direction

The BoolTable resource is used to establish the behavior of each DOF along solution evolution (see BoolTable), that is, if the constraint is turned on/off. This permits to establish scenarios of alternating constraints along solution evolution.

DOFs with constraints turned on are fixed. In case of definition of displacements associated with such nodes, these will be prescribed along solution evolution, following the established table of displacement along time, for each DOF (see Displacements). Otherwise, Giraffe will simply consider a zero-displacement value.

SpecialConstraints

Starts a command block for creation of special constraints.

Syntax:

```
SpecialConstraints      N
Name   ID       data
```

- N: number of special constraints
- Name: current special constraint name
- ID: current special constraint identification number
- data: current special constraint data (depends on special constraint resources and requirements)

Example:

```
SpecialConstraints      1
SameDisplacement        1       Nodes  1        2       BoolTable  1
```

Additional information:

Each special constraint is defined by a specific keyword followed by the special constraint identification number (must be an ascending sequence starting from number one) and additional data. Each special constraint available and its input data is explained next. Models of special constraints are presented in [6] and [7].

Differently from Constraints keyword, SpecialConstraints are established keeping original system DOFs and including additional unknowns to enforce desired constraints (Lagrange Multipliers). Thus, inclusion of special constraints in the model will increase the number of unknows.

SameDisplacement

Creates a "same displacement" special constraint.

Syntax:

SameDisplacement	SCID	Nodes	ID1	ID2	BoolTable	BDSC

- SCID: current special constraint identification number
- ID1 and ID2: nodes identification numbers
- BDSC: BoolTable data for current special constraint

Example:

SameDisplacement	1	Nodes	1	2	BoolTable	1

Additional information:

This constraint is used to enforce that the two selected nodes will present the same displacements (but not necessarily the same rotation). It can be used to represent a spherical joint in a mechanism, for example. The selected nodes should be initially coincident for that purpose.

BoolTable keyword is optional. It permits creating a scenario in which the special constraint is turned on/off along solution steps (see BoolTable). If the user does not include BoolTable, Giraffe assumes that the special constraint will be considered for all solution steps as turned on.

Remark: if this special constraint is used in dynamic simulations, velocity initial conditions are to be set only to the first node. If one sets different velocity initial conditions for both nodes, the first node velocity initial conditions are considered for both nodes. The second node velocity initial conditions are ignored.

SameRotation

Creates a "same rotation" special constraint.

Syntax:

SameRotation	SCID	Nodes	ID1 ID2	BoolTable	BDSC

- SCID: current special constraint identification number
- ID1 and ID2: nodes identification numbers
- BDSC: BoolTable data for current special constraint

Example:

SameRotation	1	Nodes	1	2	BoolTable	1

Additional information:

This constraint is used to enforce that the two selected nodes will present the same rotation (but not necessarily the same displacement).

BoolTable keyword is optional. It permits creating a scenario in which the special constraint is turned on/off along solution steps (see BoolTable). If the user does not include BoolTable, Giraffe assumes that the special constraint will be considered for all solution steps as turned on.

Remark: if this special constraint is used in dynamic simulations, angular velocity initial conditions are to be set only to the first node. If one sets different angular velocity initial conditions for both nodes, the first node angular velocity initial conditions are considered for both nodes. The second node angular initial velocity conditions are ignored.

RigidNodeSet

Creates a "rigid node set" special constraint.

Syntax:

RigidNodeSet	SCID	PilotNode	PID	NodeSet	NSID	BoolTable	BDSC

- SCID: current special constraint identification number
- PID: pilot node identification number
- NSID: node set identification number
- BDSC: BoolTable data for current special constraint

Example:

RigidNodeSet	1	PilotNode	1	NodeSet	1	BoolTable	1

Additional information:

This constraint is used to establish a rigid region, which is formed by the connection of all the nodes in the selected node set. The pilot node will always present 6 degrees of freedom, which will rule the movement of all the nodes in the node set. The pilot node can be a node inside the node set or an independent node.

BoolTable keyword is optional. It permits creating a scenario in which the special constraint is turned on/off along solution steps (see BoolTable). If the user does not include BoolTable, Giraffe assumes that the special constraint will be considered for all solution steps as turned on.

This special constraint imposes that general rigid body movement may play a role for the set of nodes assigned. Then, one can handle large displacements and large rotations with no kinematic limitations.

Remark: if this special constraint is used in dynamic simulations, displacement and rotation initial conditions are to be set only for the pilot node. Giraffe automatically evaluates, using rigid body's equations, the proper initial conditions for each node of the node set. If one sets arbitrary initial conditions for the nodes, which are not compatible to pilot node's conditions, these are ignored.

HingeJoint

Creates a "hinge joint" special constraint.

Syntax:

```
HingeJoint  SCID  Nodes  ID1    ID2    CS    CSID    LinearStiffness    LSV    LinearDamping    LDV
QuadraticDamping  QDV  BoolTable      BDSC
```

- SCID: current special constraint identification number
- ID1 and ID2: nodes identification numbers
- CSID: coordinate system identification number
- LSV: linear stiffness coefficient value
- LDV: linear damping coefficient value
- QDV: quadratic damping coefficient value
- BDSC: BoolTable data for current special constraint

Example:

```
HingeJoint      1       Nodes  1       2       CS      1       LinearStiffness  0.0
        LinearDamping 0.0      QuadraticDamping       0.0      BoolTable        1
```

Additional information:

This constraint is used to represent a hinge joint. It enforces that the two selected nodes will present the same displacements, as in SameDisplacement constraint. Furthermore, the direction $\mathbf{e}_3$ from the chosen coordinate system CS will represent the direction of free relative rotation between the chosen nodes (hinge axis). The direction $\mathbf{e}_{3_\mathrm{A}}$ is updated, since it is attached to the rotation of the node A (first node defined for the hinge joint). The directions of $\mathbf{e}_{1_\mathrm{B}}$ and $\mathbf{e}_{2_\mathrm{B}}$ are also updated, following the node B rotations (second node defined for the hinge joint).

At the beginning, we assume that both nodes coordinate systems lie at the same directions. During the simulation, the hinge joint ensures that $\mathbf{e}_{3_\mathrm{A}} \equiv \mathbf{e}_{3_\mathrm{B}}$. For that, two constraints r_1 and r_2 for rotations are enforce by:

$$\mathrm{r}_1 = \mathbf{e}_{3_\mathrm{A}} \cdot \mathbf{e}_{1_\mathrm{B}} = 0 \quad (29)$$

$$\mathrm{r}_2 = \mathbf{e}_{3_\mathrm{A}} \cdot \mathbf{e}_{2_\mathrm{B}} = 0 \quad (30)$$

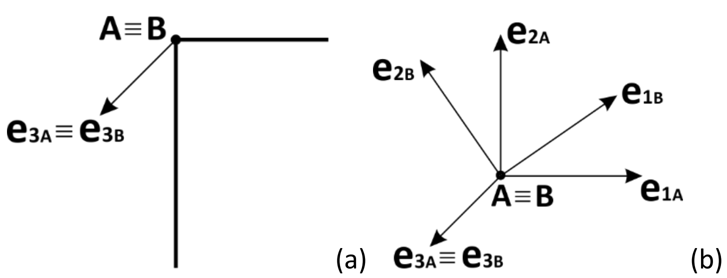


Figure 18 – (a) Example of a hinge joint between two beams. (b) The coordinate systems of nodes A and B after nodes rotation.

Figure 18(a) shows an example of hinge joint, located at nodes A and B (coincident). The coordinate systems of both nodes is initially the same, and defined by the keyword CS. After some movement, the rotation at nodes A and B may differ, such that the systems (which follows nodes rotations) are as in Figure 18(b). Note that the constraints (29) and (30) are obeyed in such transformation.

The parameter LinearStiffness permits entering a stiffness coefficient, such that the hinge presents a torsion stiffness. The moment $\mathrm{M_{spring}}$ generated by the torsion stiffness spring is given by:

$$\mathrm{M_{spring}} = \mathrm{K_\theta}\theta \quad (31)$$

where $\mathrm{K_\theta}$ is the linear stiffness coefficient and θ is the accumulated angle of relative rotation around the hinge direction. The variable θ may represent finite rotations, involving many turns around the hinge axis.

Analogously, one can enter linear and quadratic damping coefficients by the keywords LinearDamping and QuadraticDamping. Then, the moment $\mathrm{M_{damper}}$ can be evaluated by:

$$\mathrm{M_{damper}} = \mathrm{C_{1\theta}}(\boldsymbol{\omega}_\mathrm{A} - \boldsymbol{\omega}_\mathrm{B}) \cdot \mathbf{e}_{3\mathrm{A}} + \mathrm{C_{2\theta}}\|(\boldsymbol{\omega}_\mathrm{A} - \boldsymbol{\omega}_\mathrm{B})\|(\boldsymbol{\omega}_\mathrm{A} - \boldsymbol{\omega}_\mathrm{B}) \cdot \mathbf{e}_{3\mathrm{A}} \quad (32)$$

where $\mathrm{C_{1\theta}}$ and $\mathrm{C_{2\theta}}$ are respectively linear and quadratic damping coefficients and $\boldsymbol{\omega}_\mathrm{A}$ and $\boldsymbol{\omega}_\mathrm{B}$ are the instantaneous angular velocities of nodes A and B, respectively.

Remark: If one sets different velocity initial conditions for both nodes, the first node velocity initial conditions are considered for both nodes. The second node velocity initial conditions are, then, ignored. Furthermore, angular velocity conditions may have independent value for both nodes. However, their compatibility is done by:

1) Giraffe evaluates the direction of the hinge axis $\mathbf{e}_{3\mathrm{A}}$ and projects the angular velocity initial condition of both nodes on this direction;
2) The components that lie in the direction of the hinge axis $\mathbf{e}_{3\mathrm{A}}$, are independent in both nodes;

The components that lie in direction orthogonal to the hinge axis are set, taking the first node values and copying then to the second one's, which has its values ignored.

BoolTable keyword is optional. It permits creating a scenario in which the special constraint is turned on/off along solution steps (see BoolTable). If the user does not include BoolTable, Giraffe assumes that the special constraint will be considered for all solution steps as turned on.

UniversalJoint

Creates a "universal joint" (Cardan) special constraint.

Syntax:

```
UniversalJoint  SCID    Nodes  ID1    ID2    CSA    CSAID  CSB    CSBID  BoolTable
        BDSC
```

- SCID: current special constraint identification number
- ID1 and ID2: nodes identification numbers
- CSAID and CSBID: coordinate systems identification numbers
- BDSC: BoolTable data for current special constraint

Example:

```
UniversalJoint  1       Nodes  1      2      CSA    1      CSB    2      BoolTable  1
```

Additional information:

This constraint is used to represent a universal (cardan) joint. It enforces that the two selected nodes will present the same displacements, as in SameDisplacement constraint. The selected nodes should be coincident for that purpose. Additionally, two coordinate systems are defined, associated with the movement of the first and the second defined nodes (A and B, respectively). The directions $\mathbf{e}_{3_A}$ and $\mathbf{e}_{3_B}$ should be used to represent the directions of the axles that are connected using the joint. Both coordinate systems CSA and CSB directions are updated during simulation, according to the rotations experienced by nodes A and B. The constraint imposed to the represent the rotation transmission of cardan joint is:

$$r_3 = \mathbf{e}_{1_A} \cdot \mathbf{e}_{2_B} = 0 \tag{33}$$

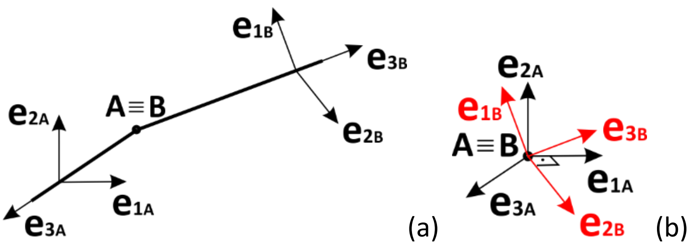


Figure 19 – (a) Example of a universal joint between two beams. (b) The coordinate systems of nodes A and B.

Figure 19(a) shows an example of cardan joint, located at nodes A and B (coincident). The coordinate systems of both nodes are different and **must obey** at the beginning of the simulation the constraint (33), otherwise an error message will be shown in Giraffe output window.

Remark: If one sets different velocity initial conditions for both nodes, the first node velocity initial conditions are considered for both nodes. The second node velocity initial conditions are ignored. Furthermore, angular velocity conditions may have independent value for both nodes. However, their compatibility is done by:

1) Giraffe evaluates the directions $\mathbf{e}_{3_A}$ and $\mathbf{e}_{3_B}$, and projects the angular velocity initial condition of both nodes to respective directions;
2) The component of angular velocity of node A that lies in direction $\mathbf{e}_{3_A}$ is also imposed to direction $\mathbf{e}_{3_B}$;
3) The components of angular velocity in other directions are independent.

BoolTable keyword is optional. It permits creating a scenario in which the special constraint is turned on/off along solution steps (see BoolTable). If the user does not include BoolTable, Giraffe assumes that the special constraint will be considered for all solution steps as turned on.

TranslationalJoint

Creates a "translational joint" special constraint.

Syntax:

TranslationalJoint	SCID	Nodes	ID1 ID2 RotationNode	ID3	CS	CSID
BoolTable	BDSC					

- SCID: current special constraint identification number
- ID1 and ID2: nodes identification numbers (to be connected by translational joint)
- ID3: rotation node identification number (to rule translational joint direction update)
- CSID: coordinate system identification number
- BDSC: BoolTable data for current special constraint

Example:

TranslationalJoint	1	Nodes	1 2	RotationNode	3	CS	1
BoolTable	1						

Additional information:

This constraint is used to represent a translational joint between the first and second chosen nodes (nodes A and B, identified by ID1 and ID2). It enforces that the two selected nodes will present relative displacements only along a direction $\mathbf{e_3}$. The direction $\mathbf{e_3}$ is taken from the chosen coordinate system CS and is updated along simulation according to the rotations experienced by the rotation node assigned (ID3). The constraints enforced are the following:

$$\mathrm{r}_1 = \mathbf{e_1} \cdot (\mathbf{u_A} - \mathbf{u_B}) = 0 \quad (34)$$

$$\mathrm{r}_2 = \mathbf{e_2} \cdot (\mathbf{u_A} - \mathbf{u_B}) = 0 \quad (35)$$

This type of joint is very useful for establishing suspension systems, as in the example:

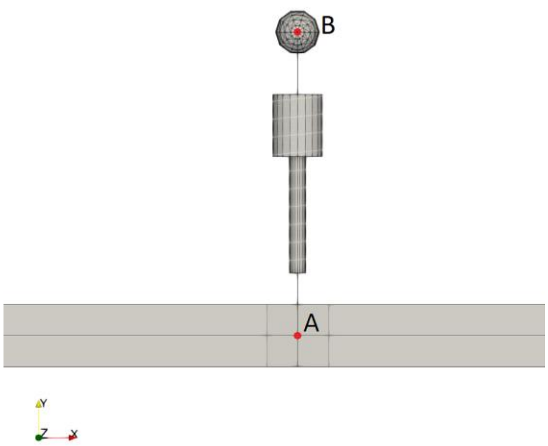


Figure 20 – Example of a suspension system

In this example a spring/dashpot element is defined between nodes A and B. Also, one has a mass element defined at node B, while node A is embedded in frame structure, as a part meshed using beam elements. In order to avoid undesirable rotations of the spring/dashpot one may establish a translational joint constraint between nodes A and B, thus considering gobal direction Y as the initial direction for $\mathbf{e_3}$. In this case node A should be used as RotationNode. With that, only relative displacements along current direction $\mathbf{e_3}$ are permitted and the suspension system may behave as desirable. Nodes A and B may translate or rotate with the frame structure, as direction $\mathbf{e_3}$ is updated, accordingly.

BoolTable keyword is optional. It permits creating a scenario in which the special constraint is turned on/off along solution steps (see BoolTable). If the user does not include BoolTable, Giraffe assumes that the special constraint will be considered for all solution steps as turned on.

Contacts

Starts a command block for creation of contact constraints.

Syntax:

Contacts		N
Name	ID	data

- N: number of contacts
- Name: current contact name
- ID: current contact identification number
- data: current contact data (depends on contact resources and requirements)

Example:

```
Contacts        2
NSSS    1       NodeSet         1       SurfaceSet      1       MU      0.0     EPN     1e8
        CN      0.0     EPT     1e7     CT      0.0     Pinball 1000    Radius  0.0
        MaxPointwiseInt         1   BoolTable 1
SSSS    2       SurfaceSet1     1       SurfaceSet2     2       MU      0.0     EPN     1e8
        CN      0.0     EPT     1e7     CT      0.0     Pinball 1000  BoolTable 1
```

Additional information:

Each contact is defined by a specific keyword followed by the contact identification number (must be an ascending sequence starting from number one) and additional data. Each contact available and its input data is explained next.

Details about Giraffe contact formulations can be found in papers: [8] [9] [10] [11] [12] [13]. A complete explanation on most methods implemented in Giraffe may be found in [14].

NSSS

Creates a contact constraint for the interaction between a node set and a surface set (NSSS).

Syntax:

NSSS	CID	NodeSet		NSID	SurfaceSet		SSID	MU	MUV	EPN	EPNV
	CN	CNV	EPT	EPTV	CT	CTV	Pinball	PV	Radius	RV	
MaxPointwiseInt			NP	BoolTable BTC							

- CID: current contact constraint identification number
- NSID: node set identification number
- SSID: surface set identification number
- MUV: coefficient of friction value
- EPNV: penalty coefficient to enforce normal contact constraint (no penetration)
- CNV: normal damping parameter coefficient
- EPTV: penalty coefficient to enforce tangential contact constraint (sticking condition)
- CTV: tangential damping parameter coefficient
- PV: pinball radius value (contact rough searching)
- RV: sphere radius value surrounding each node in the node set
- NP: maximum number of contact pointwise interactions between each sphere and surface
- BTC: bool table data for current contact constraint (see BoolTable)

Example:

NSSS	1	NodeSet		1	SurfaceSet		1	MU	0.0	EPN	1e8
	CN	0.0	EPT	1e7	CT	0.0	Pinball	1000	Radius	0.0	
MaxPointwiseInt			1	BoolTable 1							

Additional information:

This contact formulation uses developments detailed in [12]. It is an enhanced master-slave, which considers a spherical surface around each node defined in the chosen node set (sphere). Each sphere interacts with surfaces in the chosen surface set, in case of contact occurrence.

Constraints enforcements are done by penalty method. Thus, it is necessary for the user to input penalty parameters data. Usually these may be calibrated based on physical information related to the desired scenario, basing on equivalent local stiffness, leading to allowable penetration on each contact zone.

Damping coefficients are useful for dissipation of energy during impact simulations, avoiding high frequency oscillations on contact forces.

Pinball radius is a rough search geometrical parameter that is used by Giraffe to establish probable and not probable contact interactions. The larger the penalty parameter, the heavier will be the model, since Giraffe will spend time for a larger number of possible contact interactions. However, small pinball radii lead to loosing contact detection. Thus, it is a compromise between accuracy and solution speed. In case the user is in doubt about this parameter, it is better to test it with high values and, afterwards, decrease it.

Usually only a single pointwise contact interaction is permitted between each sphere and each surface. However, some surfaces have the possibility of seeking for more than one pointwise contact solution (typically on non-convexity scenarios).

BoolTable keyword is optional. It permits creating a scenario in which the contact constraint is turned on/off along solution steps (see BoolTable). If the user does not include BoolTable, Giraffe assumes that the contact constraint will be considered for all solution steps as turned on.

SSSS

Creates a contact constraint for the interaction between two surface sets (SSSS).

Syntax:

```
//Syntax 1 – a single friction coefficient value
SSSS    CID     SurfaceSet1   SS1ID   SurfaceSet2   SS2ID   MU      MUV     EPN     EPNV
        CN      CNV     EPT     EPTV    CT      CTV     Pinball PV      BoolTable BTC
//Syntax 2 – static and dynamic friction coefficient values
SSSS    CID     SurfaceSet1   SS1ID   SurfaceSet2   SS2ID   MUS     MUSV    MUD     MUDV
EPN     EPNV    CN      CNV     EPT     EPTV    CT      CTV     Pinball PV      BoolTable BTC
//Optional keywords:
WriteReport
```

- CID: current contact constraint identification number
- SS1ID: surface set 1 identification number
- SS2ID: surface set 2 identification number
- MUV: coefficient of friction value
- MUSV: static coefficient of friction value
- MUDV: dynamic coefficient of friction value
- EPNV: penalty coefficient to enforce normal contact constraint (no penetration)
- CNV: normal damping parameter coefficient
- EPTV: penalty coefficient to enforce tangential contact constraint (sticking condition)
- CTV: tangential damping parameter coefficient
- PV: pinball radius value (contact rough searching)
- BTC: bool table data for current contact constraint (see BoolTable)
- WriteReport: keyword to instruct Giraffe to produce reports for each local contact problem solved (only use it for debugging purposes because it takes time for writing)

Example:

```
SSSS    1       SurfaceSet1   1       SurfaceSet2   2       MU      0.0     EPN     1e8
        CN      0.0     EPT     1e7     CT      0.0     Pinball 1000    BoolTable 1
```

Additional information:

This contact formulation uses developments detailed in [9] and [10]. It is master-master contact formulation, which considers interaction between surfaces with no election of slave points.

Constraints enforcements are done by penalty method. Thus, it is necessary for the user to input penalty parameters data. Usually these may be calibrated based on physical information related to the desired scenario, basing on equivalent local stiffness, leading to allowable penetration on each contact zone.

Damping coefficients are useful for dissipation of energy during impact simulations, avoiding high frequency oscillations on contact forces.

Pinball radius is a rough search geometrical parameter that is used by Giraffe to establish probable and not probable contact interactions. The larger the penalty parameter, the heavier will be the model, since Giraffe will spend time for a larger number of possible contact interactions. However, small pinball radii lead to loosing contact detection. Thus, it is a compromise between accuracy and solution speed. In case the user is in doubt about this parameter, it is better to test it with high values and, afterwards, decrease it.

When the user performs a degeneration of surfaces involved in the SSSS contact, Giraffe automatically considers all degenerated cases for contact. With that the user may construct a set of curve/surface or point/surface contact pairs automatically within the same contact creation.

BoolTable keyword is optional. It permits creating a scenario in which the contact constraint is turned on/off along solution steps (see BoolTable). If the user does not include BoolTable, Giraffe assumes that the contact constraint will be considered for all solution steps as turned on.

InitialConditions

Starts a command block for creation of initial conditions to be used in a transient dynamic analysis.

Syntax:

```
InitialConditions       N
InitialCondition ICID   Node   NID    DU    DUX    DUY    DUZ    OMEGA    OX
         OY      OZ     SolutionStep  SSID
```

- N: number of initial conditions
- ICID: current initial condition identification number
- NID: node identification number
- DUX, DUY and DUZ: components of velocity vector (on a global coordinate system)
- OX, OY and OZ: components of angular velocity vector (on a global coordinate system)
- SSID: solution step identification number

Example:

```
InitialConditions       1
InitialCondition 1      Node   1      DU    0.0    1.0    0.0    OMEGA    0.0
         0.0     0.0    SolutionStep  1
```

Additional information:

Each initial condition is defined followed by the initial condition identification number (must be an ascending sequence starting from number one) and additional data. All data is interpreted on global coordinate system. The solution step input is necessary to associate the initial condition to a given solution step (dynamic).

Remark: when inserting initial conditions to nodes involved in a special constraint, please note that some of them may be ignored, according to the kind of special constraint.

Points

Starts a command block for creation of points to be used to compound geometric entities (e.g.: surfaces).

Syntax:

```
Points   N
Point    ID    X    Y    Z
```

- N: number of points
- ID: current point identification number
- X: current point X coordinate (on a global coordinate system)
- Y: current point Y coordinate (on a global coordinate system)
- Z: current point Z coordinate (on a global coordinate system)

Example:

```
Points   3
Point    1     1.0    0.0    3.0
Point    2     0.0    2.5    -5.1
Point    3     0.0    0.0    -10.0
```

Additional information:

Each point is defined by the keyword Point followed by the point identification number (must be an ascending sequence starting from number one), coordinates X, Y and Z.

Arcs

Starts a command block for creation of arcs to be used to compound geometric entities (e.g.: extruded or revolved surfaces).

Syntax:

```
Arcs    N
Arc     ID      InitialPoint    XIP             YIP             FinalPoint      XFP     YFP
        CenterPoint     XCP     YCP
```

- N: number of arcs
- ID: current arc identification number
- XIP: X coordinate of the initial point of the arc
- YIP: Y coordinate of the initial point of the arc
- XFP: X coordinate of the final point of the arc
- YFP: Y coordinate of the final point of the arc
- XCP: X coordinate of the center point of the arc
- YCP: Y coordinate of the center point of the arc

Example:

```
Arcs    2
Arc     1       InitialPoint    0.0             -1.0            FinalPoint      0.0
        1.0             CenterPoint     -1.0e5          0.0
Arc     2       InitialPoint    0.024           -0.132          FinalPoint      0.024
0.132           CenterPoint     -0.075          0.0
```

Additional information:

Each arc is defined by the keyword Arc followed by the arc identification number (must be an ascending sequence starting from number one), its initial point, final point and center point.

The arc is supposed to lie on a local XY plane. The coordinate parameters indicating its initial point, end point and center point are understood on a local coordinate system (to be defined by the user). Figure 21 illustrates the arc:

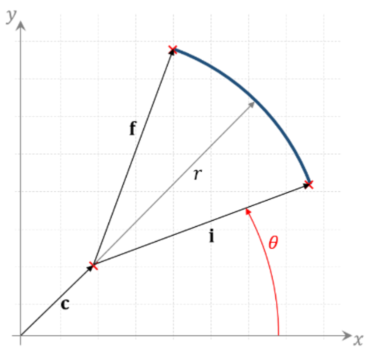


Figure 21 – Arc definition on a local coordinate system (figure from [7])

The arc parameterization $\mathbf{a}(\theta)$ is given by:

$$\mathbf{a}(\theta) = \begin{bmatrix} r\cos\theta + \mathrm{XCP} \\ r\sin\theta + \mathrm{YCP} \\ 0 \end{bmatrix} \tag{36}$$

where r is the arc radius (evaluated by Giraffe automatically). The arc center is given by local coordinates $\mathbf{c} = (\mathrm{XCP}, \mathrm{YCP})$. In Figure 21 one may observe two particular evaluations of parameterizations, for the initial point and for the end point of the arc, given by $\mathbf{i}$ and $\mathbf{f}$. The location of these points in the local coordinate system compose the input data, such that $\mathbf{i} = (\mathrm{XIP}, \mathrm{YIP})$ and $\mathbf{f} = (\mathrm{XFP}, \mathrm{YFP})$.

Surfaces

Starts a command block for creation of surfaces.

Syntax:

Surfaces		N	
Name	ID	data	optional

- N: number of surfaces
- Name: current surface name
- ID: current surface identification number
- data: current surface data (depends on surface resources and requirements)
- optional: optional keywords for degenerating a surface

Example:

Surfaces	2								
RigidTriangularSurface_1		1	Points	1	2	3	PilotNode		1
FlexibleSECylinder_1	2	A	0.1	B	0.1	N	3.0	CS	1
	NormalExterior Nodes	1	2						

Additional information:

Each surface is defined by a specific keyword followed by the surface identification number (must be an ascending sequence starting from number one) and additional data. Each surface available and its input data is explained next.

The user may choose a fixed value for a given convective coordinate or, alternatively, establish a number of divisions to be performed by Giraffe along the valid range of a given convective coordinate, thus establishing a set of fixed values for such coordinate. If this is the choice, Giraffe performs divisions uniformly along that coordinate.

When employing degeneration, the original surface turns into a curve (or a set of curves) or a point (or a set of points). Examples of degenerations are given next:

```
//Degeneration into a single point with Coord1 = C1V and Coord2 = C2V
Degeneration   Coord1 C1V      Coord2 C2V
//Degeneration into a set of points with Coord1 = C1V and ND2 divisions along Coord2 range
Degeneration   Coord1 C1V      Div2      ND2
//Degeneration into a set of points with ND1 divisions along Coord1 range and Coord2 = C2V
Degeneration   Div1      ND1      Coord2 C2V
//Degeneration into a set of points with ND1 divisions along Coord1 range and ND2 divisions along Coord2 range
Degeneration   Div1      ND1      Div2      ND2
//Degeneration into a single curve with Coord1 = C1V
Degeneration   Coord1 C1V
//Degeneration into a single curve with Coord2 = C2V
Degeneration   Coord2 C2V
//Degeneration into a set of curves with ND1 divisions along Coord1
Degeneration   Div1      ND1
//Degeneration into a set of curves with ND2 divisions along Coord2
Degeneration   Div2      ND2
```

RigidTriangularSurface_1

Creates a rigid triangular surface.

Syntax:

RigidTriangularSurface_1	SID	Points	ID1	ID2	ID3	PilotNode	PNID

- SID: current surface identification number
- ID1, ID2 and ID3: points identification numbers
- PNID: pilot node identification number

Example:

RigidTriangularSurface_1	1	Points	1	2	3	PilotNode	1

Additional information:

The vertices of a rigid triangular region are defined by A, B and C, positioned at $\mathbf{x}_A$, $\mathbf{x}_B$ and $\mathbf{x}_C$, respectively. The surface is parameterized by:

$$\Gamma(\zeta,\theta) = [\mathrm{N}_A(\zeta,\theta) \quad \mathrm{N}_B(\zeta,\theta) \quad \mathrm{N}_C(\zeta,\theta)]\begin{bmatrix}\mathbf{x}_A\\ \mathbf{x}_B\\ \mathbf{x}_C\end{bmatrix}, \tag{37}$$

with $\mathrm{N}_A(\zeta,\theta) = -\frac{1}{2}(\zeta+\theta)$, $\mathrm{N}_B(\zeta,\theta) = \frac{1}{2}(1+\zeta)$ and $\mathrm{N}_C(\zeta,\theta) = \frac{1}{2}(1+\theta)$. The parameters ζ and θ map the points inside the triangular region. An example is given in Figure 22.

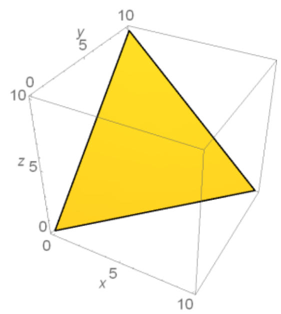


Figure 22 – Rigid triangular surface example

This surface is rigidly connected to a pilot node, which rules its movement.

Note: this surface is currently available for using together with the contact NSSS.

RigidOscillatorySurface_1

Creates a rigid oscillatory surface.

Syntax:

RigidOscillatorySurface_1			SID	A1	A1V	A2	A2V	A12	A12V	
	Lambda1	L1V	Lambda2		L2V	Phi1	P1V	Phi2	P2V	Waves1
	W1N	Waves2	W2N	CS	CSID	PilotNode		PNID		

- SID: current surface identification number
- A1V: amplitude to sin in direction ζ of surface parameterization
- A2V: amplitude to sin in direction θ of surface parameterization
- A12V: amplitude to the product of sines in directions ζ and θ of surface parameterization
- L1V: wave length along direction ζ of surface parameterization
- L2V: wave length along direction θ of surface parameterization
- P1V: phase along direction ζ of surface parameterization
- P2V: phase along direction θ of surface parameterization
- Waves1: number of waves along direction ζ of surface parameterization
- Waves2: number of waves along direction θ of surface parameterization
- CSID: coordinate system identification number
- PNID: pilot node identification number

Example:

RigidOscillatorySurface_1			1	A1	1.0	A2	1.0	A12	0.0	
	Lambda1	1.0	Lambda2		2.0	Phi1	0.0	Phi2	0.0	Waves1
	2.5	Waves2	1.5	CS	1	PilotNode		1		

Additional information:

A rigid oscillatory surface may be used to define wave patterns on a surface, possibly in two directions. One may define it as aligned with an arbitrary direction, and rigidly attached to a pilot node, which will rule its movement along the model evolution. Let one define a function in a local coordinate system (P, ζ, θ), where P is the pilot node position – origin of the system:

$$\Gamma(\zeta,\theta) = \mathbf{x}_{\mathrm{P}} + \mathbf{Q}\begin{bmatrix} \zeta \\ \theta \\ \Psi(\zeta,\theta) \end{bmatrix}, \tag{38}$$

where $\mathbf{x}_{\mathrm{P}}$ is the pilot node position, $\mathbf{Q}$ is a rotation matrix that rules the alignment of the surface in space, and depends on its initial orientation and on the pilot node rotation experienced during the model evolution. Finally, $\Psi(\zeta,\theta)$ is a function used to describe the local geometry of the surface, given by:

$$\Psi(\zeta,\theta) = A_1 \sin\left(\frac{2\pi\zeta}{\lambda_1} + \phi_1\right) + A_2 \sin\left(\frac{2\pi\theta}{\lambda_2} + \phi_2\right) + A_{12} \sin\left(\frac{2\pi\zeta}{\lambda_1} + \phi_1\right) \sin\left(\frac{2\pi\theta}{\lambda_2} + \phi_2\right), \tag{39}$$

where A_1, A_2 and A_{12} are amplitudes, λ_1 and λ_2 are wave-lengths and ϕ_1 and ϕ_2 are phases, all used to define the desired surface.

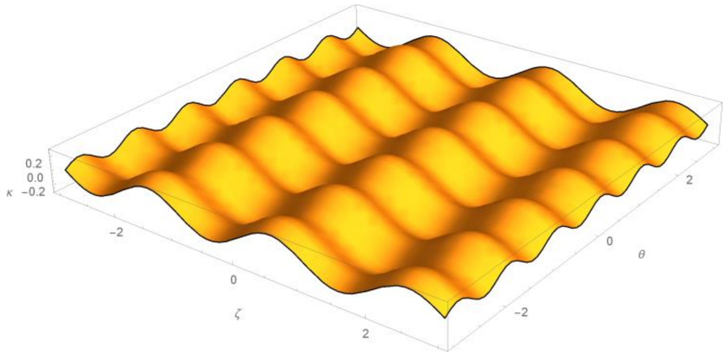


Figure 23 – Oscillatory surface example

Note: this surface is currently available for using together with the contact NSSS.

FlexibleSECylinder_1

Creates a flexible super elliptical surface.

Syntax:

```
//Surface with normal pointing outwards super elliptical cylinder
FlexibleSECylinder_1    SID    A    AV    B    BV    N    NV    CS    CSID
        NormalExterior Nodes  ID1   ID2
//Surface with normal pointing inwards super elliptical cylinder
FlexibleSECylinder_1    SID    A    AV    B    BV    N    NV    CS    CSID
        NormalInterior Nodes  ID1   ID2
//Defining surface with two distinct coordinate system alignments
FlexibleSECylinder_1    SID    A    AV    B    BV    N    NV    CSA    CSAID
        CSB    CSBID    NormalInterior  Nodes  ID1   ID2
```

- SID: current surface identification number
- AV: semi-axis value lying in direction E1
- BV: semi-axis value lying in direction E2
- NV: super ellipse exponent value
- CSID: coordinate system identification number (single alignment option)
- CSAID and CSBID: coordinate systems identification numbers (two distinct alignments option)
- ID1 and ID2: nodes identification numbers

NormalExterior or NormalInterior keywords are used to assign that the normal direction of the surface points outwards/inwards of super elliptical cylinder.

Example:

```
//Surface with normal pointing outwards super elliptical cylinder
FlexibleSECylinder_1    1    A    0.1    B    0.1    N    3.0    CS    1
        NormalExterior Nodes  1    2
//Surface with normal pointing inwards super elliptical cylinder
FlexibleSECylinder_1    2    A    0.1    B    0.1    N    3.0    CS    1
        NormalInterior Nodes  1    2
//Defining surface with two distinct coordinate system alignments
FlexibleSECylinder_1    3    A    0.1    B    0.1    N    3.0    CSA    1
        CSB    2    NormalInterior  Nodes  1    2
```

Additional information:

A flexible surface with super-elliptical cross section can be defined with this command. A schematic visualization of a deformed surface can be seen in Figure 24. The cross section of the surface is given by the expression:

$$\left|\frac{x}{a}\right|^{n}+\left|\frac{y}{b}\right|^{n}=1, \tag{40}$$

where "a" and "b" are super ellipse semi-axis. Parameter "n" is the curve exponent. The surface extreme positions are guided by the movement of nodes, located at each extreme cross section centroid. This includes translation and rotation. In Figure 24 one may see an example where one extreme cross section is twisted with respect to the other extreme cross section. This can be used to represent the external surface of a beam element in an approximated way.

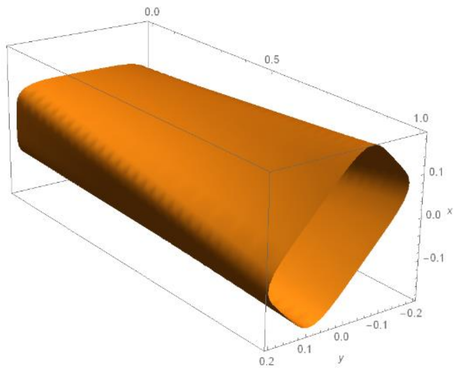


Figure 24 – Cylinder with super-elliptical cross section

The alignment of the surface extreme cross-sections at reference configuration is defined by two coordinate systems located at both nodes that define the surface limits. In case of a single direction is defined for both nodes, use a single coordinate system input using CS keyword. In case of distinct alignments, use two coordinate system inputs by keywords CSA and CSB.

Remark: the definition of two distinct coordinate systems (e.g.: CSA 1 CSB 2) is particularly useful when the reference configuration of two connecting surfaces FlexibleSECylinder_1 are not aligned. For example, see the extract of input code to Giraffe defined next, where two pairs of connected surfaces are defined. The pair on the left size uses two distinct alignments for FlexibleSECylinder_1, such that at the connection of surfaces the same coordinate system identification number is assigned for both. The pair on the right size assumes a single coordinate system for each surface, creating a distinct connection with discontinuities between surfaces.

In cases of creating a surface set to represent a single contact patch composed by many surfaces, the first option is desirable, since no holes are created between surfaces, which may lead to contact lose or bad convergence/divergence in models.

```
…
Node    12    -1    -1    0.7
Node    13    -1    +1    0.7
Node    14    +1    +1    0.7
Node    15    +2    -1    0.7
Node    16    +2    +1    0.7
Node    17    +4    +1    0.7
…
CS      4
CSYS    1     E1    1     0     0     E3    0     0     1
CSYS    2     E1    0     0     1     E3    0     1     0
CSYS    3     E1    0     0     1     E3    1     1     0
CSYS    4     E1    0     0     1     E3    1     0     0
…
FlexibleSECylinder_1 1 A 0.3 B 0.3 N 2.4 CSA    2    CSB    3    NormalExterior    Nodes 12  13
FlexibleSECylinder_1 2 A 0.3 B 0.3 N 2.4 CSA    3    CSB    4    NormalExterior    Nodes 13  14
FlexibleSECylinder_1 3 A 0.3 B 0.3 N 2.4 CS  2    NormalExterior    Nodes 15  16
FlexibleSECylinder_1 4 A 0.3 B 0.3 N 2.4 CS  4    NormalExterior    Nodes 16  17
```

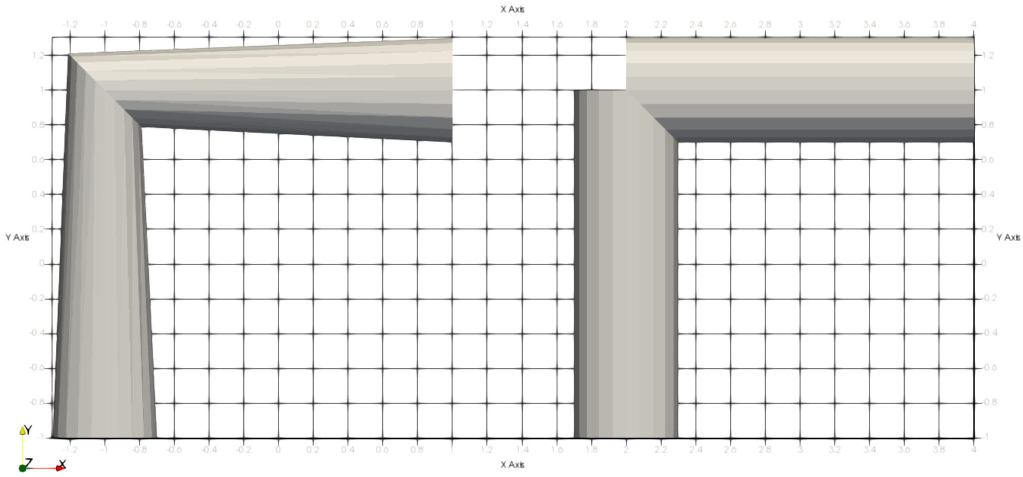


Figure 25 – Example of distinct/single coordinate systems to align two FlexibleSECylinder_1 surfaces

FlexibleTriangularSurface_2

Creates a flexible triangular surface.

Syntax:

FlexibleTriangularSurface_2	SID	Nodes	ID1	ID2	ID3	ID4	ID5	ID6

- SID: current surface identification number
- ID1, ID2, ID3, ID4, ID5 and ID6: nodes identification numbers

Example:

FlexibleTriangularSurface_2	1	Nodes	1	2	3	4	5	6

Additional information:

The nodes A, B and C are the vertices of a triangular surface and D, E and F are mid-points of the edges of a reference triangle. They are located on $\mathbf{x}_K$, where K assumes any of such node's indexes. A triangular surface Γ may be parameterized by:

$$\Gamma(\zeta,\theta) = \begin{bmatrix} N_A^2(\zeta,\theta) & N_B^2(\zeta,\theta) & N_C^2(\zeta,\theta) & N_D^2(\zeta,\theta) & N_E^2(\zeta,\theta) & N_F^2(\zeta,\theta) \end{bmatrix} \begin{bmatrix} \mathbf{x}_A \\ \mathbf{x}_B \\ \mathbf{x}_C \\ \mathbf{x}_D \\ \mathbf{x}_E \\ \mathbf{x}_F \end{bmatrix}, \quad (41)$$

where $N_K^2(\zeta,\theta)$ are shape functions in plane $\zeta\theta$. This surface experiences deformation following the interpolation of nodes, located at points A, B, C, D, E and F. Figure 26 shows an example of such parameterization, deformed.

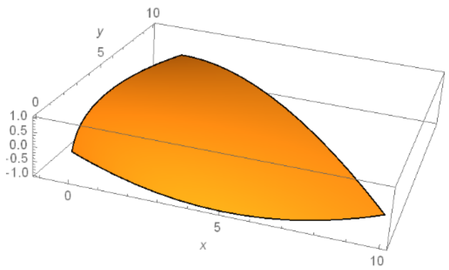


Figure 26 – Flexible triangular surface example

This surface can be used to establish a surface candidate to contact covering a shell element. The node sequence pattern follows exact the same rule of Shell_1 element (see Shell_1).

FlexibleArcExtrusion_1

Creates a surface generated by the extrusion of an arc along the path defined by nodes.

Syntax:

```
//Surface with external normal pointing to the center of arc
FlexibleArcExtrusion_1 SID     Arc     AID     CS     CSID     Nodes  ID1     ID2
          Concave
//Surface with external normal pointing opposite to the center of arc
FlexibleArcExtrusion_1 SID     Arc     AID     CS     CSID     Nodes  ID1     ID2     Convex
```

- SID: current surface identification number
- AID: arc identification number
- CSID: coordinate system identification number
- ID1, ID2: nodes identification numbers
- Concave: indicates that the surface external normal points to the center of arc
- Convex: indicates that the surface external normal points in the direction opposite to the center of arc

Example:

```
FlexibleArcExtrusion_1 1     Arc     1     CS     1     Nodes  1     2
          Concave
FlexibleArcExtrusion_1 2     Arc     1     CS     1     Nodes  1     2     Convex
```

Additional information:

This surface is based on the definition of an arc on a local coordinate system, as shown in Figure 21 and presented in [7]. The flexible extruded arc surface is designed to be attached to two nodes, defining the direction and the limits of extrusion.

$$\Gamma(\zeta,\theta) = \mathrm{h}_1(\mathbf{Q}_1\mathbf{a}(\theta) + \mathbf{x}_1) + \mathrm{h}_2(\mathbf{Q}_2\mathbf{a}(\theta) + \mathbf{x}_2). \tag{42}$$

where $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are operators, which encompass: (i) transformation between local and global coordinate systems – from local arc definition to the desired global orientation and (ii) rotation experienced by nodes 1 and 2, respectively. Vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ are the positions of nodes 1 and 2, respectively (considered as local origins of the local coordinate system employed to define the arc). Shape functions h_1 and h_2 are given by:

$$\mathrm{h}_1 = \frac{1}{2}(1-\zeta) \text{ and}$$
$$\mathrm{h}_2 = \frac{1}{2}(1+\zeta), \tag{43}$$

which define the extrusion parameter ζ. The local curve parameterization is given by:

$$\mathbf{a}(\theta) = \begin{bmatrix} r\cos\theta + \mathrm{XCP} \\ r\sin\theta + \mathrm{YCP} \\ 0 \end{bmatrix}, \tag{44}$$

as defined in Arcs section.

The extrusion direction is E3 of the local coordinate system CSID. Figure 27 illustrates the extruded surface. In this figure the arc center point $\mathbf{c} = (\mathrm{XCP}, \mathrm{YCP}) = (c_1, c_2)$.

If the user's choice for the external normal direction is "Convex", it will be given by:

$$\mathbf{n}_{\mathrm{ext}} = \frac{\Gamma_{,\theta} \times \Gamma_{,\zeta}}{\|\Gamma_{,\theta} \times \Gamma_{,\zeta}\|}. \tag{45}$$

If the choice is "Concave", it will be given by:

$$\mathbf{n}_{\mathrm{ext}} = -\frac{\Gamma_{,\theta} \times \Gamma_{,\zeta}}{\|\Gamma_{,\theta} \times \Gamma_{,\zeta}\|}. \tag{46}$$

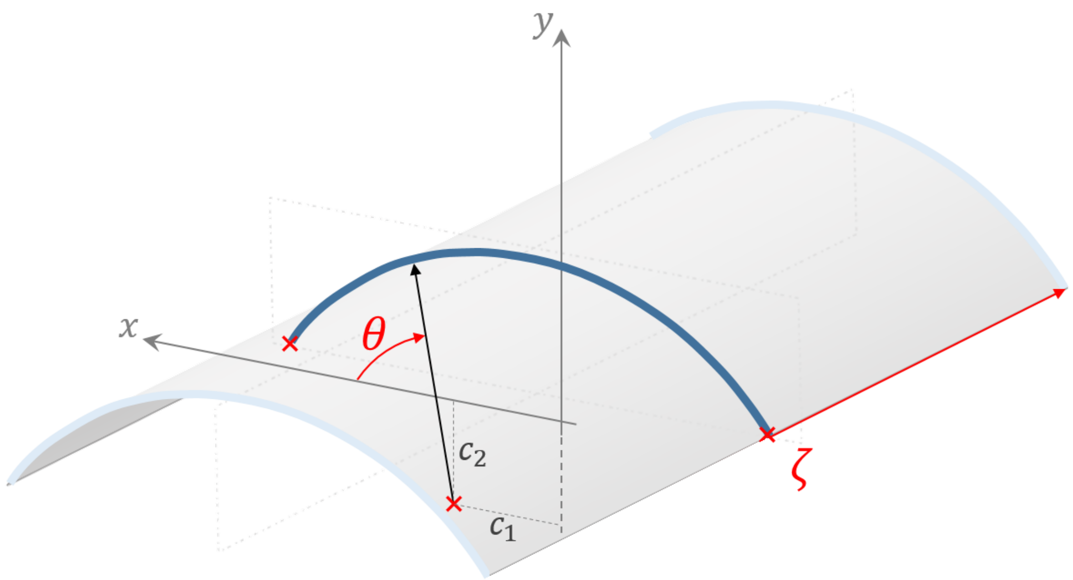


Figure 27 – Flexible extruded arc surface (figure from [7])

Note: this surface is currently available for using together with the contact SSSS.

RigidArcRevolution_1

Creates a surface generated by the revolution of an arc about a local axis.

Syntax:

```
//Surface with external normal pointing to the center of arc
RigidArcRevolution_1   SID    Arc    AID    CS    CSID    Node    NID    Concave
         RevolutionAngle      RANG   FactorX      XF      FactorZ        ZF
//Surface with external normal pointing opposite to the center of arc
RigidArcRevolution_1   SID    Arc    AID    CS    CSID    Node    NID    Convex
         RevolutionAngle      RANG   FactorX      XF      FactorZ        ZF
```

- SID: current surface identification number
- AID: arc identification number
- CSID: coordinate system identification number
- NID: node identification number
- Concave: indicates that the surface external normal points to the center of arc
- Convex: indicates that the surface external normal points in the direction opposite to the center of arc

Optional data:

- RANG: value for the maximum revolution angle coordinate ϕ (default ϕ ranges from 0 to 2π – full revolution)
- XF: ovalization coefficient for local x direction (default value is 1.0)
- ZF: ovalization coefficient for local z direction (default value is 1.0)

Example:

```
RigidArcRevolution_1   1    Arc    1    CS    1    Node    1    Concave
```

Additional information:

This surface is based on the definition of an arc on a local coordinate system, as shown in Figure 21 and presented in [7]. The rigid arc revolution surface is designed to be attached to a single node.

$$\Gamma(\theta,\phi) = \mathbf{Q}\mathbf{a}(\theta,\phi) + \mathbf{x}. \qquad (47)$$

where $\mathbf{Q}$ is an operator that transforms between local and global coordinate systems – from local arc definition to the desired global orientation. Vector $\mathbf{x}$ is the position of the node, considered as the origin of the local coordinate system employed to define the arc.

The local surface parameterization $\mathbf{a}(\theta,\phi)$ is given by:

$$\mathbf{a}(\theta,\phi) = \begin{bmatrix} (r\cos\theta + c_1)(x_{\mathrm{factor}}\cos\phi) \\ r\sin\theta + c_2 \\ -\left((r\cos\theta + c_1)(z_{\mathrm{factor}}\sin\phi)\right) \end{bmatrix}, \qquad (48)$$

such that the revolution axis is E2 of the local coordinate system CSID. The definition of the arc is made following the guidelines presented in in Arcs section. Revolution angular parameter is given by ϕ .Optional data x_{factor} and z_{factor} are coefficients to rule an ovalization pattern. If not input by the user, their values are considered as $x_{\text{factor}} = 1$ and $z_{\text{factor}} = 1$. Figure 28 illustrates the revolved surface. In this figure the arc center point $\mathbf{c} = (\text{XCP}, \text{YCP}) = (c_1, c_2)$.

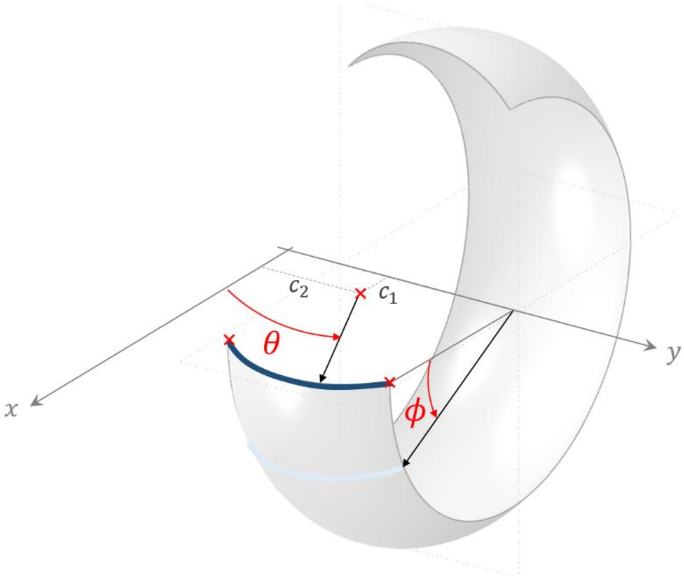


Figure 28 – Rigid arc revolution surface (figure from [7])

If the user's choice for the external normal direction is "Convex", it will be given by:

$$\mathbf{n}_{\text{ext}} = \frac{\Gamma_{,\phi} \times \Gamma_{,\theta}}{\|\Gamma_{,\phi} \times \Gamma_{,\theta}\|}. \tag{49}$$

If the choice is "Concave", it will be given by:

$$\mathbf{n}_{\text{ext}} = -\frac{\Gamma_{,\phi} \times \Gamma_{,\theta}}{\|\Gamma_{,\phi} \times \Gamma_{,\theta}\|}. \tag{50}$$

Note: this surface is currently available for using together with the contact SSSS.

RigidNURBS_1

Creates a rigid NURBS surface attached to a node and oriented according to a local coordinate system.

Syntax:

RigidNURBS_1	SID	CS	CSID	PilotNode	NID	CADData	CDID

- SID: current surface identification number
- CSID: coordinate system identification number
- NID: pilot node identification number
- CDID: CADData identification number

Example:

RigidNURBS_1	1	CS	1	PilotNode	1	CADData	1

Additional information:

RigidNURBS_1 follows the rigid body-surface parameterization presented in [15] given by:

$$\Gamma(u,v) = \mathbf{Q}_O \mathbf{s}(u,v) + \mathbf{x}_O, \tag{3}$$

where $\mathbf{s}(u,v)$ is a locally-defined NURBS surface parameterization, as shown in equation (2), $\mathbf{Q}_O$ is a rotation tensor to align the NURBS surface with the desired coordinate system defined by the CS keyword and $\mathbf{x}_O$ to translate it to a desired location, which is the pilot node position. The expression $\Gamma(u,v)$ depends on the model degrees of freedom, leading to the possibility of updating the rigid body surface on a transient dynamics model evolution. The pilot node translation and rotation will rule the rigid surface position/orientation.

An example of RigidNURBS_1 is shown in Figure 29:

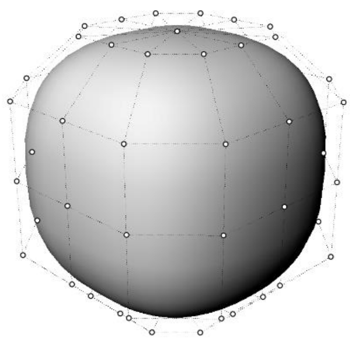


Figure 29 – Rigid NURBS surface (figure from [15])

Note: this surface is currently available for using together with the contact SSSS.

Monitors

Creates monitors for post-processing results of nodes, elements, contacts and node sets.

Syntax:

```
Monitor          Sample SV
//Optional keyword to monitor nodes:
MonitorNodes          nodes IDs
//Optional keyword to monitor elements:
MonitorElements       elements IDs
//Optional keyword to monitor contacts:
MonitorContacts       contacts IDs
//Optional keyword to monitor node sets:
MonitorNodeSets       node sets IDs
```

- SV: sampling for saving data in monitor output files (use 1 to save all converged solutions and larger integer numbers for decreasing data size)
- nodes IDs: list of node identification numbers (to be monitored)
- elements IDs: list of element identification numbers (to be monitored)
- contacts IDs: list of contact identification numbers (to be monitored)
- node sets IDs: list of node set identification numbers (to be monitored)

Example:

```
Monitor          Sample SV
MonitorNodes          1     2
MonitorElements       1     2
MonitorContacts       1
MonitorNodeSets       1
```

Additional information:

Monitors are extremely useful for analyzing time series. Each monitor has specific file formats, depending on the entity chosen. Giraffe saves monitor files during the simulation evolution with a sampling frequency ruled by the attribute input after Sample keyword. NodeSets monitor evaluates along time the following quantities:

- the average position of the nodes in the node set;
- the total force applied on all nodes in the node set;
- the total moment applied on the nodes in the node set (the total moment is composed by the moments applied at each node and the transport moment of the force in each node to the average position of the nodes – taken as pole)

PostFiles

Creates post files for post-processing results using Paraview™ post-processor interface.

Syntax:

```
PostFiles
MagFactor       MFV
WriteMesh       MF
WriteRenderMesh         RMF
WriteRigidContactSurfaces       RCSF
WriteFlexibleContactSurfaces    FCSF
WriteForces     FF
WriteConstraints        CF
WriteSpecialConstraints         SCF
WriteContactForces      CFF
WriteRenderRigidBodies          RBF
WriteRenderParticles    PF
```

- MFV: value of the magnification factor for displacements (for visualization purposes)
- MF: Boolean flag to write (1) or not (0) the mesh file
- RMF: Boolean flag to write (1) or not (0) the render mesh file
- RCSF: Boolean flag to write (1) or not (0) the rigid contact surfaces file
- FCSF: Boolean flag to write (1) or not (0) the flexible contact surfaces file
- FF: Boolean flag to write (1) or not (0) the forces file
- CF: Boolean flag to write (1) or not (0) the constraints file
- SCF: Boolean flag to write (1) or not (0) the special constraints file
- CFF: Boolean flag to write (1) or not (0) the contact forces file
- RBF: Boolean flag to write (1) or not (0) the rigid bodies file
- RBF: Boolean flag to write (1) or not (0) the particles file

Example:

```
PostFiles
MagFactor       1.0
WriteMesh       1
WriteRenderMesh         1
WriteRigidContactSurfaces       0
WriteFlexibleContactSurfaces    0
WriteForces     0
WriteConstraints        0
WriteSpecialConstraints         0
WriteContactForces      0
WriteRenderRigidBodies          0
WriteRenderParticles    0
```

Additional information:

PostFiles keyword activates saving of output files containing information for post-processing the simulation using PARAVIEW™. The sampling for saving post files is established on solution steps definition.

The MagFactor keyword is a magnification factor that will be used to multiply all the displacements experienced in the model, in the visualization of deformed shape frames. If the user enters "1.0", the deformed shape will show deformation patterns in real scale. A larger value than "1.0" can be used in case of simulations involving very small displacements/rotations, to help on visualization of results.

Some write control flags have to be set by the user. Each one may be turned on/off, by the values "1" or "0", respectively. The choice of adequate save outputs permit to visualize more details of the model and are very useful for generating high-quality animations and good post-processing interpretations.

Each write control flag is described below:

- WriteMesh: to write the base mesh information. E.g.: beams are represented by lines passing representing the axis. Particles are represented by points. Shells are represented my mid-surfaces.
- WriteRenderMesh: to write the rendered mesh, every element as a 3D solid. E.g.: beams are represented by the chosen cross section extruded along the axis direction. Shells are represented including the thickness information.
- WriteRigidContactSurfaces: to write the rigid contact surfaces. E.g.: RigidTriangularSurface_1.
- WriteFlexibleContactSurfaces: to write the flexible contact surfaces. E.g.: FlexibleTriangularSurface_2.
- WriteForces: to write data containing information of external applied forces on nodes. It covers NodalLoads and NodalFollowerLoads. It may be used to construct arrow glyphs in Paraview™ interface.
- WriteConstraints: to write constraints symbols.
- WriteSpecialConstraints: to write special constraints symbols.
- WriteContactForces: to write data with contact forces locations and associated normal and friction values. It may be used to construct arrow glyphs in Paraview™ interface.
- WriteRenderRigidBodies: to write data with rigid bodies rendering points.
- WriteRenderParticles: to write particles data. External surfaces are represented.

Useful data is written by Giraffe when requesting WriteRenderMesh. A vector data associated with each cell, named ElementProperties contains information about:

- Element type associated number (according to Table 7)
- Associated material number
- Associated section number (for beams, shells and trusses)
- Associated coordinate system number

Table 7 – Element types and associated numbers

Element type	number
Beam_1	1
Pipe_1	2
Shell_1	3
Mass_1	4
SpringDashpot_1	5
RigidBody_1	6
Truss_1	7

The objective of this data is to provide the user the possibility of creating selections for better post-processing in Paraview^{TM} (e.g.: selecting only cells associated with beam elements, or with a given material number, etc.). See Appendix for more information.

Note: post files are always created in the model, even if the user does not request then. In such case, the only output will be the base mesh.

SolverOptions

Sets solver options (for parallel processing).

Syntax:

```
SolverOptions
Processors      NP      LinSys   ST
```

- NP: number of processors (cores) to be used for processing the model
- ST: solver type for systems of linear equations ("Direct" or "Iterative").

Example:

```
SolverOptions
Processors      4       LinSys   Direct
```

Additional information:

The SolverOptions keyword is used by Giraffe to set the parallel processing solver options. It rules how Giraffe will use OpenMP™ parallel processing routines, which can be really useful for speeding up model solution. Linear systems of equations are solved by PARDISO™ library routines.

SolutionSteps

Establishes a sequence of solution steps to be solved by Giraffe.

Syntax:

```
SolutionSteps   N
Name   ID       data
```

- N: number of solution steps
- Name: current solution step name
- ID: current solution step identification number
- data: current solution step data (depends on solution step resources and requirements)

Example:

```
SolutionSteps   2

Static    1
EndTime         2
TimeStep        0.1
MaxTimeStep     0.2
MinTimeStep     0.01
MaxIt    12
MinIt    3
ConvIncrease    2
IncFactor       1.2
Sample 1

Dynamic         2
EndTime         3
TimeStep        0.1
MaxTimeStep     0.2
MinTimeStep     0.01
MaxIt    12
MinIt    3
ConvIncrease    2
IncFactor       1.2
Sample 2
RayleighDamping        Alpha   0       Beta     0       Update 0
NewmarkCoefficients    Beta    0.3     Gamma            0.5
```

Additional information:

Each solution step is defined by a specific keyword followed by the solution step identification number (must be an ascending sequence starting from number one) and additional data. Each solution step available and its input data is explained next.

Solution steps are used to create a sequence of solutions. There is a global "time" tracking parameter to rule all solution steps. The default start-time is zero. Then, each solution step has a definition of end-time. Note that the end-time of a given solution step must be larger than the end-time of the previous solution step, otherwise Giraffe will prompt an error message prior to solution start. Exceptions are modal analysis solution steps that have no end-time parameter as input. In this case, time is considered frozen during modal analysis.

Multiple solution steps may be created to establish starting/ending of constraint actions, contacts, special constraints, or even to split between statics and dynamics, according to the nonlinear model convenience.

The final converged model configuration at the end of a solution step is always taken as the start point for the next solution step. When modal analysis is performed, no changes exist on the model configuration for a next solution step.

Constraints, special constraints and contacts are considered according to the defined BoolTable in each particular creation of such resources. Definition of loads or displacements is made for each solution step following the time variable as a global tracking.

After simulation is finished, the user will find requested result files for each solution step in separate folders: "/post/solution_i/", where "i" is the solution identification number.

Static

Creates a solution step to solve a nonlinear static analysis.

Syntax:

```
Static    SID
EndTime        EV
TimeStep       TS
MaxTimeStep    MAX
MinTimeStep    MIN
MaxIt          MAXIT
MinIt          MINIT
ConvIncrease   CONV
IncFactor      INCF
Sample         SA
```

- SID: current solution step identification number
- EV: end time of current solution step
- TS: time-step of current solution step
- MAX: maximum time-step of current solution step
- MIN: minimum time-step of current solution step
- MAXIT: maximum number of iterations to be performed during Newton-Raphson routine, for each time-step within the current solution step
- MINIT: minimum number of iterations, to indicate convergence easiness
- CONV: number of sequential converged time-steps to indicate convergence easiness
- INCF: time-step increasing factor
- SA: sampling variable to rule post-processing files generation.

Example:

```
Static    1
EndTime        2.5
TimeStep       0.10
MaxTimeStep    0.25
MinTimeStep    0.01
MaxIt          15
MinIt          3
ConvIncrease   4
IncFactor      1.5
Sample         1
```

Additional information:

A static solution step is defined by a sequence of attributes, with the objective of creating an auto-adaptive scheme for nonlinear solution, capable of increasing or decreasing the time-step automatically. In the context of a static analysis, the time variable may be understood as a scalar tracking parameter that permits evaluation of a sequence of loads, constraints and boundary conditions. The time-period of a given static solution step goes from the previously converged time value until the end time of the current solution step, set by the user. Depending on the easiness or hardness of convergence, time-step may be updated automatically according to the parameters, explained below:

- EndTime: defines the final instant for the current solution step.
- TimeStep: defines the initial time step to be used for the evolution of the nonlinear model.
- MaxTimeStep: defines the maximum time step to be used for the evolution of the nonlinear model.
- MinTimeStep: defines the minimum time step to be used for the evolution of the nonlinear model. In case of convergence difficulties, Giraffe automatically performs bisections (i.e., decreases automatically the time-step). In case of many unsuccessful bisections, when the minimum time step is approached, the simulation stops with an error message.
- MaxIt: defines the maximum number of iterations, which will be performed prior to assume that divergence occurred. There are two possibilities of divergence: by achieving the maximum number of iterations or by achieving a very high residual value, defined by ConvergenceCriteria keyword.
- MinIt: defines the minimum number of iterations. Once a time step converges with less or equal to this number of iterations, the next time step will be increased by the IncFactor coefficient. Thus, it may be seen as an identifier of easy solution.
- ConvIncrease: defines the sequential number of converged time steps that, once achieved, will also show that there is the possibility of increasing the time step value. Then, IncFactor coefficient is used to increase the time-step. Once applied, the converged partial solutions counting process re-starts.
- IncFactor: defines the factor for increasing the time step, in case of easy convergence.
- Sample: defines the sampling for saving post-processing files with partial converged solutions along time evolution. Entering the number “1” claims Giraffe to save all converged steps (many files can be generated).

Dynamic

Creates a solution step to solve a nonlinear dynamic analysis (transient dynamics).

Syntax:

```
Dynamic         SID
EndTime         EV
TimeStep        TS
MaxTimeStep     MAX
MinTimeStep     MIN
MaxIt           MAXIT
MinIt           MINIT
ConvIncrease    CONV
IncFactor       INCF
Sample          SA
RayleighDamping       Alpha   AD      Beta    BD      Update UD
NewmarkCoefficients   Beta    BN      Gamma           GN
```

- SID: current solution step identification number
- EV: end time of current solution step
- TS: time-step of current solution step
- MAX: maximum time-step of current solution step
- MIN: minimum time-step of current solution step
- MAXIT: maximum number of iterations to be performed during Newton-Raphson routine, for each time-step within the current solution step
- MINIT: minimum number of iterations, to indicate convergence easiness
- CONV: number of sequential converged time-steps to indicate convergence easiness
- INCF: time-step increasing factor
- SA: sampling variable to rule post-processing files generation.
- AD: coefficient that multiplies mass matrix for Rayleigh damping evaluation
- BD: coefficient that multiplies stiffness matrix for Rayleigh damping evaluation
- UD: flag to update (1) or not (0) the Rayleigh damping matrix in each time step beginning
- BN and GN: Newmark time-integrator β and γ parameters

Example:

```
Dynamic         1
EndTime         2.5
TimeStep        0.10
MaxTimeStep     0.25
MinTimeStep     0.01
MaxIt           15
MinIt           3
ConvIncrease    4
IncFactor       1.5
Sample          1
RayleighDamping       Alpha   0.0     Beta    0.0     Update 0
NewmarkCoefficients   Beta    0.3     Gamma           0.5
```

Additional information:

A dynamic solution step is defined by a sequence of attributes, with the objective of creating an auto-adaptive scheme for nonlinear solution, capable of increasing or decreasing the time-step automatically. In the context of a dynamic analysis, the time variable is the physical time, differently from static solution steps. The time-period of a given dynamic solution step goes from the previously converged time value until the end time of the current solution step, set by the user. Depending on the easiness or hardness of convergence, time-step may be updated automatically according to the same parameters explained for "Static" type of solution step.

Dynamic simulations also include damping control. Rayleigh damping model is implemented, through usage of the following instruction example:

RayleighDamping Alpha 0 Beta 0 Update 0

The attributes are explained next:

- Alpha: coefficient that multiplies mass matrix for compounding damping matrix
- Beta: coefficient that multiplies stiffness matrix for compounding damping matrix
- Update: flag, which can assume "1" or "0". If updating is turned on then the damping matrix is updated in each time step beginning, with updated information about stiffness and mass matrices. If updating is turned off then the initial calculated damping (with initial stiffness and mass matrices) is kept during the whole solution step.

Newmark method is used to integrate equations along time. Two coefficients are defined in Newmark method. One can refer to [16] for more details. These coefficients are input through NewmarkCoefficients Beta 0.3 Gamma 0.5. These are the recommended values to be used for time-integration. The user can change such values in some particular simulations to induce numerical damping. For example, increasing Gamma from 0.5 to a value up to 0.6 and keeping Beta 0.3 usually introduces high-frequency numerical damping.

Modal

Creates a solution step to solve a modal analysis.

Syntax:

Modal	SID
ExportMatrices	EMF
NumberModes	NM
Tolerance	TV
ComputeEigenvectors	CEF
NumberFrames	NF

- SID: current solution step identification number
- EMF: a flag to export (1) or not (0) mass and stiffness matrices as text files (sparse matrix formats)
- NM: number of modes required
- TV: tolerance for ARPACK™ eigenvalues/eigenvectors extraction
- CEF: a flag to compute (1) or not (0) the model eigenvectors
- NF: number of frames exported to animate each model eigenvector

Example:

Modal	1
ExportMatrices	0
NumberModes	12
Tolerance	1E-6
ComputeEigenvectors	1
NumberFrames	6

Additional information:

Modal analysis has no end-time information. Thus, during modal analysis time is considered frozen.

When performing modal analysis in a model with special constraints the results will have no meaning, due to Lagrange multipliers present in the model (still not treated for modal analysis in current Giraffe version).

Giraffe evaluates always the lowest magnitude eigenvalues of the system (possibly complex numbers). Depending on the requested results, the available files will be:

- DOF_table_i.txt: a table containing the system connectivity. It contains, for each node and local DOF, the global DOF number.
- eigenvalues_solution_i.txt: a list with the evaluated eigenvalues (real and imaginary parts). In case the eigenvalue is a real number, the natural frequency may be evaluated as the square root of it (rad/s).
- m_mass_i.txt: mass matrix of the system
- m_stiffness_i.txt: stiffness matrix of the system.

Post-processing of modal analysis is detailed in Appendix: Post-processing modal analysis using Paraview™.

ConcomitantSolution

Establishes a modal concomitant solution to be solved repeated times within a given solution step (or solution steps).

Syntax:

```
ConcomitantSolution    Sample SV       BoolTable        BDC
Modal   NumberModes NM        Tolerance         TV
```

- SV: sampling variable to rule concomitant solution call
- BDC: bool table data for concomitant solution
- NM: number of modes required
- TV: tolerance for ARPACK^{TM} eigenvalues/eigenvectors extraction

Example:

```
ConcomitantSolution    Sample 5        BoolTable        1
Modal   NumberModes 10        Tolerance         1E-6
```

Additional information:

A concomitant solution may be created to ask Giraffe to solve extra solutions within a given solution step, or even along more than one solution step. For example, during a static or dynamic solution steps, the user may be interested in evaluating system modal analysis along time evolution. In this case a concomitant solution may be created. It does not influence in time-evolution, neither in solution steps sequence.

According to the sample variable a modal concomitant solution will be called. If SV is "1", all converged time-steps will lead to a call of concomitant solution. Otherwise, larger integer SV will lead to less concomitant solution calls, always at each SV converged time-steps. Currently only concomitant modal analysis is available in Giraffe.

At the end of the simulation, the user will find as the result of concomitant solution a text file containing the time-series of evaluated eigenvalues along time. It is located inside the folder "/post/concomitant_solution/".

ConvergenceCriteria

Establishes convergence criteria.

Syntax:

ConvergenceCriteria	
ForceTolerance	FTV
MomentTolerance	MTV
ForceMinimumReference	FMRV
MomentMinimumReference	MMRV
ConstraintMinimumReference	CMRV
DisplacementTolerance	DTV
RotationTolerance	RTV
LagrangeTolerance	LTV
DisplacementMinimumReference	DMRV
RotationMinimumReference	RMRV
LagrangeMinimumReference	LMRV
DivergenceReference	DRV

- FTV: force tolerance value
- MTV: moment tolerance value
- FMRV: force minimum reference value
- MMRV: moment minimum reference value
- CMRV: constraint minimum referentece value
- DTV: displacement tolerance value
- RTV: rotation tolerance value
- LTV: Lagrange multiplier tolerance value
- DMRV: displacement minimum reference value
- RMRV: rotation minimum reference value
- LMRV: Lagrange multiplier minimum reference value
- DRV: divergence reference value

Example:

ConvergenceCriteria	
ForceTolerance	1e-4
MomentTolerance	1e-4
ForceMinimumReference	1e-5
MomentMinimumReference	1e-5
ConstraintMinimumReference	1e-7
DisplacementTolerance	1e-4
RotationTolerance	1e-4
LagrangeTolerance	1e-4
DisplacementMinimumReference	1e-6
RotationMinimumReference	1e-6
LagrangeMinimumReference	1e-6
DivergenceReference	1e+15

Additional information:

Since Giraffe was designed to solve nonlinear finite element models, one has to define convergence criteria, in order to guide the Newton-Raphson iterative method to stop, according to some rules. Next, we describe each individual convergence criterion that Giraffe applies. A solution is considered as "converged" if all the applied criteria are obeyed simultaneously. Stricter criteria will need more iterations to achieve convergence, but will have more precision.

Default convergence criteria usually works properly for general nonlinear simulations. In such cases, the user does not need to re-establish then. The usage of the ConvergenceCriteria command in Giraffe input file should be made with care, and is proper for advanced users.

- ForceTolerance: a factor that multiplies the norm of the external forces vector, used to establish a criterion of maximum allowable error for the norm of the unbalanced forces vector. Default value is 0.01%.
- MomentTolerance: a factor that multiplies the norm of the external moments vector, used to establish a criterion of maximum allowable error for the norm of the unbalanced moments vector. Default value is 0.01%.
- ForceMinimumReference: a value of force, taken as very small, to avoid the establishment of a never achievable null convergence criterion. This would occur in cases for which no external forces are applied. In such situations, the criterion of maximum allowable error for the norm of the unbalance forces vector is evaluated by: ForceMinimumReference * ForceTolerance. Default value is 1e-5.
- MomentMinimumReference: a value of moment, taken as very small, to avoid the establishment of a never achievable null convergence criterion. This would occur in cases for which no external moments are applied. In such situations, the criterion of maximum allowable error for the norm of the unbalance moments vector is evaluated by: MomentMinimumReference * MomentTolerance. Default value is 1e-5.
- ConstraintMinimumReference: a small value, taken as the maximum residual for the constraints established by SpecialConstraints command. Default value is 1e-7.
- DisplacementTolerance: a factor that multiplies the norm of the displacements vector (experienced during the current time-step – for dynamics or sub step – for statics). It is

used to establish a criterion of maximum allowable norm for the iterative displacements increment (Newton Raphson). Default value is 0.01%.

- RotationTolerance: a factor that multiplies the norm of the rotations vector (experienced during the current time-step – for dynamics or sub step – for statics). It is used to establish a criterion of maximum allowable norm for the iterative rotations increment (Newton Raphson). Default value is 0.01%.
- LagrangeTolerance: a factor that multiplies the norm of the Lagrange multipliers vector (experienced during the current time-step – for dynamics or sub step – for statics). It applies only when the simulation has SpecialConstraints. It is used to establish a criterion of maximum allowable norm for the iterative Lagrange multipliers increment (Newton Raphson). Default value is 0.01%.
- DisplacementMinimumReference: a value of displacement, taken as very small, to avoid the establishment of a never-achievable null criterion for the maximum allowable norm of the iterative displacements increment. This would occur for cases in which no displacements occur in the system. In such situations, the criterion for maximum allowable norm of the iterative displacements increment is evaluated by: DisplacementMinimumReference * DisplacementTolerance. Default value is 1e-6.
- RotationMinimumReference: a value of rotation, taken as very small, to avoid the establishment of a never-achievable null criterion for the maximum allowable norm of the iterative rotations increment. This would occur for cases in which no rotations occur in the system. In such situations, the criterion for maximum allowable norm of the iterative rotations increment is evaluated by: RotationMinimumReference * RotationTolerance. Default value is 1e-6.
- LagrangeMinimumReference: a value of Lagrange multiplier, taken as very small, to avoid the establishment of a never-achievable null criterion for the maximum allowable norm of the iterative Lagrange multipliers increment. This would occur for cases in which only null Lagrange multipliers occur in the system. In such situations, the criterion for maximum allowable norm of the iterative Lagrange multipliers increment is evaluated by: LagrangeMinimumReference * LagrangeTolerance. It applies only when the simulation has SpecialConstraints. Default value is 1e-6.
- DivergenceReference: defines a very large residual number, which once achieved, means "divergence". Then, Giraffe automatically will perform a bisection (dividing the last load factor by two) in order to try to achieve convergence in next time-step or sub step. Default value is 1e+15.

Acknowledgements

Giraffe developers and users would like to thank FAPESP and CNPq for funding research projects and scholarships related to Giraffe developments.

References

1. GAY NETO, A.; MARTINS, C. A.; PIMENTA, P. M. Static analysis of offshore risers with a geometrically-exact 3D beam model subjected to unilateral contact. **Comp. Mechanics** , v. 53, p. 125-145, 2014.

2. GAY NETO, A. Dynamics of Offshore Risers using a Geometrically-exact Beam Model with Hydrodynamic Loads and Contact with the Seabed. **Eng. Structures**, v. 125, p. 438-454, 2016.

3. CAMPELLO, E. M. B.; PIMENTA, P. M.; WRIGGERS, P. A triangular finite shell element based on a fully nonlinear shell formulation. **Comp. Mechanics**, v. 31, p. 505-518, 2003.

4. PIEGL, L.; TILLER, W. **The NURBS book**. Second. ed. Heidelberg: Springer, 1997.

5. YOJO, T. **Análise não-linear geometricamente exata de pórticos espaciais**. São Paulo: Universidade de São Paulo, v. Tese de Doutorado, 1993.

6. GAY NETO, A. Simulation of Mechanisms Modeled by Geometrically-Exact Beams using Rodrigues Rotation Parameters. **Comp. Mechanics**, v. 59 (3), p. 459-481, 2017.

7. DE CAMPOS, P. R. R.; GAY NETO, A. Rigid Body formulation in a finite element context with contact interaction. **Comp. Mech.**, n. First Online: 24 March 2018, 2018.

8. GAY NETO, A.; PIMENTA, P. M.; WRIGGERS, P. Self-contact modeling on beams experiencing loop formation. **Comp. Mechanics** , v. 55(1), p. 193-208, 2015.

9. GAY NETO, A.; PIMENTA, P. M.; WRIGGERS, P. A Master-surface to Master-surface Formulation for Beam to Beam Contact. Part I: Frictionless Interaction. **Comput. Methods Appl. Mech. Engrg.** , v. 303, p. 400-429, 2016.

10. GAY NETO, A.; PIMENTA, P. M.; WRIGGERS, P. A Master-surface to Master-surface Formulation for Beam to Beam Contact. Part II: Frictional Interaction. **Comput. Methods Appl. Mech. Engrg.**, v. 319, p. 146-174, 2017.

11. GAY NETO, A.; PIMENTA, P. M.; WRIGGERS, P. Contact between rolling beams and flat surfaces. **Int. J. Numer. Meth. Engng**, v. 97, p. 683-706, 2014.

12. GAY NETO, A.; PIMENTA, P. M.; WRIGGERS, P. Contact between spheres and general surfaces. **Comput. Methods Appl. Mech. Engrg.**, v. 328, p. 686-716, 2018.

13. ZAVARISE, G.; WRIGGERS, P. Contact with friction between beams in 3-D space. **Int. J. Numer. Meth. Engng.**, v. 49, p. 977-1006, 2000.

14. GAY NETO, A. **Modelagem computacional do contato pontual entre corpos:** uma visão integrada. São Paulo: University of São Paulo, v. Habilitation Thesis (in Portuguese), 2018.

15. GAY NETO, A.; WRIGGERS, P. Numerical method for solution of pointwise contact between surfaces. **Submitted to CMAME**, 2020.

16. WRIGGERS, P. **Nonlinear Finite Element Methods**. Berlin Heidelberg: Springer-Verlag , 2008.

Appendix

Selection by element properties in Giraffe data using Paraview™

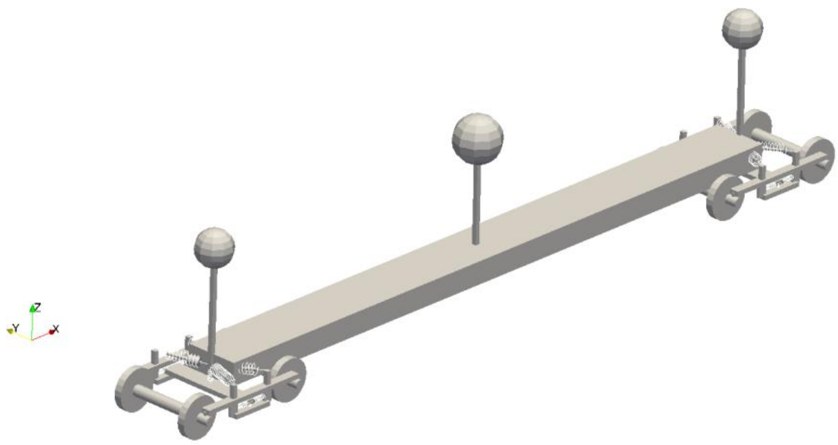
Problem statement:

Imagine that you have data results for a processed simulation in Giraffe, containing many kinds of elements, material ID's (numbers), etc.

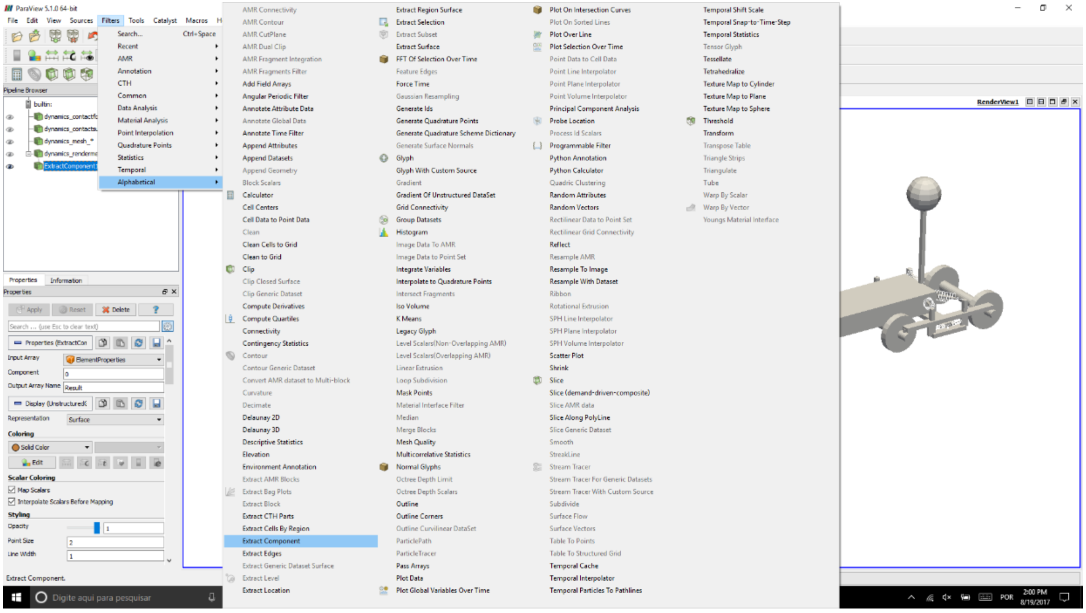
A hypothetic post-processing scenario is proposed: you would like to create a plot only containing cells, filtered by element type, material properties ID, or to a more complex extract based on data information. This is possible by using Paraview's filters.

Step-by-step procedure:

1. Select in the model pipeline browser the data you would like to operate with. In this example, we choose a render mesh data.



2. Create an "Extract Component" filter:



3. Choose the input array to guide the filtering operation.
   Giraffe writes "ElementProperties" array associated with cells in a render mesh visualization, which contains numbers following the meaning:

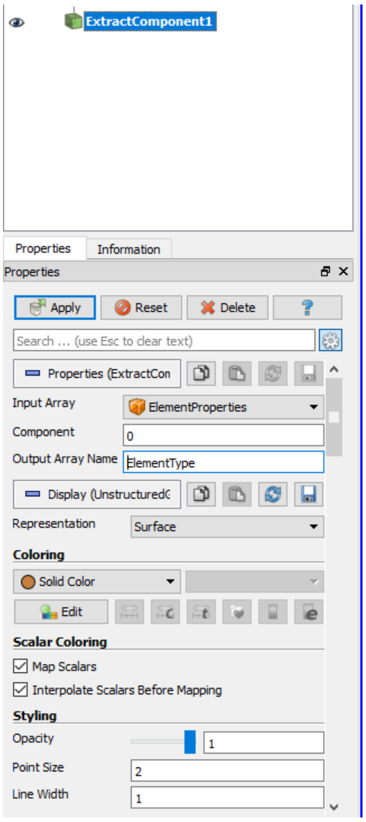
   0 – element type number

   1 – material ID

   2 – section ID

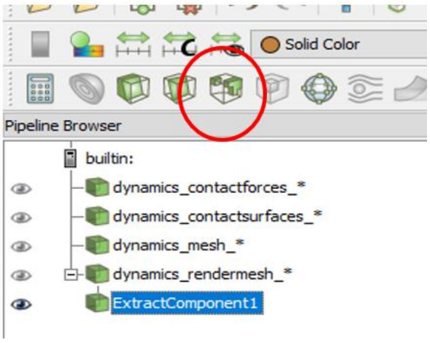
   3 – coordinate system ID

   In our example, we are interested in element type number, since we want to select only a given type of element. Choose a name for the output of the filter. In our example, we chose "ElementType". Click "Apply" to make the filtering operation have effect.

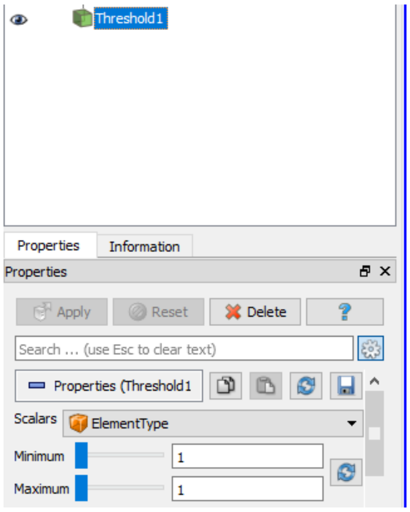


Now we need to extract only the "Beam_1" elements from the just-filtered data. Element types numbering follow

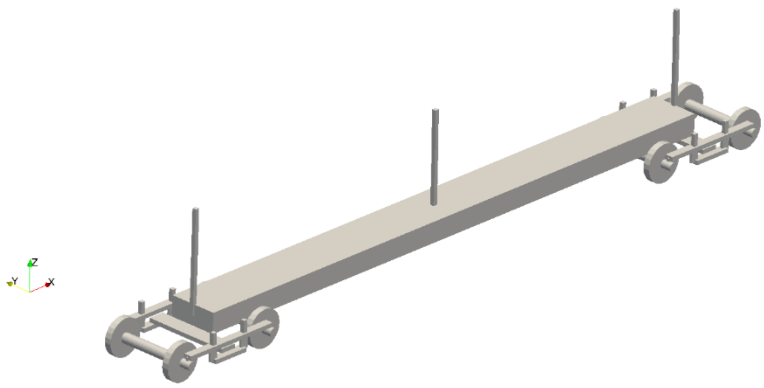
Table 7. This is done by the “Threshold” filter. Create it to operate in data just-filtered in previous step:



4. In “Threshold” filter properties, choose the scalar to guide the new selection. In this example, “ElementType”, just created in previous step. Then, choose the Minimum/Maximum values to control the new selection range, based on the scalar selected. In our case, we are interested only in number 1 – associated to Beam_1 elements.



5. The new plot will filter only Beam_1 elements:



Alternatively, one may follow another steps by running a Python routine in Paraview™:

Step-by-step procedure:

1. Select in the model pipeline browser the data you would like to operate with. In this example, we choose a render mesh data.
2. Click in Tools->Python Shell.
3. Copy and Paste the following python routine in the Python Shell and press enter.

(script available in Giraffe Releases/Documentation/Giraffe&Paraview/selection_script.py)

```
######################################################################
#Python script for selection using Giraffe ElementProperties data#
######################################################################

src = GetActiveSource()                          #obtains the active source of data (selection in pipeline browser)
filt1 = ExtractComponent()                       #creates a filter in variable 'filt1'
filt1.InputArray = 'ElementProperties'           #assigns 'ElementProperties' as the InputArray to 'filt1'
filt1.Component = 0                              #assigns the index '0' (ElementType) as the Component to 'filt1'.
filt1.OutputArrayName = 'ElementTypeNumber'      #assigns 'ElementType' as the OutputArrayName to 'filt1'
f = Threshold()                                  #creates a filter
filt1.UpdatePipeline()
SetActiveSource(src)                             #sets the original source of data
filt2 = ExtractComponent()                       #creates a filter in variable 'filt2'
filt2.InputArray = 'ElementProperties'           #assigns 'ElementProperties' as the InputArray to 'filt2'
filt2.Component = 1                              #assigns the index '1' (MaterialNumber) as the Component to 'filt2'.
filt2.OutputArrayName = 'MaterialNumber'         #assigns 'MaterialNumber' as the OutputArrayName to 'filt2'
f = Threshold()                                  #creates a filter
filt2.UpdatePipeline()
SetActiveSource(src)                             #sets the original source of data
filt3 = ExtractComponent()                       #creates a filter in variable 'filt3'
filt3.InputArray = 'ElementProperties'           #assigns 'ElementProperties' as the InputArray to 'filt3'
filt3.Component = 2                              #assigns the index '2' (SectionNumber) as the Component to 'filt3'.
filt3.OutputArrayName = 'SectionNumber'          #assigns 'SectionNumber' as the OutputArrayName to 'filt3'
f = Threshold()                                  #creates a filter
filt3.UpdatePipeline()
SetActiveSource(src)                             #sets the original source of data
filt4 = ExtractComponent()                       #creates a filter in variable 'filt4'
filt4.InputArray = 'ElementProperties'           #assigns 'ElementProperties' as the InputArray to 'filt4'
filt4.Component = 3                              #assigns the index '3' (CSNumber) as the Component to 'filt4'.
filt4.OutputArrayName = 'CSNumber'               #assigns 'CSNumber' as the OutputArrayName to 'filt4'
f = Threshold()                                  #creates a filter
filt4.UpdatePipeline()
SetActiveSource(src)                             #sets the original source of data
```

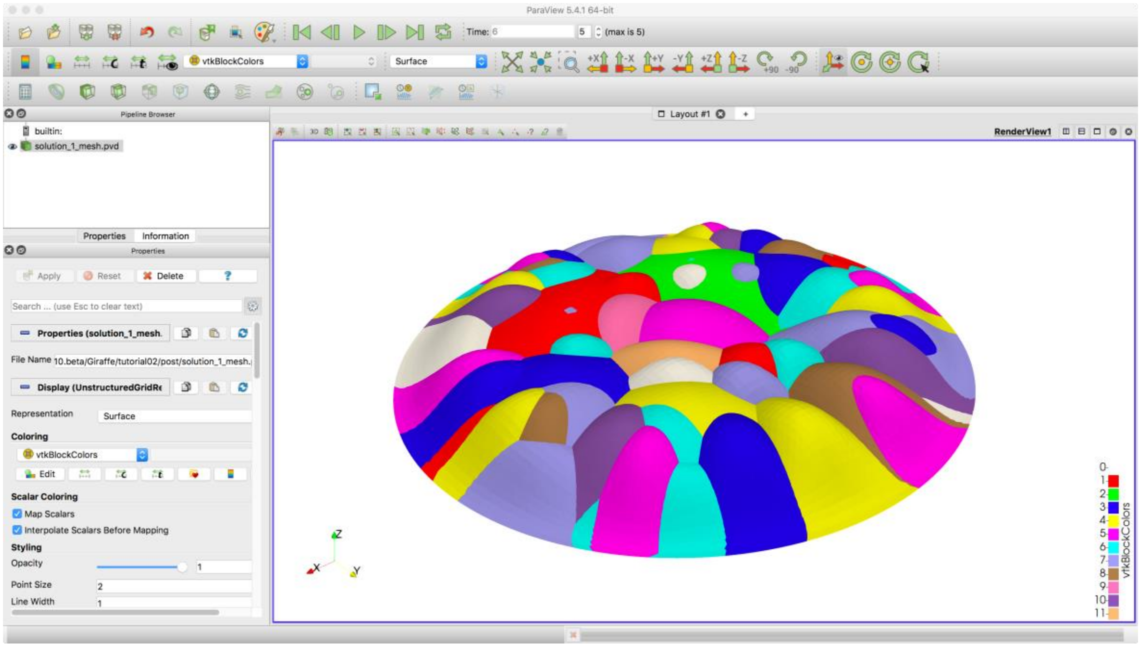
Post-processing modal analysis using Paraview™

Problem statement:

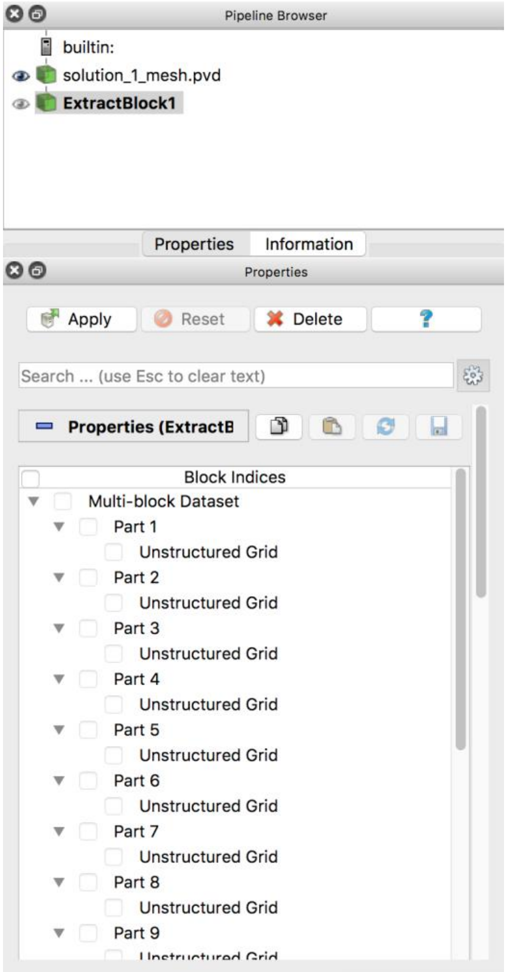
When performing a modal analysis as a solution step "i" the user may request the vibration modes evaluation. In this case, Giraffe saves result files on "/post/solution_i". These files are automatically loaded in Paraview™ by opening the related "solution_i_mesh.pvd" file, located in "/post" folder. Next, we show how to post-process vibration modes using Paraview™.

Step-by-step procedure:

1. Open in Paraview™ the file "solution_i_mesh.pvd" (for a given "i"). Click Apply button on the Pipeline browser. With that, all modes will be opened simultaneously. Paraview™ will show results like this:



2. We need to instruct Paraview™ to extract each mode of interest for plotting results. This is done by employing a filter named: "Extract Block". It can be activated by selecting in the model tree the file "solution_i_mesh.pvd". Then, go to:
   Filters->Alphabetical->Extract Block. Paraview™ will show a menu for the choice of the desired block, to post-process:



3. Each block encompasses results for a given vibration mode. For example, by selecting "Part 4" block and clicking "Apply", we are able to see and animate results for the vibration mode associated with the fourth eigenvalue found by Giraffe. This follows also for the other parts, associating the part number with the eigenvalue sequential number.