Session 9:
Overset and 6-DOF Simulations

Bob Biedron

Learning Goals

- What this will teach you
  - Static and dynamic simulations using overset meshes (general)
  - Using FUN3D with (lib)SUGGAR++ for dynamic simulations
  - Setup for overset, 6DOF simulations
- What you will not learn
  - Setup and use of SUGGAR++ (stand-alone code; covered in another session)
- What should you already know
  - Basic time-accurate and dynamic-mesh solver operation and control

Part I – Overset Simulations

Setting

- Background
  - Many (most?) moving-body problems of interest involve large relative motion - rotorcraft, store separation are prime examples
    - Deforming meshes can accommodate only limited relative motion before mesh degenerates
    - Single rigid mesh can accommodate only one body, and not relative motion
    - Use overset grids to overcome these limitations - not to overcome complex geometry per se – that's why we use unstructured grids!
- Compatibility
  - FUN3D requires both DiRTlib and SUGGAR++ codes from PSU
  - Grid formats: VGRID, AFLR3, FieldView (FV)
- Status
  - Bodies in contact / emerging bodies - no near-term plans

Overset Mesh Simulations – General (1/4)

- Configuring FUN3D
  - Compile / install DiRTlib and SUGGAR; available scripts (download from FUN3D website) make it "easy"
  - When configuring FUN3D, use **`--with-dirtlib=/path/to/dirtlib`** and **`--with-suggar=/path/to/suggar`**
  - FUN3D will expect to find the following libraries in those locations:
    - **`libdirt.a, libdirt_mpich.a`** and **`libp3d.a`** (these may be soft links to the actual serial and mpi builds of DiRTlib)
    - **`libsuggar.a`** and **`libsuggar_mpi.a`** (may be soft links)
    - Scripts do this automatically – they put links to all archives in one spot, so **`/path/to/dirtlib = /path/to/suggar`**
- Grids (remember z is "up" for FUN3D)
  - A *composite* overset grid is comprised of 2 or more *component* grids - independently generated - but with similar cell sizes in the fringe areas
  - SUGGAR++ is used to create the composite mesh

Overset Mesh Simulations – General (2/4)

- Boundary conditions:
  - SUGGAR++ needs BC info for each *component* grid - set either via the SUGGAR++ input XML file OR an auxiliary file for each *component* grid; SUGGAR++ will output this auxiliary file for the *composite* mesh
  - FUN3D also needs BC info for the *composite* grid; depending on grid type, file names / content may differ slightly between FUN3D / SUGGAR

	VGRID grid	FV grid	AFLR3 grid
FUN3D	grid.mapbc (standard VGRID file)	grid.mapbc (***not*** same as VGRID)	grid.mapbc (***not*** same as VGRID)
SUGGAR++	grid.mapbc (standard VGRID file)	grid.ext.suggar_mapbc (***not*** same as VGRID)	grid.ext.suggar_mapbc (***not*** same as VGRID)

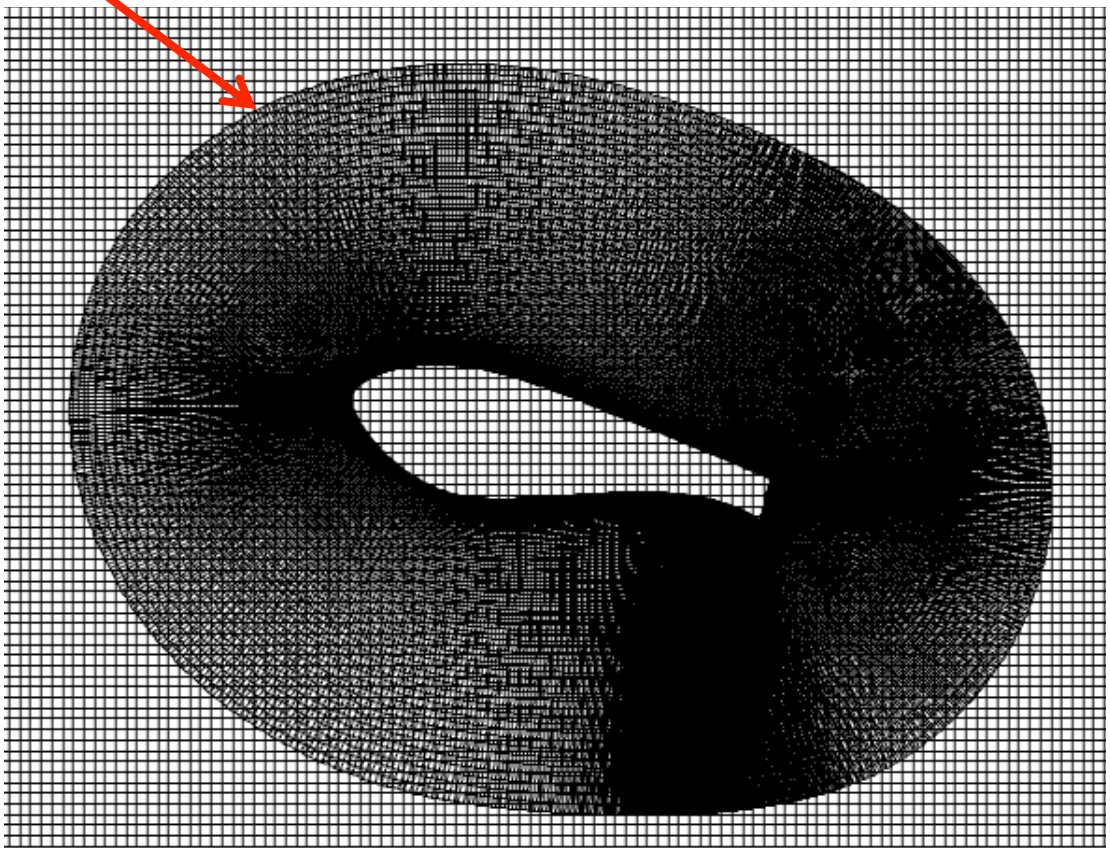
  - "ext" is the FUN3D grid extension, e.g.: grid.fvgrid_fmt, grid.r8.ugrid
  - AFLR3 / FV grids: suggar_mapbc file has extra column; FUN3D ignores

```
3                                ! number of boundaries (patches)
1 5000 Box          farfield ! patch_index, fun3d_bc, family_name, suggar_bc
2 4000 Wing_Surf    solid
3 -1   Wing_FarFld  overlap
```

Overset Mesh Simulations – General (3/4)

- Boundary conditions (cont):
  - set BC type to -1 in component-grid “mapbc” files for boundaries that are set via interpolation from another mesh

Grid Courtesy Eric Lynch, GA Tech



Overset Mesh Simulations – General (4/4)

- Create an XML input file for SUGGAR++
  - Basic SUGGAR++ setup covered in another session; however must show some XML here to show certain FUN3D-specific points
  - Set the name for the **`<composite_grid>`** and **`<domain_connectivity>`** files to the name of your FUN3D project
  - Can mix and match component grid types (VGRID, FV, AFLR) and select one of the types for the composite grid - but recall VGRID only supports tetrahedra
- Run SUGGAR++ and make sure it all works as expected. You should now have a **`[project].dci`** file; this **d**omain **c**onnectivity **i**nformation file contains all necessary overset data for solver interpolation between the nonmoving component meshes
- Good idea to use the “gviz” tool from PSU to view composite mesh assembly, holes points, fringe points, etc.

Overset Mesh Simulations – Static (1/2)

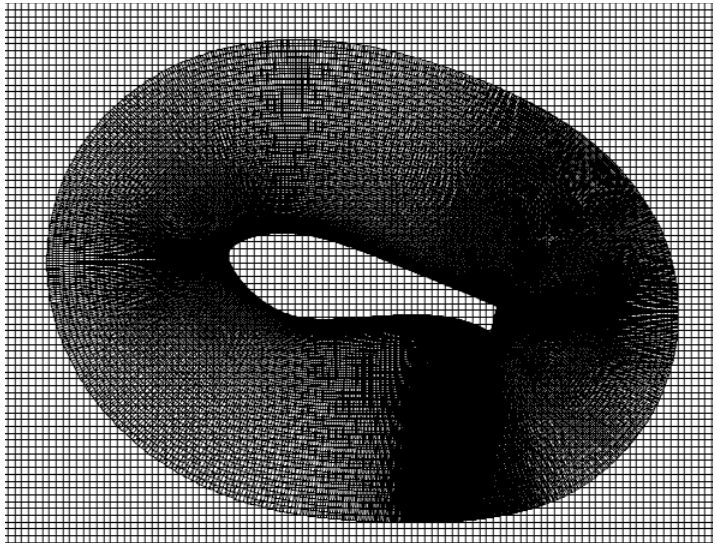
- Running FUN3D with static overset meshes:
  - Add **`--overset`** to any other CLOs you may have and run as usual
  - In screen output, should see:

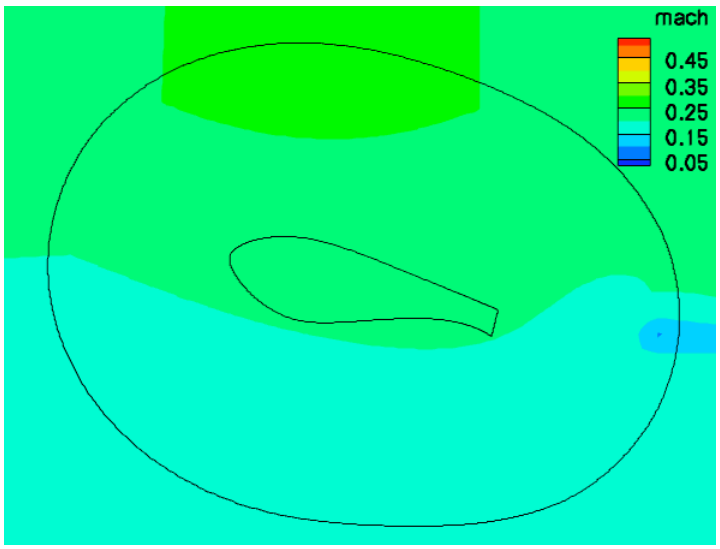
```
Reading DCI data:  ([project].dci)
Loading of dci file header took Wall …
Opening filename:      ([project].g21) (repeated nproc times !)
Loading of dci file took Wall Clock time = 5.324230 seconds
Using DiRTlib version 1.40 for overset capability
DiRTlib developed by Ralph Noack, Penn State University Applied Research
Laboratory
```

  - Followed by the usual FUN3D output, ending with **`Done.`**
  - If you request visualization output data for an overset case, “iblank” data will automatically be output to allow blanking of the hole / out points for correct visualization of the solution / grid in Tecplot

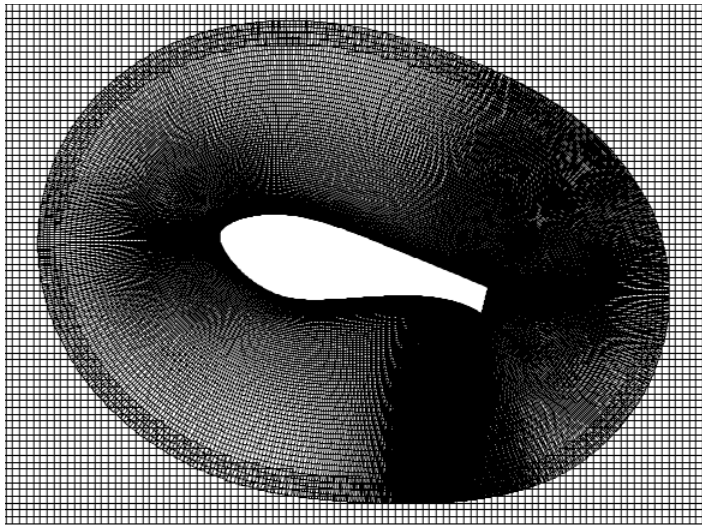
Overset Mesh Simulations – Static (2/2)

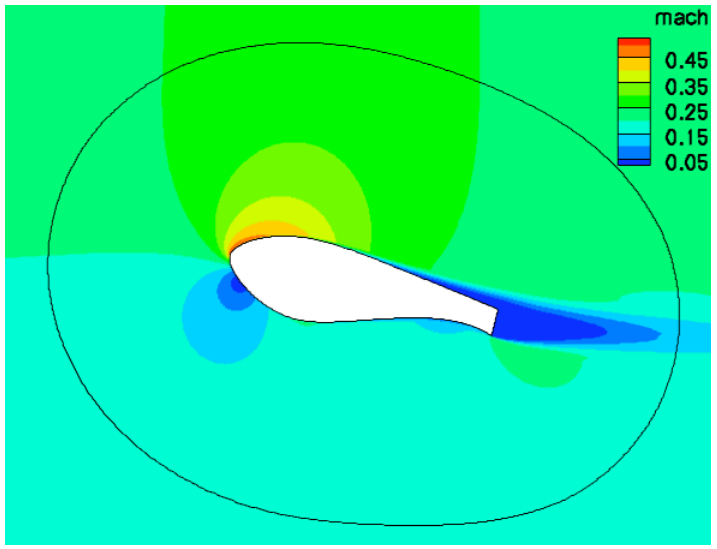
without iblank





with iblank





Overset Mesh Simulations – Dynamic (1/4)

- SUGGAR++ setup
  - Starting with a basic SUGGAR++ XML file:
    - Add **<dynamic/>** to **<body>** elements that are to move, e.g.

```
<body name="wing">
  <volume_grid name="wing" style="vgrid_set" filename="wing"/>
</body>
<body name="store">
  <dynamic/>
  <volume_grid name="store" style="vgrid_set" filename="store"/>
</body>
```

    - Note: better to use a self-terminated **<dynamic/>** rather than **<dynamic>** … **</dynamic>** since if there are any **<transform>** elements in between, SUGGAR++ won't apply them unless explicitly told to
  - Use SUGGAR++ to generate the initial (t = 0) composite grid; let's assume you called the XML file **Input.xml_0**

Overset Mesh Simulations – Dynamic (2/4)

- In the FUN3D **`moving_body.input`** file
  - Define the bodies and specify motion as usual; boundary numbers correspond to those in the *composite* mesh mapbc file, accounting for any boundary lumping that may be selected at run time
  - use the component body names from the **`Input.xml_0`** file
  - Add name of the xml file used to generate the t = 0 composite mesh:

```
&composite_overset_mesh
  input_xml_file = 'Input.xml_0'
/
```

- Running FUN3D
  - Use CLOs **`--overset --moving_grid --dci_on_the_fly`**
  - The last tells FUN3D to call libSUGGAR++ routines to compute new overset data when the grids are moved; if this CLO is not present, solver will try to read the corresponding dci file from disk

Overset Mesh Simulations – Dynamic (3/4)

- Running FUN3D (cont)
  - Note: for dynamic meshes, the *component* grids (and any "suggar_mapbc" files) must be available (can be soft linked) in the FUN3D run directory, in *addition* to the t = 0 *composite*-grid files
  - When using **`--dci_on_the_fly`**, must specify *one* additional processor for SUGGAR++ (in future, hope to be able to use more)
    - The *first* processor gets assigned the SUGGAR++ task
    - *This processor must have enough memory for entire overset problem* (same as needed for SUGGAR++ alone)
  - Other overset-grid CLOs

`--dci_period N` periodic motion over N steps (default **`0`**)

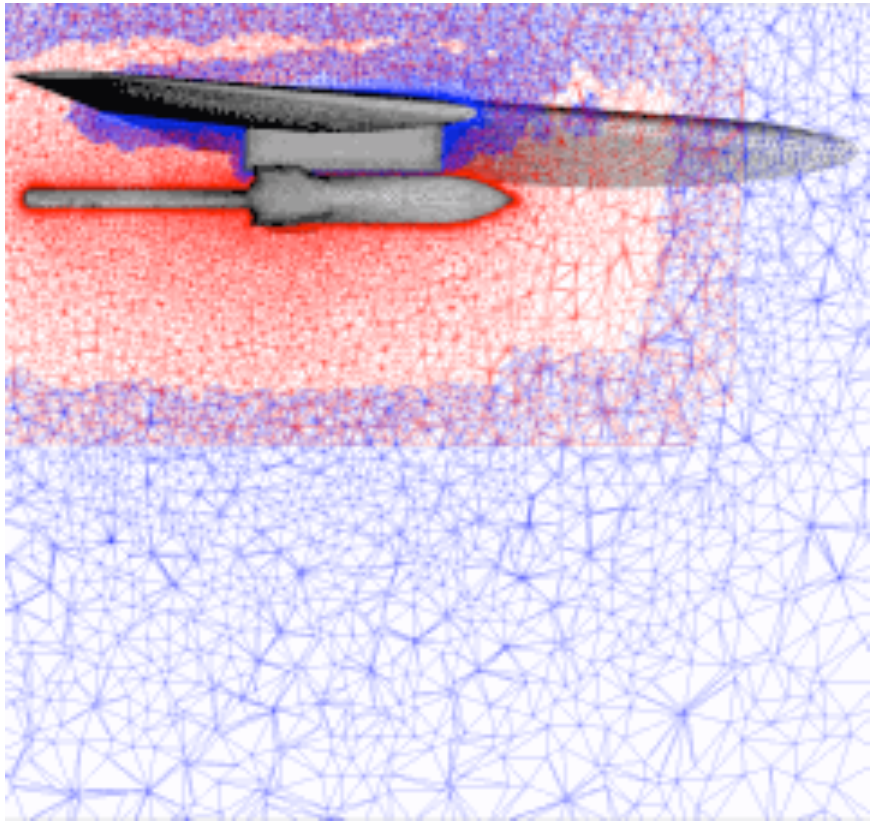
`--dci_freq N` compute dci data only every N^{th} step (**`1`**)

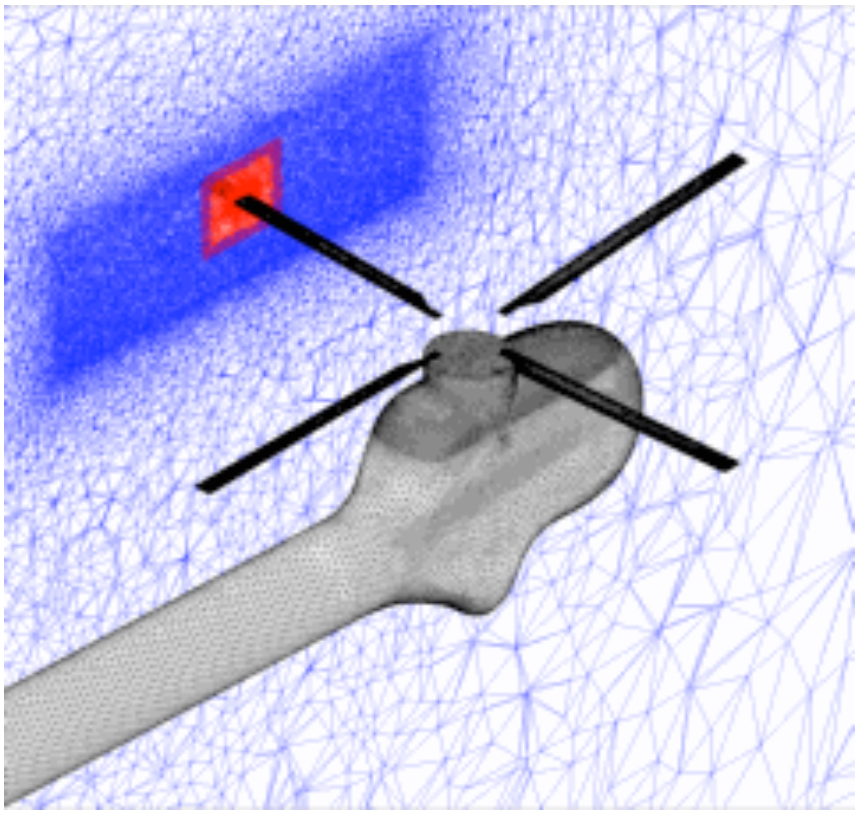
`--reuse_existing_dci` use existing files if present, even with **`--dci_on_the_fly (.F.)`**

`--grid_motion_and_dci_only` create dci files; no flow solve (**`.F.`**)

Overset Mesh Simulations – Dynamic (4/4)

- As always, can use animation to verify; these were done ex post facto, but GVIZ has motion replay options too





Part II – 6-DOF Simulations

Setting

- Background
  - FUN3D is currently coupled to the 6-DOF library originally developed by the Univ. of Alabama at Birmingham and Mississippi State Univ. under the DOD PET program
- Compatibility
  - Requires limited-availability library (available only to Government Organizations and Govt. Contractors working on a DOD contract)
  - Requires overset grids (DiRTlib and SUGGAR++)
- Status
  - 6-DOF capability in place but exercised very little to date - one or two validation cases - we simply are not working tasks which need 6-DOF (rotorcraft utilizes a very different 6-DOF capability)
  - Use Version 11.3 or higher - a couple of significant fixes for 6-DOF
  - Version 11.3 has a minor bug in 6-DOF module for the case where the grid is not scaled 1:1 with the full-sized configuration; fixed for v11.4

UAB 6-DOF Libraries (1/2)

- Originally developed by the Univ. of Alabama at Birmingham and Mississippi State Univ. under the DOD PET program
- Maintained and distributed by Nathan Prewitt Nathan.C.Prewitt@usace.army.mil
- General attributes
  - Multi body; hierarchical body definition
  - Allows for constrained motion (not yet implemented in FUN3D)
  - Allows for prescribed motion (e.g. specified motion of fins - not yet implemented in FUN3D)
  - Runge-Kutta 4th order time integration; quaternion based
  - Works with dimensional data
  - Rigid bodies only
- FUN3D user does not directly interact with the 6-DOF library, except to compile it and link against FUN3D; 6-DOF specific input primarily via FUN3D's **`moving_body.input`** file

UAB 6-DOF Libraries (2/2)

- Configuring FUN3D
  - Compile the 6DOF libraries, following the README that comes with the package. Top-level directory is called 6DOF (below that will be EXP, HT and Motion directories; you need to compile source via makefiles in each directory, as per the README file)
  - When configuring FUN3D, use **`--with-sixdof=/path/to/6DOF`**
  - FUN3D will expect to find the following libraries in those locations:
    - **`6DOF/Motion/lib/libmo.a`**
    - **`6DOF/HT/lib/libht.a`**
    - **`6DOF/EXP/lib/libexp.a`**
  - Recall that overset grids are required, so need **`--with-dirtlib=/path/to/dirtlib`** and **`--with-suggar=/path/to/suggar`** too
- Input for 6-DOF is a combination of nondimensional data (basic flow solver input via **`fun3d.nml`**: e.g. time step) and dimensional (e.g. mass and inertial properties via **`moving_body.input`**)

`moving_body.input` File (1/2)

- 6-DOF obviously has moving bodies so as usual need to group boundaries into moving bodies and since this is overset, need to set the initial XML file:

```
&body_definitions
  n_moving_bodies = 1,          ! number of bodies in motion
  body_name(1) = 'store',       ! name must be in quotes
  n_defining_bndry(1) = 3,      ! number of boundaries that define this body
  defining_bndry(1,1) = 5,      ! index 1: boundry number index 2: body number
  defining_bndry(2,1) = 6,      ! index 1: boundry number index 2: body number
  defining_bndry(3,1) = 7,      ! index 1: boundry number index 2: body number
  mesh_movement(1) = 'rigid',   ! 6DOF likely incompatible with deforming meshes (currently)
  motion_driver(1) = '6dof'     ! 6DOF is in the driver's seat
  dimensional_output = .true.   ! moving body history files will contain dimensional data
  body_frame_forces  = .true.   ! moving body F/M history output relative to body frame
  ref_velocity = 1011.7,        ! sound speed ft/sec at 26k ft - to dimensionalize for 6DOF
  ref_density = 0.00102,        ! slug/ft3 at 26k ft - to dimensionalize for 6DOF
  ref_length = 1.00             ! actually the length scale L*ref/Lref (1 unit in grid = 1 ft)
/                               ! bug in v11.3: grid MUST be scaled 1:1; fixed in v11.4
&composite_overset_mesh
  input_xml_file = 'Input.xml_0'    ! same as used to create composite mesh
/
```

`moving_body.input` File (2/2)

- Additional namelist for specifying body mass, inertia, external forces, etc

```
&sixdof_motion
  mass(1) = 62.1118,          ! body mass (slugs), body 1
  cg_x(1) = 1.483333333333, ! x-location of CG in body coordinates, body 1
  cg_y(1) = 10.83333333333, ! y-location of CG in body coordinates
  cg_z(1) = -2.95000000000, ! z-location of CG in body coordinates
  i_xx(1) = 20.0              ! Ixx momment of inertia, body 1
  i_yy(1) = 360.0,            ! Iyy momment of inertia
  i_zz(1) = 360.0,            ! Izz momment of inertia
  i_xy(1) = 0.0,              ! Ixy product of inertia
  i_xz(1) = 0.0,              ! Ixz product of inertia
  i_yz(1) = 0.0,              ! Iyz product of inertia
  body_lin_vel(:,1) = 0.0, 0.0, 0.0,  ! initial velocity (x,y,z components), body 1
  body_ang_vel(:,1) = 0.0, 0.0, 0.0,  ! initial ang. velocity (p,q,r components) of body 1
  euler_ang(1,1) = 0.0,       ! initial euler angle - yaw, body 1
  euler_ang(2,1) = 0.0,       ! initial euler angle - pitch
  euler_ang(3,1) = 0.0,       ! initial euler angle - roll
  gravity_dir(:) = 0.0, 0.0, -1.0   ! x,y,z components of gravity vector (z "up" in fun3d)
  gravity_mag = 32.2,         ! gravitational constant
  n_extforce(1) = 2,          ! no. of external forces applied, body 1
  file_extforce(1,1) = 'force_fwd_body1.dat' ! file with forward ejector force vs time
  file_extforce(2,1) = 'force_aft_body1.dat' ! file with aft ejector force vs time
/                                            ! similar provisions for external moments
```

External Force/Moment Specification

- Rudimentary provision for imposing ejector forces
- Input is dimensional, consistent with units used in **`moving_body.input`**
- Analogous format for imposed moment specification
- Example

```
! Body Name      ! must be consistent with name in moving_body.input
'store'
! Force Name
'fwd_ejector'
! Coordinate System (0 inertial, >0 body frame)
1
! Number of Data Points to Read
3
! Repeat Flag (0...last values remain forever; 1...repeat data)
0
! Time           Fx    Fy    Fz    Xloc         Yloc         Zloc
  0.0            0.0   0.0 -2400.0 0.893333333 10.833333333 -2.12
  0.055          0.0   0.0 -2400.0 0.893333333 10.833333333 -2.12
  0.05500000001 0.0   0.0     0.0 0.893333333 10.833333333 -2.12
```

Things To Look For In Screen Output (1/2)

- 6DOF info section starts with some useless info (from user point of view)

```
6DOF Initialization:

   Nondimesionalization factors for 6DOF equations:
   (6DOF force/moment nondim. differs from aerodynamics)
    inertia_factor      =  0.98039216E+03
    mass_factor         =  0.98039216E+03
    gravity_factor      =  0.97700436E-06
    length_factor       =  0.10000000E+01
    velocity_factor     =  0.98843531E-03
    time_factor         =  0.10117000E+04
    force_factor        =  0.95784741E-03
    moment_factor       =  0.95784741E-03
    body 1
     aero_force_factor   =  0.97295271E-03
     aero_xmoment_factor =  0.58377163E-03
     aero_ymoment_factor =  0.58377163E-03
     aero_zmoment_factor =  0.58377163E-03
```

Things To Look For In Screen Output (2/2)

- After which the user input is echoed:

```
Gravity Magnitude and Direction:   0.322000E+02  0.000000E+00  0.000000E+00
 -0.100000E+01

Dimensional 6DOF data for Body 1   Time =   0.0000000E+00
 Body Name:          store
 Mass:               0.621118E+02
 CG Location:        0.148333E+01  0.108333E+02 -0.295000E+01
 Ixx,Iyy,Izz:        0.200000E+02  0.360000E+03  0.360000E+03
 Ixy,Ixz,Iyz:        0.000000E+00  0.000000E+00  0.000000E+00
 Linear Vel:         0.000000E+00  0.000000E+00  0.000000E+00
 Angular Vel:        0.000000E+00  0.000000E+00  0.000000E+00
 Yaw,Pitch,Roll:     0.000000E+00  0.000000E+00  0.000000E+00

External forces for Body 1 imposed from the file(s) :
 force_fwd_body1.dat
 force_aft_body1.dat
```

- Note that CG location output here is in the body-frame, so generally won't differ at restart; however velocities and angular orientation are current values relative to the inertial frame, so will change at restart

Output Files

- In addition to the usual output files, for moving-grids there are 4 ASCII Tecplot files for each body; these are the primary 6-DOF data of interest
  - **`PositionBody_N.dat`** tracks linear (x,y,z) and angular (yaw, pitch, roll) displacement of the CG
  - **`VelocityBody_N.dat`** tracks linear (V_x,V_y,V_z) and angular (Ω_x,Ω_y,Ω_z) velocity of the CG
  - **`AeroForceMomentBody_N.dat`** tracks force components (F_x,F_y, F_z) and moment components (M_x,M_y,M_x)
  - **`ExternalForceMomentBody_N.dat`** tracks applied force F/M (6-DOF only)
  - Data in all files are nondimensional by default (e.g. “forces” are actually force coefficients); **`moving_body.input`** file has option to supply dimensional reference values such that *this* data is output in dimensional form (see previous example moving_body.input)
  - Forces are by default given in the inertial reference system; option to output forces in the body-fixed system (see previous example)

Sample Case - Wing/Pylon/Store (1/6)

- Only 6-DOF case computed to date is for the “classic” 1990 data set from the AEDC Aerodynamic Wind Tunnel (4T)
  - Mach 0.95 (data for Mach 1.2 also available)
  - Grid used was one created for a *cell-centered* solver, and so is actually inappropriate for FUN3D; also relatively coarse at 2.3M nodes
  - Grid includes a portion of the sting used in the tunnel, but sting contributions to forces/moments ignored (next slide)
  - Tunnel aerodynamic F/M data taken in a quasi-static manner
  - Trajectory based on full scale, 26k feet altitude, with ejector forces
  - Example **`moving_body.input`** file shown in previous slides correspond to this case, so won’t repeat here
  - Nondimensional time step of 5.0585 corresponds to 0.005 seconds; time-accurate solution started from converged steady-state solution
  - Large number of orphans for first 15 time steps or so (t ~ 0.075 sec); max 1544 orphans at 4th time step (t = 0.02 sec)

Sample Case - Wing/Pylon/Store (2/6)

- "Sting" was not metric in experimental force measurement; forces on these boundaries are excluded by having a file called **`remove_boundaries_from_force_totals:`**

```
File for turning off the contribution of selected boundaries
No. boundaries to turn off (be careful with boundary lumping)
6
Boundary to turn off
1
2
3
4
6
8
```

- Run Steady state case with CLO: **`--overset`**
- Run 6-DOF restart from steady state with CLO's:

```
--six_dof --dci_on_the_fly --overset --moving_grid
--temporal_err_control 0.01 --animation_freq +5
```

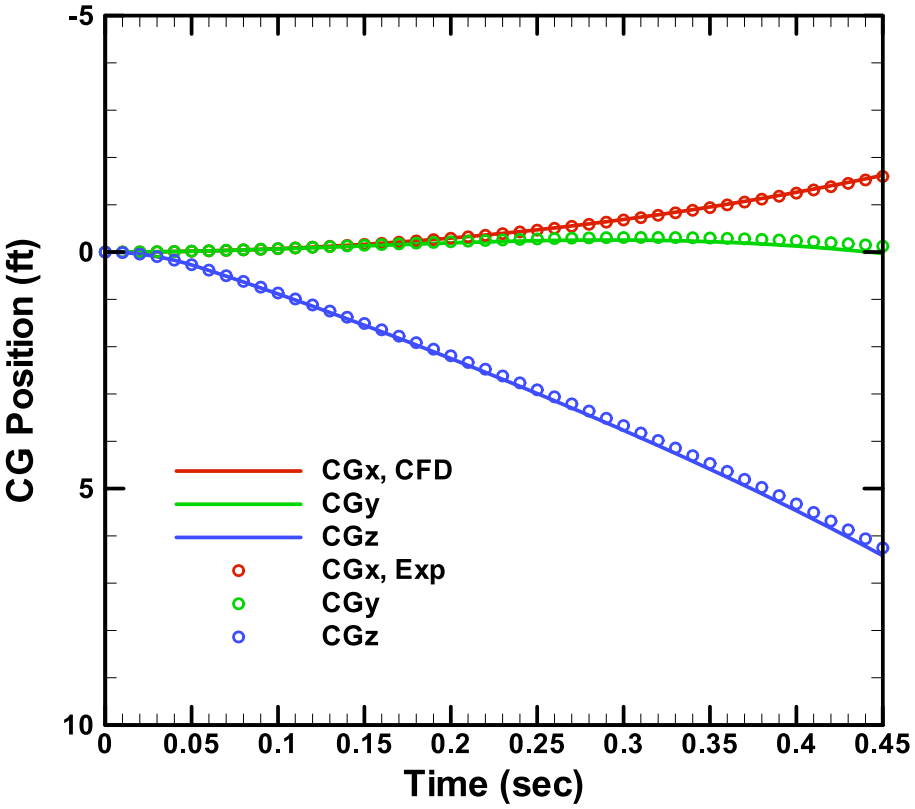
Sample Case - Wing/Pylon/Store (2/6)

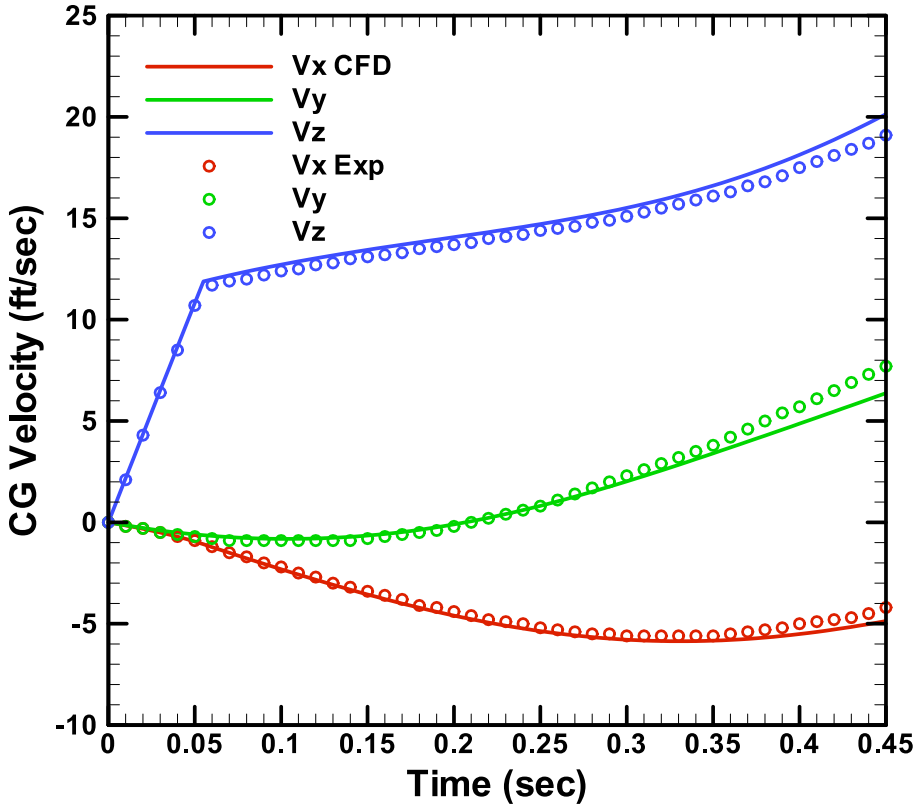
- SUGGAR++ XML file **`Input.xml_0`** (SUGGAR++ covered elsewhere)

```
<global>
   <donor_quality value="0.9"/>
   <symmetry_plane axis="Y"/>
   <minimize_overlap keep_inner_fringe="yes"/>
   <output>
     <unstructured_grid style="unsorted_vgrid_set" filename="wingstore"/>
     <domain_connectivity style="unformatted_gen_drt_pairs" filename="wingstore.dci"/>
   </output>
   <body name="wingstore">
     <body name="wing">
       <transform>
         <scale value= '1.6666666666667'/>
       </transform>
       <volume_grid name="wing" style="vgrid_set" filename="zx03wing"/>
     </body>
     <body name="store">
       <transform>
         <scale value= '1.6666666666667'/>
       </transform>
       <dynamic/>
       <volume_grid name="store" style="vgrid_set" filename="zx03bomb">
         <specified_donor_suitability_function value="2.e-20"/>
       </volume_grid>
     </body>
   </body>
</global>
```

Sample Case - Wing/Pylon/Store (3/6)

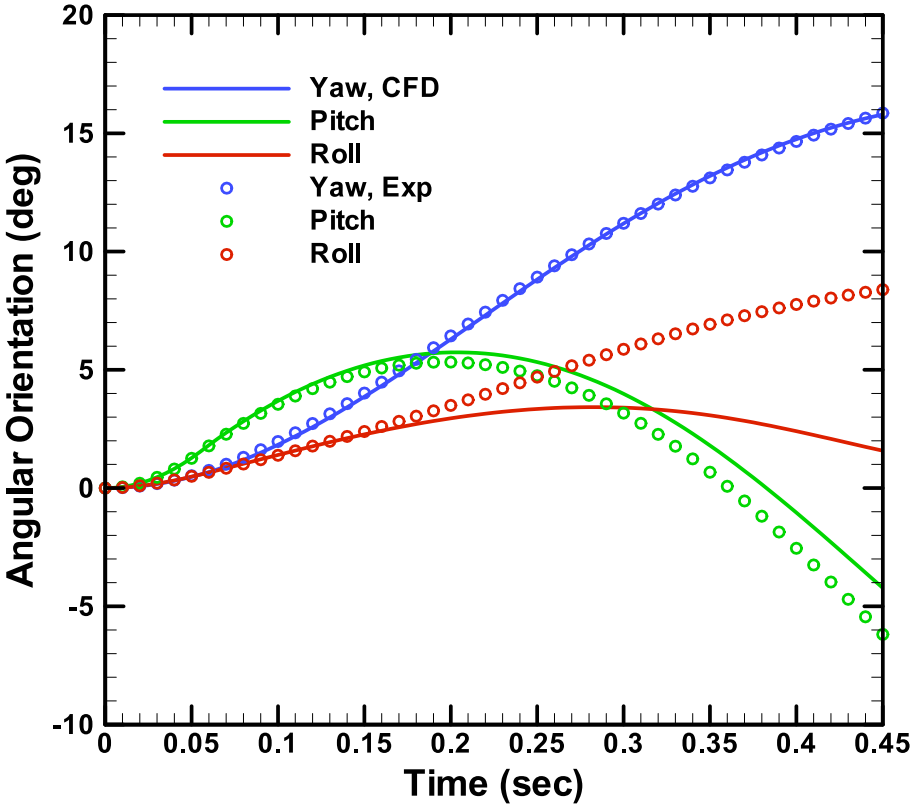
- Store Trajectory: CG position and velocity from **`PositionBody_1.dat`** and **`VelocityBody_1.dat`**

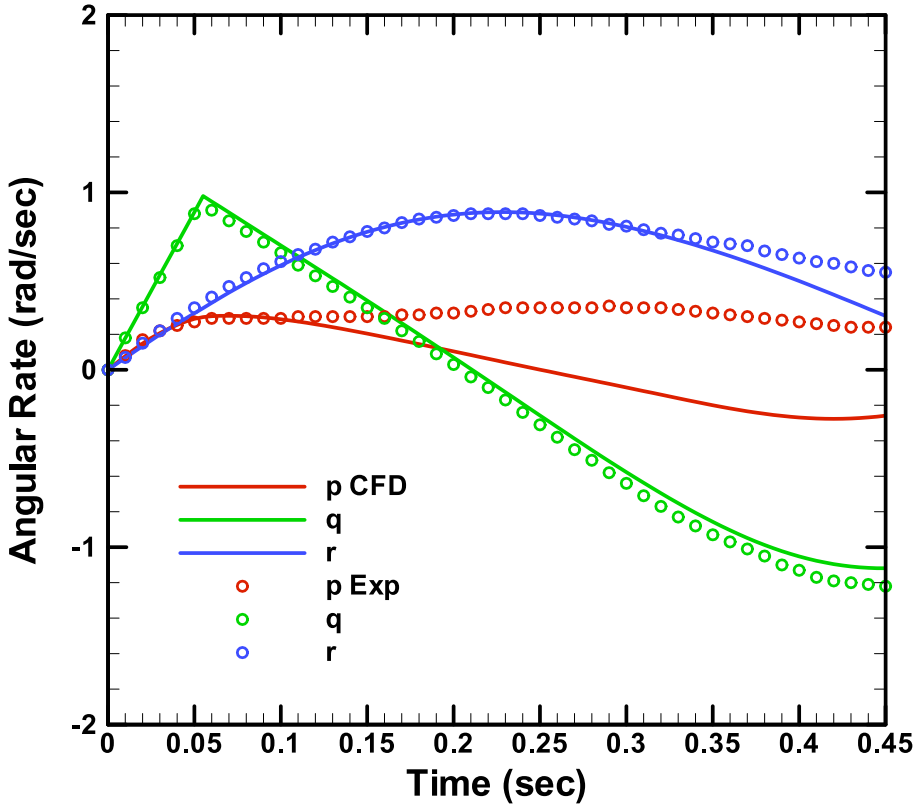




Sample Case - Wing/Pylon/Store (4/6)

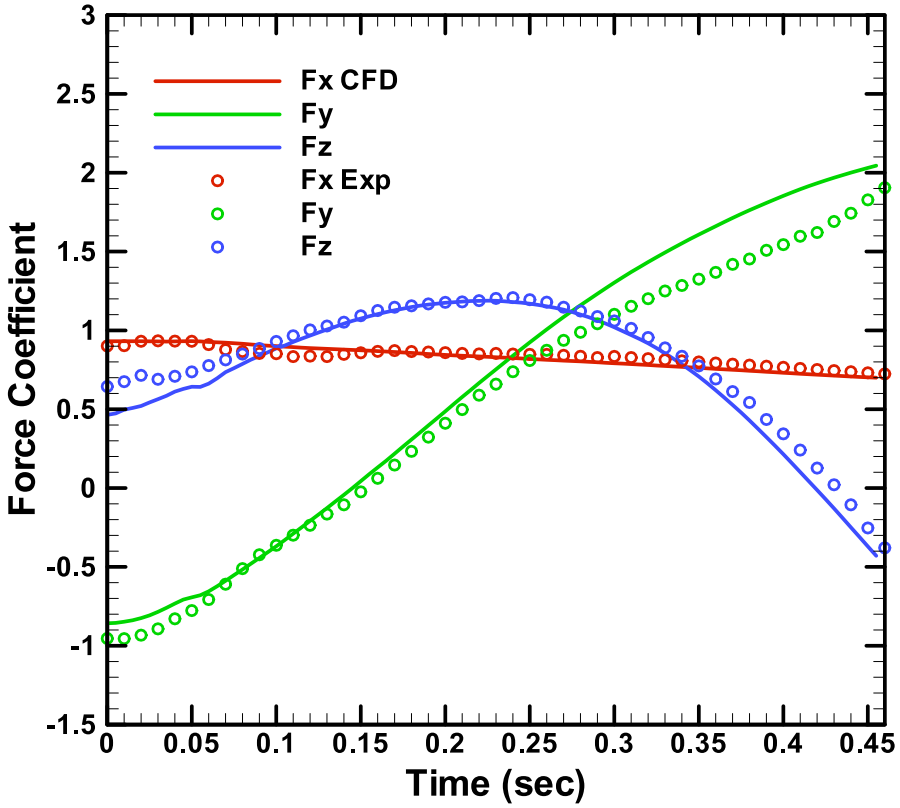
- Store Trajectory: angular orientation and angular rates from **`PositionBody_1.dat`** and **`VelocityBody_1.dat`**; low value of I_{xx} presumably makes roll more sensitive to force/moment inaccuracies

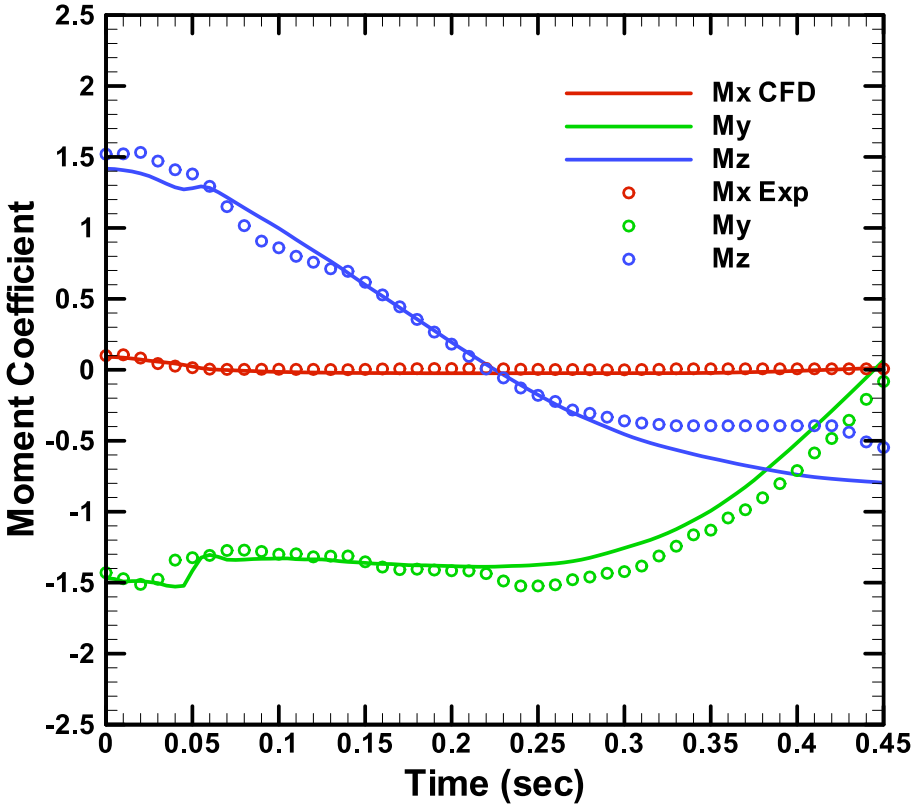




Sample Case - Wing/Pylon/Store (5/6)

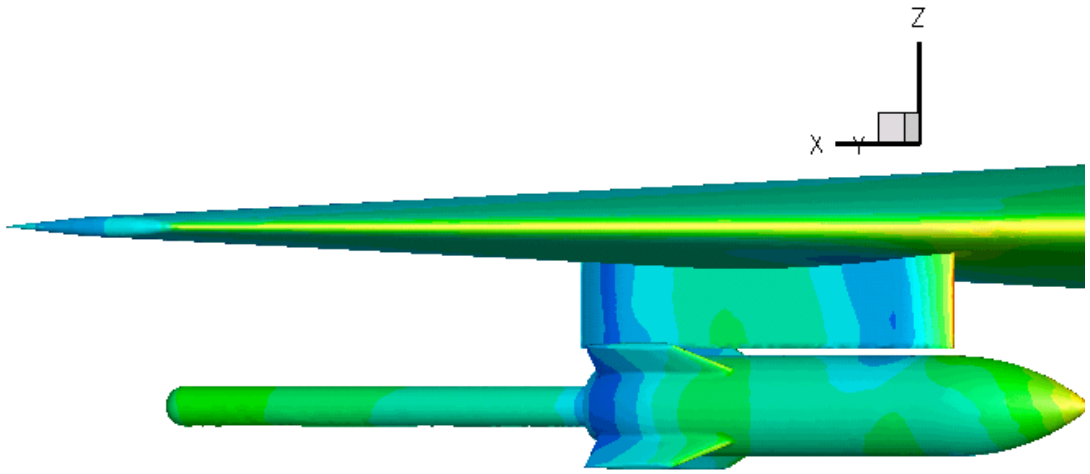
- Store Aerodynamics: force and moment coefficients - nondimensionalized from dimensional data in **`AeroForceMomentBody1.dat`**





Sample Case - Wing/Pylon/Store (6/6)

- Colorful Fluid Dynamics: pressure coefficient



List of Key Input/Output Files

- Beyond basics like **`fun3d.nml`**, **`[project]_hist.tec`**, etc.:
- Input
  - **`moving_body.input`** (any moving body case)
  - **`Input.xml_0`** (dynamic overset; no standard name)
  - **`[project].dci`** (any overset case)
  - **`force_fwd_body1.dat`** (optional, 6DOF only, no standard name)
- Output
  - **`PositionBody_N.dat`** (any moving body case)
  - **`VelocityBody_N.dat`** (any moving body case)
  - **`AeroForceMomentBody_N.dat`** (any moving body case)
  - **`ExternalForceMomentBody_N.dat`** (6DOF only)

FAQ's

- Underutilized capability, so not many "frequently" asked questions…
- How long does it take?
  - Currently (July 2010), the 2.3 million node Wing/Store/Pylon simulation (starting from a steady-state solution) takes approximately 2 hrs on 80(+1) processors of a 3.0 GHz P4 Dual Core 4GB GigE cluster (92 time steps using temporal_err_control 0.01 with max 50 subiterations); note that this case is small enough that a single processor for SUGGAR++ is not an impediment - not true as problem size increases
- Why don't I get any DCI files output from a 6-DOF case like I do from other overset, moving-grid cases?
  - 6-DOF cases are assumed to be non-periodic, so there would seem to be no need to reuse DCI data, hence no need to output them and waste file space - output can be turned on by altering a flag in the code if desired

What We Learned

- How to set up and run static and dynamic overset meshes in FUN3D
  - To fully utilize, requires knowledge of SUGGAR++ - covered in another session
- 6-DOF simulations
  - Modest amount of additional input required beyond that required for moving overset case with forced/specified motion
  - Reluctant to call this capability “ready for prime time” based on one or two results - but very willing to work with users to iron out problems or add needed capabilities