5. OVERFLOW-D-Mode Operation with Grid Movement

This section describes typical run sequences with OVERFLOW 2.3 in the OVERFLOW-D-mode with grid movement. As it does for OVERFLOW-D-mode without grid motion, the code goes through three steps before reaching the flow solver. First, it generates the overlapping Cartesian off-body grid system. Second, if the run uses MPI parallelization, OVERFLOW splits grids for load balancing and then assigns groups of grids to individual MPI processes. Each group is run in a separate process, with MPI calls facilitating communication between the groups. Third, hole cutting and interpolation for the overlapping grids are determined using the domain connectivity software DCF.

As described in the previous chapter, each X-ray is tied to a body, identified by "Comp ID" number in the GEN-X window of **overgrid**. This allows the X-ray to move with the body. Body or Component ID number *n* refers to the n^{th}-component defined in the **Config.xml** file for motions controlled with GMP. The Body or Component ID number refers to the n^{th}-component defined in the FOMOCO input file for motions controlled using NAMELIST **&SIXINP**. This Body ID also matches **IBLINK** supplied for each grid in **&SIXINP**.

The NAMELIST **&OMIGLB** input **DYNMCS** is set to TRUE to allow near-body grid motion. It is not possible to move the automatically generated Cartesian off-body grids. User-defined level-1 regions specified in NAMELIST **&BRKINP** may move if they are associated with a body using the input parameter **IBDYTAG** (see Chapter 4).

Grid motion is specified using the **&OMIGLB** NAMELIST input parameter **I6DOF**. There are three options in OVERFLOW 2.3 for performing grid motion based on **I6DOF**.

I6DOF = 0 Body motion is performed based on the user supplied subroutine **omisoft/sixdof/user6.F**. The user must create the subroutine to describe the motion for this option. The code must be recompiled with the desired subroutine included. This method allows access to most of the moving body variables inside the code and can be used for prescribed motion or a six-degree-of-freedom (6-DOF) model. This requires a good knowledge of the workings of the code. Several examples are included in **omisoft/sixdof/user6.F**. Details of this method will not be included in this manual. This option allows the user to add an external 6-DOF model or to add more complex physics such as grid deformation.

I6DOF = 1 Body motion is performed based on the input in NAMELIST **&SIXINP**. This method supports 6-DOF motion only and does not allow for prescribed motion. Motion is determined based on the integrated aerodynamic forces and moments on the body plus a user-specified external force or moment. Body motion is relative to the global coordinate system only for this option. Bodies may only be simply connected (i.e., parent–child). Only simple motion constraints are allowed.

I6DOF = 2 Body motion is performed based on the Geometry Manipulation Protocol[1] (GMP) XML files **Config.xml** and **Scenario.xml**. This method allows for prescribed motion and/or 6-DOF motion. 6-DOF motion is determined based on the integrated aerodynamic forces and moments on the body plus user-specified external forces and moments. The XML files allow for hierarchical specification of the relationships between bodies, and thus allow for relative motion between bodies. Because of the additional flexibility allowed with GMP this is the recommended method for moving-body simulations in OVERFLOW 2.3. The XML files can be created in, and prescribed motion of the body may be previewed using **overgrid**.

Note: while the XML interface appears to allow considerable flexibility, the 6-DOF integration routines in OVERFLOW 2.3 are limited. Testing of the desired functionality for a new application is highly recommended. Further, the motion previewing capability in **overgrid** and the motion routines in OVERFLOW 2.3 may not agree on the interpretation of the XML files, though this should be fixed in Chimera Grid Tools 2.1. One method of testing in OVERFLOW 2.3 is to set **ITER**=0 and **ITERT**=0 for all grids, turning off the flow solver and turbulence model.

The non-dimensional position, orientation, forces, and moments of a moving body will be written to the **sixdof.out** and **animate.out** files. The format for both files is given in Appendix A.

5.1 Body Motion Non-Dimensionalization

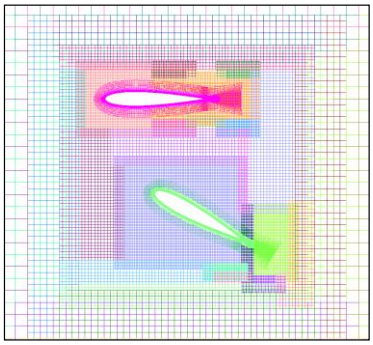
Non-dimensionalization of the NAMELIST input quantities used by the 6-DOF models for **I6DOF**=1 or **I6DOF**=2 is based on V_{ref} (reference velocity), L (length one in the computational grid), and ρ_∞ (free-stream density). The reference velocity may be specified using the **&FLOINP** NAMELIST parameter **REFMACH** if it is different from the free-stream Mach number (**FSMACH**). The non-dimensional quantities in the input are defined as

Length:	$l^* = 1/L$
Mass:	$m^* = m/(\rho_\infty L^3)$
Velocity:	$V^* = V/V_{ref}$
Time:	$t^* = t(V_{ref}/L)$
Acceleration:	$a^* = a\,(L/V_{ref}^2)$
Force:	$F^* = F/(\rho_\infty V_{ref}^2 L^2)$
Moment-of-inertia:	$I^* = I/(\rho_\infty L^5)$
Angular velocity:	$\omega^* = \omega(L/V_{ref})$
Moment:	$M^* = M/(\rho_\infty V_{ref}^2 L^3)$

The moments of inertia for a body are defined in the initial body coordinate system. Two examples of non-dimensionalization for moving body problems are shown below.

Example 5.1

Non-Dimensionalization Example: Airfoil Drop



- Assume standard sea-level conditions:
  - ρ_∞ = 0.002378 slug/ft^3
  - c_∞ = 1117 ft/sec
  - Gravity = 32.2 ft/sec^2
- Pick airfoil properties:
  - chord = 1 ft
  - weight = 30 lb (heavy!)
- Flow conditions:
  - Mach = 0.2
  - Re/chord = 1 million
- From these we have:

dimensional		non-dimensional		
L	= 1 ft	L*	= 1	(grid is in chords)
Vref	= 223.4 ft/sec	Vref*	= 1	
g	= 32.2 ft/sec^2	g*	= 645x10^{-6}	
Wt	= 30 lb	Wt*	= 0.2528	
mass	= 0.9317 slug	mass*	= 392	
Iyy	= 0.05054 slug-ft^2	Iyy*	= 21.25	

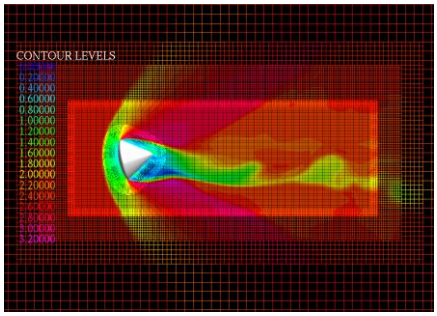
- And pick (so that 400 steps is 0.1 sec):

Δt	= 0.00025 sec	Δt*	= 0.05585

Example 5.2

Apollo Ballistic Range Model

- Ballistic range model properties:
  - diameter = 63 mm
  - mass = 575.9 g
  - (Ixx, Iyy, Izz) = (0.1833, 0.1761, 0.1761)x10^6 g-mm^2
- Assume standard sea-level conditions:
  - ρ_∞ = 1.226 g/mm^3
  - c_∞ = 0.3405x10^6 mm/sec
  - Gravity = 9807 mm/sec^2
  - μ_∞ = 1.781 g/mm-sec
- Flow conditions:
  - Mach = 2.5
  - Re/mm = 58,610/mm



- From these we have:

dimensional		non-dimensional		
L	= 1 mm	L*	= 1	(grid is in mm)
Vref	= 0.8512x10^6 mm/sec	Vref*	= 1	
g	= 9807 mm/sec^2	g*	= 13.53x10^{-9}	
mass	= 575.9 g	mass*	= 469.7x10^6	
Ixx	= 0.1833x10^6 g-mm^2	Ixx*	= 149.5x10^9	
Iyy = Izz	= 0. 1761x10^6 g-mm^2	Iyy* = Izz*	= 143.6x10^9	

5.2 Six-Degree-of-Freedom Motion Using &SIXINP

Dynamic motion can be simulated by utilizing the SIXDOF module. The NAMELIST **&SIXINP** input **IGMOVE** determines if a near-body grid is stationary (0) or moving (1). The input **IBLINK** specifies the Body ID that this grid is part of, and guarantees that each grid that is part of a body undergoes the motion of that body. The body component grids will be moved to the new position determined by the SIXDOF module. The body number used by OVERFLOW 2.3 will coincide with the order of the components used for the force and moment calculation with FOMOCO or USURP. The list of components can be found at the end of the **mixsur.fmp** file. If NAMELIST **&OMIGLB** input **DYNMCS**=.FALSE., but **IGMOVE**=1, initial movement can occur with the appropriate inputs, but the grids will then remain static for the remainder of the simulation. Thus it is not possible to turn off *all* grid motion using **DYNMCS**=.FALSE. only. **IGMOVE** may also need to be set to 0 if initial translation or rotation parameters are present.

The initial position of a grid can be specified using the NAMELIST **&SIXINP** inputs **(X00,Y00,Z00)**, **(X0,Y0,Z0)**, and **(E1,E2,E3,E4)**. **(X00,Y00,Z00)** defines the body center of gravity in the body coordinate system, **(X0**,**Y0**,**Z0)** is the location of the body center of gravity in the global coordinate system, and **(E1,E2,E3,E4)** are the Euler parameters (quaternion) that describe the rotation of the body about the center of gravity in the global coordinate system. This may be thought of as a rotation followed by a translation. The Euler parameters are defined by

$$
\begin{aligned}
E1 &= \cos(\tfrac{1}{2}\psi)\cos(\tfrac{1}{2}\theta)\sin(\tfrac{1}{2}\phi) - \sin(\tfrac{1}{2}\psi)\sin(\tfrac{1}{2}\theta)\cos(\tfrac{1}{2}\phi) \\
E2 &= \cos(\tfrac{1}{2}\psi)\sin(\tfrac{1}{2}\theta)\cos(\tfrac{1}{2}\phi) + \sin(\tfrac{1}{2}\psi)\cos(\tfrac{1}{2}\theta)\sin(\tfrac{1}{2}\phi) \\
E3 &= \sin(\tfrac{1}{2}\psi)\cos(\tfrac{1}{2}\theta)\cos(\tfrac{1}{2}\phi) - \cos(\tfrac{1}{2}\psi)\sin(\tfrac{1}{2}\theta)\sin(\tfrac{1}{2}\phi) \\
E4 &= \cos(\tfrac{1}{2}\psi)\cos(\tfrac{1}{2}\theta)\cos(\tfrac{1}{2}\phi) + \sin(\tfrac{1}{2}\psi)\sin(\tfrac{1}{2}\theta)\sin(\tfrac{1}{2}\phi)
\end{aligned}
\qquad (5.1)
$$

where ψ, θ, and ϕ are the yaw (rotation about the *z*-axis), pitch (rotation about the *y*-axis), and roll (rotation about the *x*-axis), respectively. These angles are defined for a yaw-pitch-roll active rotation of the body. E4 is the real part of the quaternion. The rotation matrix *A* can be determined from the Euler parameters by

$$
\begin{aligned}
a11 &= 1-2E2^2-2E3^2 = \cos(\psi)\cos(\theta) \\
a12 &= 2(E1*E2-E3*E4) = \sin(\psi)\cos(\theta) \\
a13 &= 2(E1*E3+E2*E4) = -\sin(\theta)
\end{aligned}
$$

$$
\begin{aligned}
a21 &= 2(E1*E2+E3*E4) = \cos(\psi)\sin(\theta)\sin(\phi)-\sin(\psi)\cos(\phi) \\
a22 &= 1-2E1^2-2E3^2 = \sin(\psi)\sin(\theta)\sin(\phi)+\cos(\psi)\cos(\phi) \\
a23 &= 2(E2*E3-E1*E4) = \cos(\theta)\sin(\phi) \\
a31 &= 2(E1*E3-E2*E4) = \cos(\psi)\sin(\theta)\cos(\phi)+\sin(\psi)\sin(\phi) \\
a32 &= 2(E2*E3+E1*E4) = \sin(\psi)\sin(\theta)\cos(\phi)-\cos(\psi)\sin(\phi) \\
a33 &= 1-2E1^2-2E2^2 = \cos(\theta)\cos(\phi)
\end{aligned}
\tag{5.2}
$$

The yaw, pitch, and roll angles can be found from the rotation matrix as

$$
\begin{aligned}
\theta &= \sin^{-1}(-a13) \\
\psi &= \tan^{-1}(a12/a11) \\
\phi &= \tan^{-1}(a23/a33)
\end{aligned}
\tag{5.3}
$$

The physical properties of a moving body (mass, weight, moments of inertia, and center of gravity location) must be included for (at least) the first grid that makes up the moving body. The moving body initial specified forces, moments, and duration must also be included for the first grid that makes up the body. These forces and moments are assumed to act about the body center of gravity.

The OVERFLOW 2.3 NAMELIST input for the dropping airfoil case in Example 5.1 using the **&SIXINP** input is shown below. The drop begins after 800 time steps are used to initialize the carriage solution. Setting **ISHIFT**=800 defines the new zero iteration values for the start of the drop. The two airfoils share the same position in the **grid.in** file, so the second airfoil is translated down to its proper initial position using **(X0**,**Y0**,**Z0)**. The location of the center of gravity of the second airfoil is defined using **(X00**,**Y00**,**Z00)**. The direction for the gravity unit-vector is defined by **(GRAVX**,**GRAVY**,**GRAVZ)**.

```
&GLOBAL
      RESTRT= .T.,  NSTEPS= 1100,
      DTPHYS=0.05585, NITNWT=5,
      NQT   = 102,
/

&OMIGLB
      IRUN  = 0,  IBYMIN= 21,
      DYNMCS= .T.,  I6DOF = 1,
      LFRINGE = 2,
/
&DCFGLB DQUAL = 0.1,
/
&GBRICK
      DFAR  = 10,  DS    = 0.016,  CHRLEN= 1,
      XNCEN = 0.5,  YNCEN = 0,  ZNCEN = 0,
      OFRINGE = 2,
/
&BRKINP /
&GROUPS /
&XRINFO IDXRAY= 1,  IGXLIST= 2,-1,  XDELTA = 0.04, /
&XRINFO IDXRAY= 2,  IGXLIST= 1,-1,  XDELTA = 0.04, /
&XRINFO IDXRAY= 3,  IGXLIST= -1,  XDELTA = 0.0, /

&FLOINP
      ALPHA = 0,  FSMACH= 0.2,  REY   = 1.E6,
/
&VARGAM /

&GRDNAM NAME  = 'Airfoil_1', /
&NITERS /
&METPRM
```

```
      IRHS   = 5, ILIMIT = 3, ILHS   = 6,
/
&TIMACU ITIME = 0,
/
&SMOACU FSO = 3,
/
&VISINP /
&BCINP
      IBTYP =   5, 51, 21,
      IBDIR =   2,  2,  3,
      JBCS  =  25,  1,  1,
      JBCE  = -25, 24, -1,
      KBCS  =   1,  1,  1,
      KBCE  =   1,  1, -1,
      LBCS  =   1,  1,  1,
      LBCE  =  -1, -1,  1,
/
&SCEINP /
&SIXINP
      ISHIFT = 800,
      IBLINK = 1,
      IGMOVE = 0,
/

&GRDNAM NAME  = 'Airfoil_2', /
&NITERS /
&METPRM /
&TIMACU /
&SMOACU /
&VISINP /
&BCINP
      IBTYP =   5, 51, 21,
      IBDIR =   2,  2,  3,
      JBCS  =  25,  1,  1,
      JBCE  = -25, 24, -1,
      KBCS  =   1,  1,  1,
      KBCE  =   1,  1, -1,
      LBCS  =   1,  1,  1,
      LBCE  =  -1, -1,  1,
/
&SCEINP /
&SIXINP
      ISHIFT = 800,
      IBLINK = 2,
      IGMOVE = 1,
      BMASS  = 392.0,
      WEIGHT = 0.02528,
      X0 = 0.25, Y0 = 0.0, Z0 = -0.3,
      X00 = 0.25, Y00 = 0.0, Z00 = 0.0,
      TJJ = 0.0, TKK = 21.3, TLL = 0.0,
      GRAVX = 0.0, GRAVY = 0.0, GRAVZ = -1.0,
/

&GRDNAM NAME  = 'Off-body grids', /
&NITERS /
&METPRM /
&TIMACU /
```

```
&SMOACU /
&VISINP /
&BCINP  /
&SCEINP /
```

5.3 GMP XML Specification

Body motion may also be specified using the GMP interface, by setting the NAMELIST **&OMIGLB** parameters **I6DOF**=2 and **DYNMCS**=.TRUE.. If **DYNMCS**=.FALSE., the code will preposition the bodies according to the **Config.xml** input file, but will not begin the motion. Nothing is specified in the **&SIXINP** NAMELIST when running in this mode (**&SIXINP** may be omitted from the input file).

The following text refers to the Geometry Manipulation Protocol (GMP) in general, and does not imply that all capabilities discussed are implemented in OVERFLOW 2.3. Only a subset of GMP is implemented in OVERFLOW. This section is copied in large part from Ref. 1.

GMP[1] is a protocol for specifying geometric hierarchies and their rigid-body motions. This protocol takes the form of a general set of datatypes and rules which can be implemented through any desired syntax, with the current choice being the Extensible Markup 1 Language[2] (XML). This low-level XML implementation is then "wrapped" with an Application Programming Interface (API). With this interface between the geometry motion and the application tools (such as the CFD flow solver), it is possible to build automated tools for performing dynamic simulations. The specification is suitable for simple analytic prescribed motions, as well as complex N-body problems with collisions and controller feedback. A fixed specification for the geometry motion allows multiple application programs, such as visualization tools, flow solvers, and post-processing tools, to be built upon a common interface. The geometry motion can be stored in a single repository, and shared among distributed applications, which minimizes errors due to duplication.

The interface specification relies heavily upon a generic function parser which is capable of parsing and interpreting arbitrary analytic functions. These analytic functions can take an arbitrary number of arguments. The parser understands the common mathematical operators and precedence rules, such as "(), ^, _, /,%,+,−", common constants such as π, and most commonly used functions such as "abs, log, sin, sqrt, tanh, . . . ". For example, pitch rate ($\alpha(t) = 10.0 \sin(2\pi t)$) for an oscillating airfoil is expressed as **10.0*sin(2*pi*t)**, and can be evaluated at run-time by providing an appropriate scalar value to substitute for the variable t. This substitution mechanism is provided by the function parser API. Within the GMP, an analytic function which takes no arguments replaces the role of a scalar value, i.e. a single scalar, or numeric value, is not an explicit type. In this manner, it is possible to use variables and constants which are appropriate to the problem, making the interface easier to specify. For instance, an angle of rotation can be specified as pi/4, as opposed to 0.7854, and the result will be evaluated at run-time by the application using the API for the function parser. In the current specification, a nomenclature is adopted to describe the analytic function datatype, and optionally the arguments which are expected. All numeric fields are specified as an arbitrary function which takes no arguments, f(). If an analytic function datatype is expected to take an argument of time in the interface, it will be described using f(t). A vector of 3 numeric fields, such as is used to describe a position, is specified as **vector: f()**.

The complete geometry which is being simulated is referred to here as a Configuration. Before a motion can be specified, it is necessary to describe the Configuration, so that a user can simply describe the motion of "the left rotor", as is intuitive, rather than being forced to refer to some application-specific geometry description. Instead of tightly coupling the Configuration information with a motion specification, the means of specifying a Configuration and the means of specifying its motion are separated. The motion specification is then built by referring to the Configuration. This allows different motions to easily refer to the same Configuration, as well as provides the ability to build separate tools which extend the Configuration.

Within GMP, the Configuration description is stored in an XML file named **Config.xml**. A typical Configuration is often made up of lower-level pieces, referred to here as Components. A simplified representation of the V-22 tilt-rotor geometry is shown in Fig. 5.1, with the different Components highlighted by color. The V-22 is made up of many Components, such as the fuselage, wing, empennage, rotors, etc. Many of these Components can also be further broken into smaller pieces, for example the rotors can be broken down into a nacelle, hub, and blades. This suggests that the Configuration is composed of a hierarchy, or tree, of Components. One possible hierarchy structure for the V-22 is shown schematically in Fig. 5.2. Notice also that this hierarchy is not unique, for example the rotors might be considered as a lower-level Component of the wing, or on the same level as the wing. These different hierarchies can become important when specifying the relative motion.

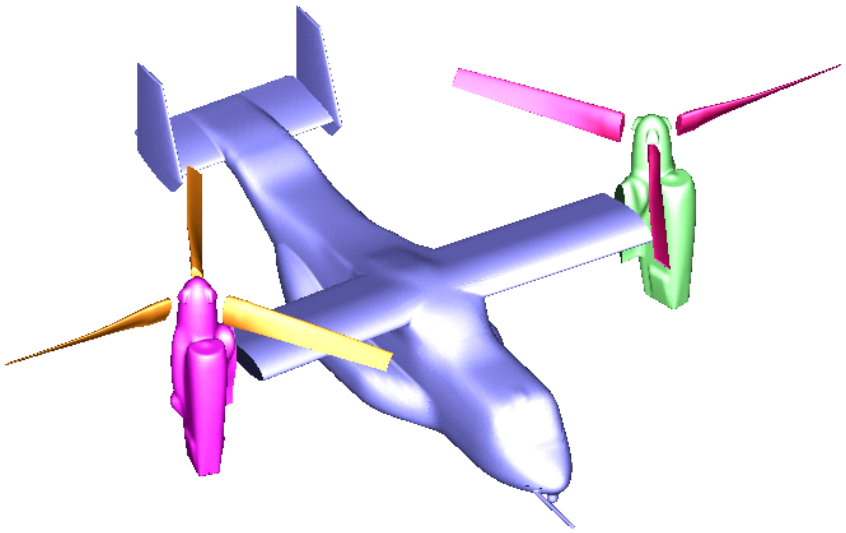


Figure 5.1 Example Configuration hierarchy for the V-22 tiltrotor.

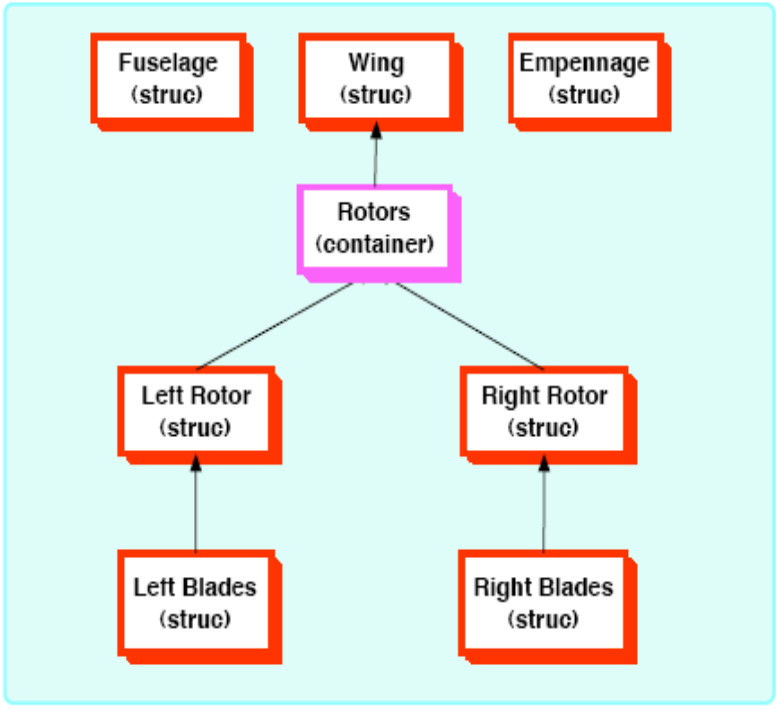


Figure 5.2 Example Configuration hierarchy for the V-22 tiltrotor. Component types are specified by color and text. Solid lines represent a parent-child relationship.

While the abstract hierarchy description of a Configuration is helpful, at some level it must be associated with the actual geometry that is to be manipulated. This is accomplished by requiring that each Component specify its Type, and optionally include some type-dependent Data. In order to promote flexibility within the Configuration specification, each type of Component is considered equal, and can be utilized anywhere within the Configuration hierarchy. Further, the Component types form an open-ended list which can be extended by future applications as needed. In other words, it is up to the external applications to determine which type of Components they can work

with and understand, not the specification, and similarly for the optional type-dependent Data. The tree diagram of Fig. 5.2 includes a label with the different Component types. These types are briefly described as

- struc: A set of structured grid (possibly overlapping) surface patches
- tri: A surface triangulation [not implemented]
- container: An agglomeration of lower-level Components
- clone: A duplicate of another (non-clone) Component [not implemented]

A Component of the Configuration hierarchy has the following complete list of attributes:

- Component:
  - Name [string] (required)
  - Type ["struc" or "container"] (required)
  - Parent [string] (optional)
  - Data [arbitrary] (optional)
    - Grid List [integers] (optional)
  - Transforms (optional)
    - Translate (optional)
      - Displacement [vector: f()] (required)
    - Rotate (optional)
      - Center [vector: f()] (required)
      - Axis [vector: f()] (required)
      - Angle [f()] (required)
    - Mirror (optional)
      - Plane ["X", "Y", or "Z"] (required)

The Parent attribute is used to specify the tree structure of the Configuration. The motion of a Component is usually specified relative to its Parent. Root nodes of the tree have no parents (the interpretation being that the inertial reference frame is the parent for motion), and multiple root nodes are allowed. The Grid List tag assigns near-body grid numbers from the **grid.in** file to the body Name that is specified. The Transforms tag is used if the Component is to be translated, rotated, or mirrored into position within the Configuration, and is made up of sub-types for specifying the actual transformations. [Note that mirroring is not useful for structured grids, since the resulting computational coordinates would be left-handed.] All coordinates used in the transformation are specified in the original, untransformed (natural) coordinate system of the appropriate geometry. Grids, as supplied in the grid.in file, are in their body coordinate frame. Transforms specified for each Component define the transformation from the Component's body frame to its parent frame. If the component has no Parent, then the transformation is to the inertial frame.

The clone Type can represent an exact duplicate, although in most instances the original is copied and then Transformed to a new position. For example, the cloned Component can be Mirrored about the x, y, or z = 0 plane for Configurations with lateral symmetry. Similarly, a single turbine blade can be cloned and Rotated to form a set of blades around a hub. In this manner errors due to duplication are reduced, and a common set of methods can easily be extended to an arbitrary number of Components.

Each motion specification refers to the Configuration description outlined in the previous section. The specific Components which are in motion are referred to by their Name attribute. The motion, or sequence of motions, is described by what is referred to as a Scenario, and is specified in an XML file named **Scenario.xml**. Scenarios are parameterized by time (t), starting at t = 0, with the units of time dependent upon the application. A Scenario is characterized by a number of actions, each occurring at a specific time, and for a specific duration. Currently, two types of actions can be specified; Prescribed motions, and Aero6DOF motions. These two types of motions are considered as distinct types as there is little commonality between them.

Both Config.xml and Scenario.xml files include additional information as attributes which apply to the entire file. For the Config.xml file, a Configuration type includes the AngleUnit attribute, which sets the units for the Rotate transformation Angle attribute. (Note that trigonometric functions still expect arguments in radians).

- Configuration:
  - Name [string] (optional)

  - AngleUnit [“radian” (default) or “degree”] (optional)

Similarly the Scenario.xml file has a Scenario type with several attributes. AngleUnit sets the units for Angle and Speed attributes for Prescribed and InitialPosition motions, as well as the Angle attribute used in Aero6DOF.

- Scenario:
  - Name [string] (optional)
  - AngleUnit [“radian” (default) or “degree”] (optional)
  - Gravity [vector: f()] (optional)

These types are included at the top of the appropriate file (see examples below).

5.4 GMP Prescribed Motion

The Prescribed motion can be specified as an arbitrary analytic function of time, or through a discrete table look-up [not implemented]. The analytic functions of time are parsed and evaluated by the stand-alone function parser. The time is interpreted as relative to the Start time of the current Prescribed motion [not in current versions of OVERFLOW 2.3], and a substitution of this current relative time is performed whenever the analytic functions are evaluated. This allows a Prescribed motion to be used multiple times within the same Scenario without modification. In order to specify the motion, either the position of a Component must be specified, or its velocity and initial position, though both position and velocity are usually needed by most CFD flow solvers. Since the initial position is available from the description of the Configuration, and it is easier to numerically integrate a function accurately than it is to differentiate, the analytic motion is Prescribed by providing the translational and angular velocities over the time period. The exception to this is the table look-up mode of operation, where the flexibility to specify only the position is allowed.

Motions are usually Prescribed relative to the parent of the Component within the Configuration hierarchy, and this is the default behavior. The motion is specified in the initial coordinate system of the input geometry (after any required Transforms have been applied within the Configuration specification). Prescribed motions have the following required and optional attributes:

- Prescribed
  - Component [string] (required)
  - Start [f()] (required)
  - Duration [f()] (optional)
  - Translate (optional)
    - Velocity [vector: f(t)] (required)
    - Frame [“parent” (default), “body”, or “inertial”] (optional)
  - Rotate (optional)
    - Center [vector: f()] (required)
    - Axis [vector: f()] (required)
    - Speed [vector: f(t)] (required)
    - Frame [“parent” (default), “body”, or “inertial”] (optional)

Start and Duration refer to the starting time and duration of the action. If the Duration is not specified the action is considered to be continued indefinitely. Prescribed motions are allowed to overlap in time intervals, and are ordered by their Start times. The Translate and Rotate commands specify the translational velocity of the center of mass of the component, and the rotation rate about an arbitrary axis through the center of rotation respectively. These commands are specified in the coordinates of the axis system specified by the Frame type. Choices for Frame are body, parent, or inertial, with the default being parent. Multiple Translate and Rotate commands can be combined within a single Prescribed action, and are applied in the order they are specified within the XML file.

In addition to Prescribed is an optional specification of InitialPosition. This can be used with either Prescribed or Aero6DOF motion, and allows the orientation of the Components within a dynamic simulation to be transformed after a Configuration has been “built.” This allows a general Configuration to be described, and then specialized if necessary for a dynamic simulation, i.e. it further decouples the Configuration and motion specification. Any Prescribed motion time level t = 0 is assumed to refer to the position of the body after the (optional) InitialPosition transforms have been applied. InitialPosition specifications have the following form and attributes; these transforms are all assumed to be in the parent frame:

- InitialPosition (optional)
  - Component [string] (required)
  - Translate (optional)
    - Displacement [vector: f()] (required)
  - Rotate (optional)
    - Center [vector: f()] (required)
    - Axis [vector: f()] (required)
    - Angle [f()] (required)
  - Mirror (optional)
    - Plane [“X”, “Y”, or “Z”] (required)

The XML files required for simulating a pitching airfoil are shown below. The free-stream Mach number is 0.29 and the reference temperature is 518.7°R. The airfoil has a one-foot chord. The grid unit is wing chords (1 foot). The airfoil motion is given by

$$\alpha = 4.0^{o}+4.2^{o}\sin(2\pi ft) \tag{5.4}$$

where the pitch oscillation is ten cycles per second. The **Config.xml** file is given by

Example 5.3 Config.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration AngleUnit="degree">
        <Component Name="WING" Type="struc">
        <Data>Grid List=1</Data>
        <Transform>
                <Rotate Center="0.25, 0.0, 0.0" Axis="0.0, 1.0, 0.0" Angle="-0.2" />
        </Transform>
        </Component>
</Configuration>
```

Here the rotation center for initial placement of the body is set at the quarter-chord location and the rotation is defined about the *y*-axis. The airfoil is initially oriented at the bottom of the oscillation (α=-0.2°). The motion is prescribed by the following **Scenario.xml** file:

Example 5.3 Scenario.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Scenario AngleUnit="degree">
        <Prescribed Component="WING" Start="6.323527">
                <Rotate Center="0.25, 0.0, 0.0" Axis="0.0, 1.0, 0.0"
                Speed="4.2*0.1940665*cos(0.1940665*t+1.5*pi)" Frame="parent" />
        </Prescribed>
</Scenario>
```

The motion will begin after a non-dimensional time of 6.323527. The airfoil will oscillate about the *y*-axis at the quarter-chord location in the parent coordinate system. The oscillation is specified in terms of the grid velocity in degrees per second. The frequency is non-dimensionalized using the grid reference length and the reference velocity. Time is non-dimensionalized in a similar manner.

A slightly more complicated example is a wing/flap configuration in which the wing and flap are oscillating while the flap is oscillating relative to the wing. The **Config.xml** file for this case is shown below.

Example 5.4 Config.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration AngleUnit="degree">
        <Component Name="Wing" Type="struc">
                <Data>Grid List=1,2</Data>
```

```
        </Component>
        <Component Name="Flap" Parent="Wing" Type="struc">
                <Data>Grid List=3</Data>
                <Transform>
                        <Rotate Center="0.81, 0.0, 0.0" Axis="0.0, 1.0, 0.0" Angle="0.0" />
                </Transform>
        </Component>
</Configuration>
```

Here the Wing is defined as containing grids 1 and 2. The Flap contains grid 3 and is a child of the Wing. Hence the Flap will inherit the motion of the Wing along with its own prescribed motion. The **Scenario.xml** file for this case is

Example 5.4 Scenario.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Scenario AngleUnit="degree">
        <Prescribed Component="Wing" Start="0">
                <Translate Velocity="0.0, 0.0, 0.25*.05*cos(.05*t)" Frame="parent" />
        </Prescribed>
        <Prescribed Component="Wing" Start="0">
                <Rotate Center="0.25, 0.0, 0.0" Axis="0.0, 1.0, 0.0" Speed="5*.05*cos(.05*t)"
                Frame="parent" />
        </Prescribed>
        <Prescribed Component="Flap" Start="0">
                <Rotate Center="0.81, 0.0, 0.0" Axis="0.0, 1.0, 0.0" Speed="40*.1*cos(.1*t)"
                Frame="parent" />
        </Prescribed>
</Scenario>
```

The Wing motion has both an oscillating translation and orientation. The translation is in terms of the non-dimensional velocity. The Flap motion is specified to be relative to its parent, defined as the Wing in the **Config.xml** file. Other prescribed motion example files are included in Ref. 1 and in the **test** subdirectory of the code.

5.5 GMP 6-DOF Motion

With the exception of Start and Duration times, the specification of Prescribed and Aero6DOF motions have little in common, and hence are treated as separate types. A component cannot be specified as having both Prescribed and Aero6DOF motions overlapping in time. Once a Component has been specified to have an Aero6DOF motion, it is no longer considered to be a child of its Parent (if it had one), and becomes a root node, i.e., the Configuration specification becomes dynamic when Aero6DOF motions are considered. Aero6DOF motions contain the same Name, Start, and Duration types as Prescribed motions, but also contain sub-types for InertialProperties, AppliedLoad, Constraint, and Controller. These latter are treated as sub-types of an Aero6DOF type, as opposed to types of their own, in order to make them more general. For example, if the AppliedLoad was a type then it would need to refer to the Aero6DOF motion it applied to in some manner. The AppliedLoad type would then need to be modified each time it was applied to a different Component. By making AppliedLoad a sub-type, it is implicit which Component it applies to, and it is also possible to use the same AppliedLoad with multiple Components without modification. For example, if a store ejector is modeled, this ejector can be tested with different store geometries simply by referencing the appropriate XML code within the specification. Similar arguments apply to Constraint and Controller. AppliedLoad, Constraint, and Controller can be thought of as "modifiers" for the Aero6DOF type. In this manner it is possible to build a library of ejector models, feedback systems, etc., which can then be used within different simulations without modification.

The component names used in the FOMOCO input file (and echoed in **mixsur.fmp**) must be consistent with the Component names used in the XML GMP files (including case). Force and moment information is assigned to individual bodies based on these names.

The complete type map for an Aero6DOF motion is

- Aero6DOF
  - Component [string] (required)
  - Start [f()] (required)
  - Duration [f()] (optional)
  - InertialProperties (required)
    - Mass [f(t)] (required)
    - CenterOfMass [vector: f(t)] (required)
    - PrincipalMomentsOfInertia [vector: f(t)] (required)
    - PrincipalAxesOrientation (required)
      - Axis [vector: f()] (required)
      - Angle [f()] (required)
  - AppliedLoad (optional)
    - Start [f()] (required)
    - Duration[f()] (optional)
    - Frame [“parent”, “body” (default), or “inertial”] (optional)
    - Force [vector: f(t)] (optional)
    - Moment [vector: f(t)] (optional)
  - Constraint (optional)
    - Start [f()] (required)
    - Duration [f()] (optional)
    - Frame [“parent”, “body” (default), or “inertial”] (optional)
    - Translate [vector: f(t)] (optional)
    - Rotate [vector: f(t)] (optional)
  - Controller (optional) [not implemented]

The initial translational and rotational velocities are either zero if no Prescribed motions were in effect previously, or are equal to the Prescribed values. It is assumed the origin of the PrincipalAxes corresponds to the CenterOfMass location at the beginning of the Aero6DOF motion. The InertialProperties are allowed to be general functions of time, as is necessary to model a rocket burning fuel [not implemented]. It is the responsibility of the application to implement a suitable model for solving the 6DOF equations under these conditions. An AppliedLoad can be specified in three different coordinate frames: body, parent, and inertial. An example of a parent frame would be a pylon ejector force for a store separation. A constant thrust could be modeled using an AppliedLoad in the body frame.

A Constraint on the other hand is always assumed to be relative to the parent frame. A Constraint is specified as either a Translate constraint, Rotate constraint, or both. The numerical inputs are bounded by 0 and 1, with 0 corresponding to unconstrained motion and 1 for no allowed motion relative to the parent system. The three components of the Constraint vector are the *x, y, z*-components of translation or rotation. Arbitrary functions of time can be specified for a Constraint or AppliedLoad. Controller types are specified as modifiers to the Aero6DOF motion, however they are currently left vague until more experience is gained with controlled, 6-DOF motions. Constraints are implemented in three stages:

1. As an external force and moment, applied such that the component should move in the desired fashion. An equal and opposite reaction force is automatically applied to the parent, if there is one. (Note that this is still under development.)
2. As corrected velocity and angular rates, again so that the component moves as desired.
3. As a corrected position.

A sample of a simple constraint is

Example 5.5

```
<Constraint Start="0.5" Duration="0.3" Type="simple" Frame="body"
      Translate="0,1,1" Rotate="1,1,1" />
```

This says that the body can translate in the (body) x-direction only, with no rotations.

The **Config.xml** input file for the dropping airfoil (Example 5.1) would be

Example 5.6 Dropping airfoil **Config.xml** file.

```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration AngleUnit="degree">
        <Component Name="Airfoil_1" Type="struc">
                <Data>Grid List=1</Data>
        </Component>
        <Component Name="Airfoil_2" Type="struc">
                <Data>Grid List=2</Data>
                <Transform>
                        <Translate Displacement="0.0, 0.0, -0.3" />
                </Transform>
        </Component>
</Configuration>
```

Here the second airfoil (Airfoil_2) is moved from its initial position in the **grid.in** file to a position 0.3 chords below the first airfoil. The **Scenario.xml** file would be

Example 5.6 Dropping airfoil Scenario.xml file.

```
<?xml version="1.0" encoding="utf-8" ?>
<Scenario Name="Airfoil Drop" Gravity="0.0, 0.0, -6.45E-4" AngleUnit="degree">
        <Aero6dof Component="Airfoil_2" Start="60000.0">
                <InertialProperties Mass="392.0" CenterOfMass="0.25, 0.0, 0.0"
                PrincipalMomentsOfInertia="0.0, 21.3, 0.0">
                <PrincipalAxesOrientation Axis="1.0, 0.0, 0.0" Angle="0.0" />
                </InertialProperties>
        </Aero6dof>
</Scenario>
```

Here the gravity vector, start time, and moments of inertia are all given as non-dimensional quantities. The airfoil is allowed to drop following 60000 non-dimensional time units.

A second example case is for a sphere that is given an upward velocity for the non-dimensional time period of t=0 to t=1. After t=1, the sphere is allowed to drop under the influence of gravity. The **Scenario.xml** file for this motion is

Example 5.7 Sphere motion Scenario.xml file

```
<?xml version="1.0" encoding="utf-8" ?>
<Scenario AngleUnit="radian" Gravity="0,0,-1.0">
        <Prescribed Component="Sphere" Start="0" Duration="1">
                <Translate Velocity="0.0, 0.0, 1.0" Frame="inertial" />
        </Prescribed>
        <Aero6dof Component="Sphere" Start="1.0">
                <InertialProperties Mass="1.0" CenterOfMass="0.0, 0.0, 0.0"
                PrincipalMomentsOfInertia="1.0, 1.0, 1.0">
                <PrincipalAxesOrientation Axis="1.0, 0.0, 0.0" Angle="0.0" />
                </InertialProperties>
        </Aero6dof>
</Scenario>
```

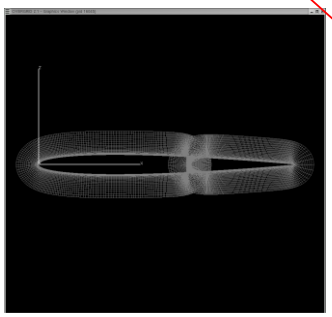
Further examples of the XML input files can be found in Ref. 1 and in the **test** subdirectory of the code.

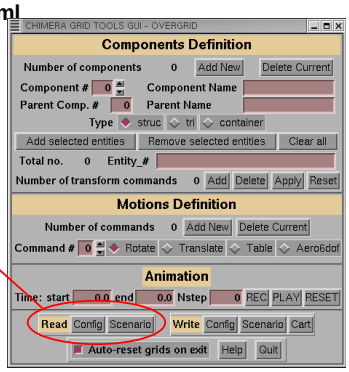
5.6 Using overgrid to Preview GMP Body Motion

The Chimera Grid Tools (CGT) graphical interface **overgrid** can be used to preview prescribed body motion and to visualize the resulting motion from 6-DOF body simulations. Examples of using **overgrid** to preview body motion and to visualize 6-DOF trajectories are shown below.

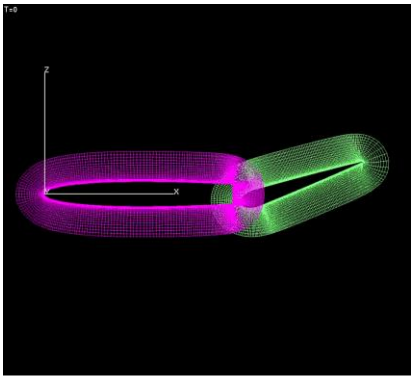
Example 5.8 Prescribed motion preview using overgrid

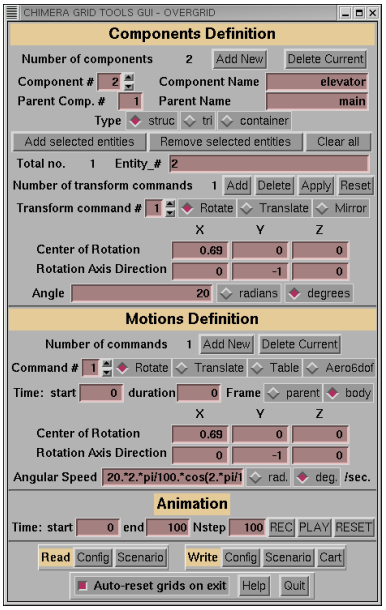
- Prescribed motion can be visualized in OVERGRID by reading in (surface grids or) **grid.in, Config.xml** and **Scenario.xml**
  - Start OVERGRID with surface grids or grid.in
  - Click “COMPONENTS”
  - On COMPONENTS menu,
    - Click Read “Config” (“OK”)
    - Click Read “Scenario” (“OK”)





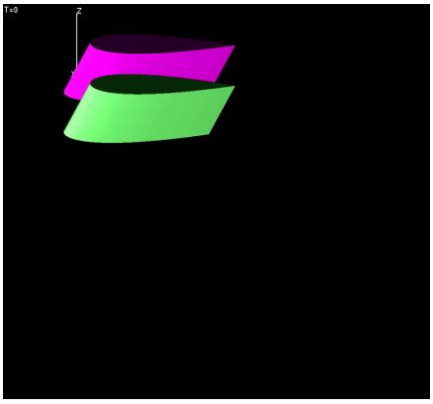
- Menu shows information on each component
  - Component names and hierarchy
  - Initial transforms from **Config.xml**
  - Prescribed motions from **Scenario.xml**
- Enter animation information
  - Start/end time and number of steps
  - Click “PLAY”

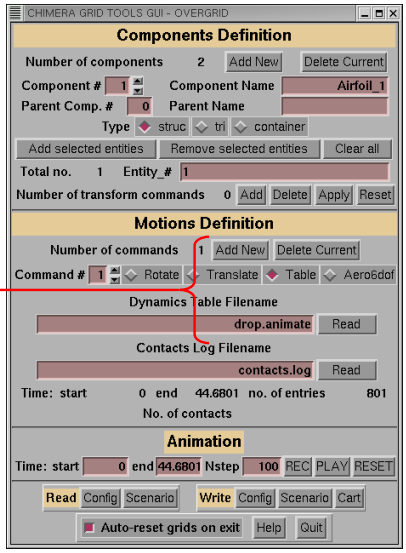




Example 5.9 6DOF motion visualization using overgrid

- For visualizing 6-DOF motion (after the OVERFLOW simulation is complete) read in **basename.animate**:
  - Click “Add New” motion command
  - Click “Table”
  - Type in animate filename and click “Read”
  - Click “PLAY”





5.7 Simulating Body Collisions

OVERFLOW 2.3 includes the capability to simulate the collision of solid bodies using the methodology described in Ref. 3. Contact between bodies is detected by using X-ray hole-cutting applied to surface grids of each body. The code considers a contact has occurred if one body cuts a hole in another body. Accurate geometric representation of collisions may require much finer X-rays than normally used for hole-cutting. To keep the DCF process from becoming very slow, it is a good practice to make the collision X-rays separate from the DCF hole-cutting X-rays. Contact detection is enabled (per body) by adding grid “0” to the **IGXLIST** in the X-ray cutter definitions. The NAMELIST input **R_COEF** in **&OMIGLB** sets the global coefficient-of-restitution to be used for collision dynamics. Note that the time of contact is only accurate to within the value of **DTPHYS** specified for the simulation. Below is an example of the X-ray input for collision detection and simulation.

Example 5.10 X-ray input specification for collision detection and simulation

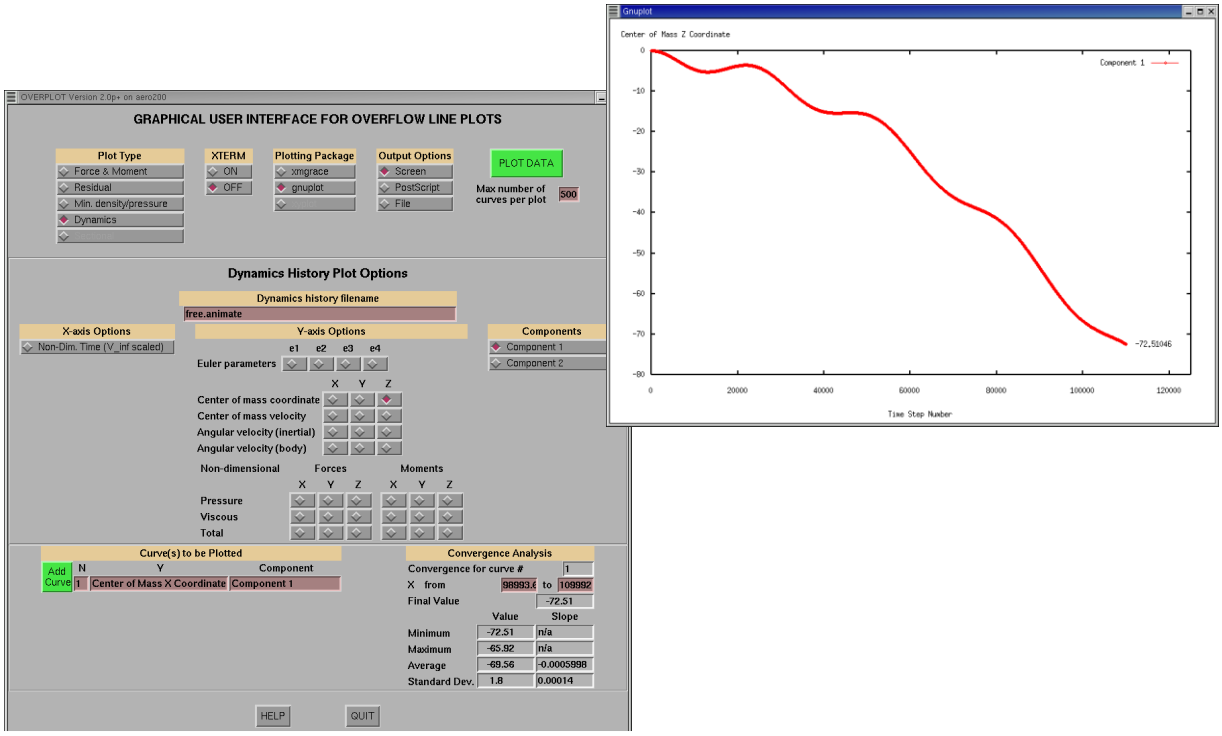
```
&XRINFO  IDXRAY=1, IGXLIST=2,0, XDELTA=0.0,  /
&XRINFO  IDXRAY=2, IGXLIST=1,0, XDELTA=0.0,  /
```

Information about each collision event is written to the **contact.out** file (see Appendix A).

5.8 Moving Body Output Files

All of the output files for non-moving body problems described in Chapter 3 and 4 are also output for moving body problems. The **q.step#**, **x.step#**, and **sixdof.step#** files are written out at the iteration number increment specified in the **&GLOBAL** NAMELIST parameter **NSAVE**. Two additional text files are also written: **animate.out** and **contact.out** (see Appendix A for file formats). The **fomoco.out** and **animate.out** files are written every time step for 6-DOF motion (**&GLOBAL NFOMO** is reset to 1). The **fomoco.out** file has force and moment coefficients non-dimensionalized by $\frac{1}{2}\rho_\infty V_{ref}^2 S_{ref}$ and $\frac{1}{2}\rho_\infty V_{ref}^2 S_{ref} L_{ref}$ respectively, where S_{ref} is the reference area and L_{ref} is the reference length for a given body specified in the FOMOCO input file. Moments are taken about the moment reference center specified in the FOMOCO input file. The **animate.out** file has forces and moments (not coefficients) non-dimensionalized by $\rho_\infty V_{ref}^2 L^2$ and $\rho_\infty V_{ref}^2 L^3$ respectively where L is the grid reference length. Moments are taken about the body center of gravity. The **overrun** script concatenates these files into **basename.{fomoco,animate,contact}**. Trajectories may be visualized using **overplot** as shown below.

Example 5.11 Trajectory plotting using overplot



5.9 Grid Adaptation to Body Motion

OVERFLOW 2.3 can adapt the off-body grid system to solution error and geometry. The original adaptation process is described in Refs. 4 and 5. The frequency of grid adaptation is determined by the NAMELIST input **NADAPT** in **&OMIGLB**. If **NADAPT**=0 no adaptation occurs. If **NADAPT** is set to a positive integer *n* then solution is adapted to solution error and geometry proximity every *n* time steps. (Solution adaption is described in Section 4.5.) If **NADAPT** is less than zero, adaptation takes place every -*n* steps based only on geometry proximity. Adaptation requires that the new off-body grid system interpolate starting values from the solution on the old grid system. One must insure that a body in motion does not leave the surrounding level-1 grids before the off-body grids are adapted. Since DCF does not cut holes in level-2 and higher grids, such a situation will result in excessive orphan points and inaccurate simulation.

References

1. Murman, S., Chan, W.M., Aftosmis, M.J., and Meakin, R.L., "An Interface for Specifying Rigid-Body Motions for CFD Applications", AIAA-2003-1237, Jan. 2003.
2. Harold, E.R., and Means, W.S., XML in a Nutshell: A Desktop Quick Reference, O'Reilly & Associates, Inc., 2001.
3. Meakin, R.L., "Multiple-Body Proximate-Flight Simulation Methods," AIAA-2005-4621, June 2005.
4. Meakin, R.L., "An Efficient Means of Adaptive Refinement Within Systems of Overset Grids," AIAA-95-1722, June 1995.
5. Meakin, R.L., "On Adaptive Refinement and Overset Structured Grids," AIAA-97-1858, June 1997.