Session 12: Overset and Rotorcraft Simulations

Bob Biedron and Beth Lee-Rausch
Learning Goals

• What this will teach you
  – Static and dynamic simulations using overset meshes (general)
  – Overview of setup for overset, articulated-blade rotorcraft simulations
    • Rigid Blades
    • Elastic Blades / Loose Coupling to Rotorcraft Comprehensive Codes
  – Overview of actuator-disc models for rotorcraft (not overset)
• What you will not learn
  – Rotorcraft Comprehensive Code set up and operation
  – SUGGAR++ operation (Ralph Noack will cover tomorrow)
• What should you already know
  – Basic time-accurate and dynamic-mesh solver operation and control
  – Rudimentary rotorcraft aeromechanics (collective, cyclic…)

FUN3D Training Workshop
April 27-29, 2010
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
  – AFLR3 and FieldView meshes not exercised much to date
  – Bodies in contact / emerging bodies - no near-term plans
Overset Mesh Simulations – General (1/3)

• Configuring FUN3D (only as a reminder, except to note compile scripts)
  – 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 (2/3)

• 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

<table>
<thead>
<tr>
<th></th>
<th>VGRID grid</th>
<th>FV grid</th>
<th>AFLR3 grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUN3D</td>
<td>grid.mapbc (standard VGRID file)</td>
<td>grid.mapbc (not same as VGRID)</td>
<td>grid.mapbc (not same as VGRID)</td>
</tr>
<tr>
<td>SUGGAR++</td>
<td>grid.mapbc (standard VGRID file)</td>
<td>grid.ext.suggar_mapbc (not same as VGRID)</td>
<td>grid.ext.suggar_mapbc (not same as VGRID)</td>
</tr>
</tbody>
</table>

  – “ext” is the FUN3D grid extension, e.g.: grid.fvgridFmt, 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 (2/3)

• 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 (3/3)

• Create an XML input file for SUGGAR++
  – Ralph Noack will provide *all* the details tomorrow; 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 `domain_connectivity` information 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 \texttt{--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].g2l) (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 \texttt{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 static-grid XML file:
    • Add `<dynamic/>` to `<body>` elements that are to move, e.g.
      ```xml
      <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 `mmbc` 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, will 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 – Rotorcraft Simulations
Trained Professionals. Closed Course. Do Not Attempt At Home.
Setting

• Background
  – FUN3D can model a rotor with varying levels of fidelity/complexity
    • As an actuator disk - when only the overall rotor influence is needed
    • As rotating, articulated-blade system (cyclic pitch, flap, lead-lag), with or without aeroelastic effects - if detailed airloads are needed
      – Trim and aeroelastic effects require coupling with a rotorcraft “comprehensive” code
    • As a steady-state problem for rigid, isolated, fixed-pitch blades in a rotating noninertial frame (not covered here)

• Compatibility
  – Coupling to the CAMRAD comprehensive code; other codes usable with appropriate middleware (not supplied)

• Status
  – Coded for multiple rotors, but largely untested
  – Only “loose” (periodic) coupling incorporated to date
  – Still an emerging capability; expect changes
Time-Averaged Actuator-Disk Simulations (1/2)

• Actuator disk method utilizes momentum/energy source terms to represent the influence of the disk (pressure jump)
  – Original implementation by Dave O’Brien (GIT Ph.D. Thesis)
  – HI-ARMS implementation (SMEMRD) by Dave O’Brien ARMDEC adds trim and ability to use C81 airfoil tables (Not covered in training)
• Simplifies grid generation – disk is embedded in computational grid (note some refinement in the vicinity of actuator surface needed for accuracy - but, Dave O’Brien recommends that delta-s of grid > delta-s disk)
• Any number of actuator disks can be modeled
• Different disk loading models available
  – **RotorType = 1** actuator disk
  • **LoadType = 1** constant (specified thrust coefficient $C_T$)
  • **LoadType = 2** linearly increasing to blade tip (specified $C_T$)
  • **LoadType = 3** blade element based (computed $C_T$)
  – **RotorType = 2** actuator blades (time-accurate) Not Functional
Time-Averaged Actuator-Disk Simulations (2/2)

• Actuator disk implementation runs orthogonal to the standard steady-state flow solver process (compressible and incompressible)
  – Standard input grid formats for the volume grids
  – Standard solver input deck (fun3d.nml)
  – Standard output is available (project.forces, project_hist.tec, project_tec_boundary.plt)
  – Want to see similar solution convergence as for a standard steady-state case

• Actuator disk model is activated in the command line by
  mpirun nodet_mpi --rotor
  – Rotor input deck file (rotor.input) is required in the local directory
  – rotor.input contains disk geometry and loading specifications
  – The disk geometry and loading are output in plot3d format in files
    source_grid_iteration#.p3d and
    source_data_iteration#.p3d
Incompressible Robin/Actuator Disk

Advance Ratio = 0.051 (V_{\text{inf}}/V_{\text{tip}})
Thrust coefficient $C_T = 0.0064$
Angle of attack = 0 deg
Shaft angle = 0 deg
**rotor.input File**

- Constant/linear loading needs only a subset of the data in the file

```plaintext
# Rotors  Uinf/Uref  Write Soln  Force Ref  Moment Ref  ! Below we set Uref = Uinf
1  1.000  1500  0.001117  0.001297  ! Adv Ratio = Uinf/Ut
!!! Main Rotor ===============  ! So here Utip/Uref = 1/AR

<table>
<thead>
<tr>
<th>Rotor Type</th>
<th>Load Type</th>
<th># Radial</th>
<th># Normal</th>
<th>Tip Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>50</td>
<td>180</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Utip/Uref</th>
<th>ThrustCoff</th>
<th>PowerCoff</th>
<th>psi0</th>
<th>PitchHing/R</th>
<th>DirRot</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.61</td>
<td>0.0064</td>
<td>-1.00</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

# Blades  TipRadius  RootRadius  BladeChord  FlapHinge/R  LagHinge/R
4  0.861  0.207  0.066  0.051  0.051

<table>
<thead>
<tr>
<th>LiftSlope</th>
<th>alpha, L=0</th>
<th>cd0</th>
<th>cd1</th>
<th>cd2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.00</td>
<td>0.002</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CL_max</th>
<th>CL_min</th>
<th>CD_max</th>
<th>CD_min</th>
<th>Swirl</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theta0</th>
<th>ThetaTwist</th>
<th>Thetas</th>
<th>Thetalc</th>
<th>Pitch-Flap</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.00</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># FlapHar</th>
<th>Beta0</th>
<th>Betals</th>
<th>Betalc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Beta2s</th>
<th>Beta2c</th>
<th>Beta3s</th>
<th>Beta3c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># LagHar</th>
<th>Delta0</th>
<th>Delta1s</th>
<th>Delta1c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delta2s</th>
<th>Delta2c</th>
<th>Delta3s</th>
<th>Delta3c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
```

- Note Vref=Vtip is bad choice for incompressible flow - suggest using rotor induced velocity
Incompressible Robin/Actuator Disk
Things To Look For In Screen Output

• If Force_ref = 1/(Vtip/Vref)^2/(\pi R^2) and
  Moment_ref = 1/(Vtip/Vref)^2/(\pi R^3)

Rotor force summary in standard output:

  Rotor Force Summary:
  Rotor   Grid Forces: Fx= 0.0000E+00 Fy= 0.0000E+00 Fz= 6.4008E-03
  Rotor   Grid Moments: Mx= -1.5898E-17 My= 8.6398E-18 Mz= 0.0000E+00
  Rotor   Shaft Forces: H = 0.0000E+00 Y = 0.0000E+00 T = 6.4008E-03
  Rotor   Shaft Moments: Mh= -1.5898E-17 My= 8.6398E-18 Q= 0.0000E+00

• Note that the force coefficients in project.forces and project_hist.tec are flow boundary forces only (no actuator disk forces) which have been normalized in the fixed wing fashion.
Articulated-Blade Simulations

• Relies on the use of overset grids; blades may be rigid or elastic

• Elastic-blade cases (or trimmed rigid-blade cases) must be coupled to a rotorcraft Computational Structural Dynamics (CSD, aka comprehensive) code such as CAMRAD, DYMORE, RCAS…
  – The CSD code provides trim solution in addition to blade deformations
  – The interface to the CSD code is through standard OVERFLOW rotor_N.onerev.txt and motion.txt type files
  – Interface codes (middleware) for CAMRAD are maintained and distributed by Doug Boyd, NASA Langley (d.d.boyd@nasa.gov)
  – FUN3D has several postprocessing utility codes tailored to CAMRAD

• A coupled elastic-blade simulation is about as complicated as it gets with the FUN3D flow solver
  – There are many small details that must be done correctly; we don’t have time to cover them all here
  – Novice users of FUN3D will want to start with simpler problems!
CFD/CSD – Loose (Periodic) Coupling

**Coupling Process**

```plaintext
Rotorcraft CSD Code
Iter i = 0: F/M_i = F/M_i(0)
Iter i > 0: F/M_i = F/M_i(1) + ΔF/M_i(1)
ΔF/M_i(1) = ΔF/M_i(2) + (F/M_i(CFD) - F/M_i(1))
Obtain Trimmed Control Settings For Rotor F/M
```

c/4 Motion - disp & rot

Move Grid + CFD Soln

F/M: C_{Fv}, C_M, C_{Chord}

F/M along c/4

no

F/M and Trim Converged?

yes

Done

**motion.txt and rotor_onerev.txt files common to FUN3D and OVERFLOW**

**CFD/CSD loose coupling implemented via shell script with error checking**
dci_gen Preprocessor (1/8)

• A rudimentary code to simplify rotorcraft setup (/utils/Rotocraft/dci_gen)
  – Uses libSUGGAR++ routines
  – Takes a single blade grid and a single fuselage / background grid (extending to far field) and assembles them into an N-bladed rotorcraft
  – Creates the SUGGAR++ XML file (Input.xml_0) needed by FUN3D
  – Generates, using libSUGGAR++ calls, the initial (t = 0) dci file and composite grid needed by FUN3D
  – Generates the composite-grid “mapbc” files needed by FUN3D
  – Component grids must be oriented as shown on following slide
    • Blade must have any “as-built” twist incorporated
    • If grids do not initially meet the orientation criteria, can use SUGGAR++ to rotate them before using dci_gen

• Don’t have to use dci_gen; could create the XML file by hand and run SUGGAR++; a more complex setup could start with dci_gen, hand edit the resulting XML file, then follow with SUGGAR++
dci_gen Preprocessor (2/8)

HART II Component Grids

Correct Axes Orientation For Fuselage Component Grid
(location of origin not critical)

Correct Axes Orientation For Blade Component Grid
(x-axis should correspond to blade feathering/pitch axis)
dci_gen Preprocessor (3/8)

HART II Composite Grid

Composite 4-Bladed Rotorcraft (Surface) Grid
dci_gen Preprocessor (4/8)

- Requires the `rotor.input` file (also required by flow solver - more later)
- Creates the initial composite mesh with the blades at zero collective, zero cyclic, zero flap; however, the rotor is tilted according to $\phi_2$ (shaft tilt); resulting mesh and dci file can then be used for multiple flight conditions
- FUN3D will “pop” the blades into the correct $t = 0$ position at the start of simulation, based either on the collective, cyclic, etc. data in `rotor.input` (rigid) or on the data in the “motion.txt” file (elastic)
- For rigid, untrimmed blades, `rotor.input` gives a complete definition of the blade motion - `dci_gen` can create dci data for all blade positions a priori; this can be done in “embarassingly parallel” manner, faster than can be done from within the flow solver
- `dci_gen` will prompt the user for input; example next slide
- `dci_gen` will read (if present) a file called `manual_hole_commands` that can be used to add problem-specific additional XML commands to aid the computation of overset connectivity data
dcigen Preprocessor (5/8)

- Usage: ./dci_gen first echos rotor.input, then prompts for input:
  - Enter a project name: (e.g. robin) 
    uh60_alw_isolated_c2_ft
  - Enter the name of the fuselage grid: (e.g. robin_fuse) 
    empty_box_coarse2_uh60_ft
  - Enter the type of fuselage/background grid: vgrid, aflr3, or fvuns 
    aflr3
  - Is this grid formatted (enter f) or unformatted (enter u) 
    f ! This question NOT asked if type = vgrid
  - Is this grid single precision (enter s) or double precision (enter d) 
    d ! This question NOT asked if type = vgrid
  - For multiple rotors, the first rotor should be the main rotor 
  - Additional rotors spin with gear ratios relative to rotor 1 
  - Enter the name of the blade grid for rotor 1: (e.g. robin_blade) 
    uh60_alw_blade_tab_c2_t2_ft
  - Enter the type of blade grid: vgrid, aflr3, or fvuns 
    aflr3
  - Is this grid formatted (enter f) or unformatted (enter u) 
    f ! This question NOT asked if type = vgrid
  - Is this grid single precision (enter s) or double precision (enter d) 
    d ! This question NOT asked if type = vgrid
  - Enter initial psi, final psi, and psi increment values for the first rotor 
    0.0 0.0 1.0 ! Just initial azimuth - elastic blades
** dci_gen Preprocessor (6/8) **

- After data summary and echo of XML commands, should see:
  
  *** Computing DCI data  
  *** Finished DCI file: uh60_alw_isolated_c2_ft.dci  

  \[ \psi(\text{rotor 1}) = 0.0000 \]

  Orphan Info:  
  Found 0 orphans because of hole cut failures  
  Sort added 0 orphans because of poor quality donors

  SUGGAR++ Resource Requirements:  
  Wall Clock Time 488.004623 seconds  
  Memory Usage 3180 Mbytes

  ** Finished Creating DCI Files **
dci_gen Preprocessor (7/8) skip - FYI

- In some cases we may supply a `manual_hole_commands` file, with, for example, the entries shown below; without this file, the red elements below would not have appeared in resulting `Input.xml_0` file shown on the next slide, and the overlap connectivity might suffer:

```xml
<global>
  <thin_cut set_to="out"/>
  <donor_quality value="0.9" />
  <minimize_overlap keep_inner_fringe="yes"/>
</global>

<volume_grid name="hartii_rotor_test">
  <skip_overlap_opt set_dsf_value="0.0"/>
</volume_grid>
```

- Alternatively to `manual_hole_commands`, run `dci_gen`, modify resulting `Input.xml_0`, and run SUGGAR++ “by hand”
dcigen Preprocessor (8/8) skip - FYI

• The resulting Input.xml file is (greatly edited to fit):

```xml
<global>
  <thin_cut set_to="out"/>
  <donor_quality value="0.9" />
  <minimize_overlap keep_inner_fringe="yes"/>
  <output>
    <composite_grid style="unsorted_vgrid_set" filename="hartii_test"/>
    <domain_connectivity style="unformatted_gen_drt_pairs" ... />
  </output>
  <body name="complete">
    <body name="rotor1_bladel">
      <dynamic/>
      <transform>
        ...
      </transform>
      <volume_grid name="hartii_rotor_test" ... >
        <skip_overlap_opt set_dsf_value="0.0"/>
      </volume_grid>
    </body>
    <body name="fuselage">
      <volume_grid name="hartii_box_test" ...">
      </volume_grid>
    </body>
  </body>
</global>
```
moving_body.input File

• For rotorcraft, need only define blades as moving bodies and set the initial XML file; actual motion info comes from rotor.input and motion.txt

```xml
&body_definitions
  n_moving_bodies = 4, ! 4 blades
  body_name(1) = 'rotor1_blade1', ! name is set by *dci_gen* - must use unaltered
  n_defining_bndry(1) = 1, ! number of boundaries that define this blade
  defining_bndry(1,1) = 2, ! index 1: boundry number index 2: body number
  mesh_movement(1) = 'deform', ! blades are elastic
  body_name(2) = 'rotor1_blade2',
  n_defining_bndry(2) = 1,
  defining_bndry(1,2) = 4,
  mesh_movement(2) = 'deform',
  body_name(3) = 'rotor1_blade3',
  n_defining_bndry(3) = 1,
  defining_bndry(1,3) = 6,
  mesh_movement(3) = 'deform',
  body_name(4) = 'rotor1_blade4',
  n_defining_bndry(4) = 1,
  defining_bndry(1,4) = 8,
  mesh_movement(4) = 'deform',
/
&composite_overset_mesh
  input_xml_file = 'Input.xml_0' ! use file generated by dci_gen
/
```

! NOTE: motion_driver() should NOT be specified
# Rotors Uinf/Uref Write Soln Force Ref Moment Ref ! Below we set Uref = Utip
1 0.245 1500 1.0 1.0 ! Adv Ratio = Uinf/Utip

--- Main Rotor ----------------------------------------------- ! So here Uinf/Uref = AR

<table>
<thead>
<tr>
<th>Rotor Type</th>
<th>Load Type</th>
<th># Radial</th>
<th># Normal</th>
<th>Tip Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>X0_rotor</td>
<td>Y0_rotor</td>
<td>Z0_rotor</td>
<td>phi1</td>
<td>phi2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Utip/Uref</th>
<th>ThrustCoff</th>
<th>PowerCoff</th>
<th>psi0</th>
<th>PitchHinge</th>
<th>DirRot</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>0.0064</td>
<td>-1.00</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0646</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># Blades</th>
<th>TipRadius</th>
<th>RootRadius</th>
<th>BladeChord</th>
<th>FlapHinge</th>
<th>LagHinge</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>26.8330</td>
<td>2.6666</td>
<td>1.741</td>
<td>0.0466</td>
<td>0.0466</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LiftSlope</th>
<th>alpha, L=0</th>
<th>cd0</th>
<th>cd1</th>
<th>cd2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.28</td>
<td>0.00</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CL_max</th>
<th>CL_min</th>
<th>CD_max</th>
<th>CD_min</th>
<th>Swirl</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.50</td>
<td>-1.50</td>
<td>1.50</td>
<td>-1.50</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theta0</th>
<th>ThetaTwist</th>
<th>Theta1s</th>
<th>Theta1c</th>
<th>Pitch-Flap</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># FlapHar</th>
<th>Beta0</th>
<th>Beta1s</th>
<th>Beta1c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Beta2s</th>
<th>Beta2c</th>
<th>Beta3s</th>
<th>Beta3c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># LagHar</th>
<th>Delta0</th>
<th>Delta1s</th>
<th>Delta1c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Delta2s</th>
<th>Delta2c</th>
<th>Delta3s</th>
<th>Delta3c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

### Key:
- Required for rigid and elastic
- Required for untrimmed rigid
- Unused (must have a value)
Nondimensional Input (1/2)

- Typically define the flow reference state for rotors based on the tip speed; thus in `rotor.input`, set $\frac{U_{\text{tip}}}{U_{\text{ref}}} = 1.0$ (data line 4)
- This way, $\frac{U_{\text{inf}}}{U_{\text{ref}}}$ (data line 1) is equivalent to $\frac{U_{\text{inf}}}{U_{\text{tip}}}$, which is the Advance Ratio, and is usually specified or easily obtained
- Since the reference state corresponds to the tip, the `mach_number` in the `fun3d.nml` file should be the tip Mach number, and the `reynolds_number` should be the tip Reynolds number
- Nondimensional rotation rate: not input directly, but it is output to the screen; you might want to explicitly calculate it up front as a later check:

$$\Omega^* = \frac{U^*_{\text{tip}}}{R^*} \text{ (rad/s, } R^* \text{ the rotor radius)}$$

and recall $\Omega = \frac{\Omega^*(L^*_\text{ref} / L^*_\text{ref})}{a^*_\text{ref}}$ (compressible) from yesterday so with $a^*_\text{ref} = \frac{U^*_\text{ref}}{M^*_\text{ref}}$ and taking $L^*_\text{ref} = R^*$

$$\Omega = \frac{M^*_\text{ref} (U^*_\text{tip} / U^*_\text{ref})}{R} \quad \text{(compressible)}$$

$$\Omega = \frac{U^*_\text{tip}}{U^*_\text{ref}} / R \quad \text{(incompressible)}$$
Nondimensional Input (2/2)

- Nondimensional time step:
  
  \[ T^* = \frac{2\pi}{\Omega^*} = \frac{2\pi R^*}{U_{tip}^*} \ (s) \]
  
  and recall \( t = t^* a_{ref}^* \left( \frac{L_{ref}}{L_{ref}^*} \right) \) (compressible) from yesterday

  so with \( L_{ref}^* = R^* \) we have

  \[ T = a_{ref}^* \left( \frac{R}{R^*} \right) 2\pi R^*/U_{tip}^* = 2\pi R/\left( M_{ref} U_{tip}^* / U_{ref}^* \right) \] (nondim time / rev)

  For N steps per rotor revolution:

  \[ \Delta t = 2\pi R/\left( N M_{ref} U_{tip}^* / U_{ref}^* \right) \] (compressible)

  \[ \Delta t = 2\pi R/\left( N U_{tip}^* / U_{ref}^* \right) \] (incompressible)

- Note: the azimuthal change per time step is output to the screen in the Rotor info section. Make sure this is consistent, to a high degree of precision (say at least 4 digits), with your choice of N steps per rev – you want the blade to end up very close to 360 deg. after multiple revs!

- Formulas above are general, but recall we usually have ref = tip, at least for compressible flow
Blade Surface “Slicing”

• Boundary surface (rotor blade) slicing is required for coupled CFD/CSD simulations; also useful for rigid-blade cases - this is what generates the data in rotor_1.onerev.txt

```plaintext
$slice_data
replicate_all_bodies = .true.    ! do the following the same on all blades
output_sectional_forces = .false. ! just lots of data we usually don’t need
tecplot_slice_output = .false.    ! ditto
slice_x(1) = .true.,          ! x=const slice - in original blade coords
nslices = -178,           ! no. slices; “-” means give start and delta
slice_location(1) = 2.8175,    ! x-location to slice (starting slice)
slice_increment = .13416666666 ! delta slice location each successive slice
n_bndrys_to_slice(1) = 1,       ! 1 bndry to search
bndrys_to_slice(1,1) = 2,       ! indices:(slice,bdry) lumping made life easy
slice_frame(1) = 'rotor1_blade1', ! ref. frame in which to slice - use body name
te_def(1) = 20,      ! look for 2 corners in 20 aft-most segments
le_def(1) = 30,      ! search 30 fwd-most pts for one most distant from TE
chord_dir(1) = -1,    ! Recall goofy original blade coord system
```

• Note: “slicing” useful for applications other than rotorcraft; see website

http://fun3d.larc.nasa.gov

FUN3D Training Workshop
April 27-29, 2010
CAMRAD Considerations

- User must set up basic CAMRAD II scripts; the `RUN_LOOSE_COUPLING` script provided with FUN3D requires 3 distinct, but related CAMRAD scripts
  - `basename_ref.scr`
    - Used to generate the reference motion data used by CAMRAD
    - Set this file to use rigid blades; zero collective/cyclic; no trim
  - `basename_0.scr`
    - Used for coupling/trim cycle “0”
    - Set up for elastic blades with trim; use CAMRAD aerodynamics exclusively (no delta airloads input); simplest aero model will suffice
  - `basename_n.scr`
    - Used for all subsequent coupling/trim cycles
    - Set up for elastic blades with trim; use same simple CAMRAD aerodynamics but now with delta airloads input
- Sample scripts (basename: `hart`) are provided in `utils/Rotorcraft`; 1st 2-4 executable lines of each script show tailoring required to use with `RUN_LOOSE_COUPLING` script
Untrimmed Rigid-Blade Simulations

• Overview of the basic steps

1. Prepare rotor blade and fuselage grids, with proper axis orientation
2. Set up the \texttt{rotor.input} file based on desired flight conditions
3. Run the \texttt{dci\_gen} utility to create a composite mesh and initial dci data
4. Set up \texttt{fun3d.nml} and \texttt{moving\_body.input} files
5. Optionally set up the \&\texttt{slice\_data} namelist in the \texttt{fun3d.nml} file
6. Run the solver with the following command line options (in addition to any other appropriate ones, like \texttt{--temporal\_err\_control})

\texttt{--moving\_grid --overset --overset\_rotor --dci\_on\_the\_fly --dci\_period 360 --reuse\_existing\_dci}

If optional step 5 is used, add the following (N as desired, typically 1)

\texttt{--slice\_freq N --output\_comprehensive\_loads}

7. Number of time steps required is case dependent – usually at least 3 revs
Trimmed, Elastic-Blade Simulations

• Overview of the basic steps; steps 1-4 are the same as for the unTrimmed rigid-blade case; use of CAMRAD is assumed

5. Set up the \texttt{&slice\_data} namelist; \textit{not optional}

6. Set up the 3 CAMRAD run script templates

7. Set up the \texttt{RUN\_LOOSE\_COUPLING} run script (a c-shell script geared to PBS environments); user-set data is near the top – sections 1 and 2

8. Set up the \texttt{fun3d.nml\_initial} and \texttt{fun3d.nml\_restart} files used by the run script; typically set the time steps in the initial file to cover 2 revs, and $2/\text{N_{blade}}$ revs in restart version

9. If using the run script make sure all items it needs are in place; script checks for missing items, but it gets old having to keep restarting because you forgot something!

10. Number of coupling cycles required for trim can vary, but 8-10 is typical for low-moderate thrust levels; high thrust cases near thrust boundary may require 10-15; user judges acceptable convergence
RUN_LOOSE_COUPLING Directory Tree

Run Directory
Script executed here
FUN3D runs here

-----

CAMRAD

Reference

Trim_0

Trim_1

-----

Principal solver files archived here at end of each trim cycle

-----

Script creates all subdirectories

-----

CAMRAD and interface codes run here and output stored here for each trim cycle

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

• Rotor info section lists some basic data:

  Rotor info, rotor 1
  Number of blades : 4
  Nondimensional rotation rate : 0.02493199
  Azimuth change (deg) per time step : 1.00000000 ! make sure its accurate
  Tip Mach number (hover) : 0.66900000
  Advance ratio : 0.24500000
  Tip radius : 26.83300000
  Force/Moment reference area : 2261.97777779
  Force/Moment reference length : 26.83300000
  Moment reference x-center : 0.00000000
  Moment reference y-center : 0.00000000
  Moment reference z-center : 0.00000000

  Note: force/moment reference data above supercedes any other input values

• If running elastic blades:

  Reading CAMRAD motion file for rotor 1: camrad_motion_dataRotor_1.dat
  nspan = 100
  npsi = 24

  Enforcing periodicity in CAMRAD motion data

  • Note: camrad_motion_data_rotor_1.dat is what FUN3D calls motion.txt
Things To Look For In Screen Output (2/2)

- Running average of integrated blade loads at the end of each time step:

  Rotor Forces and Moments, Rotor 1
  Averages over 180 steps

  Inertial Axes
  \[
  \begin{align*}
  C_x & : -0.000124 \\
  C_y & : -0.000328 \\
  C_z & : 0.009951 \\
  C_{mx} & : 0.000013 \\
  C_{my} & : -0.000049 \\
  C_{mz} & : -0.000663
  \end{align*}
  \]

  Nonrotating Shaft Axes
  \[
  \begin{align*}
  C_x & : -0.000124 \\
  C_y & : -0.000328 \\
  C_z & : 0.009951 \\
  C_{mx} & : 0.000013 \\
  C_{my} & : -0.000049 \\
  C_{mz} & : -0.000663
  \end{align*}
  \]

  Wind Axes
  \[
  \begin{align*}
  Cl & : 0.009949 \\
  Cd & : -0.000212
  \end{align*}
  \]

  Performance Parameters
  \[
  \begin{align*}
  \text{Thrust, } C_t & : 0.009951 \\
  \text{Torque, } C_q & : 0.000663
  \end{align*}
  \]
Postprocessing (1/2)

• For elastic blades, or rigid blade cases with optional “slicing” and
  --output_comprehensive_loads, the following files are output;
  – rotor_1.onerev.txt (OVERFLOW standard, airloads F/M data)
  – motion_rotor_1.onerev.txt (similar to above, but motion data)
• Utility code process_rotor_data.f90, with input file
  process_rotor_data.input (code and sample input in utils/Rotorcraft)
  – Extracts aero and displacement data into a number of Tecplot files:
    • airloads_polarplotRotor_1.dat
    • sectional_forces_vs_azimuth_rotor_1.dat
    • sectional_forces_vs_radius_rotor_1.dat
    • computed_qc_position_vs_azimuth_rotor_1.dat (section c/4 positions
      and section pitch)
    • computed_qc_position_vs_radius_rotor_1.dat
    • mean_sectional_forces_vs_radius_rotor_1.dat
  – “forces” and “polarplot” have M²C_N, M²C_M, and M²C_x data
  – The first three files also have equivalent “mean removed” versions
Postprocessing (2/2)

Sample Plots Possible Via process_rotor_airloads.f90 Output
List of Key Input/Output Files

• Beyond basics like `fun3d.nml`, `[project]_hist.tec`, etc.:

• Input
  – `moving_body.input`
  – `Input.xml_0` (dynamic overset; no standard name)
  – `[project].dci` (all overset)
  – `rotor.input` (all R/C)
  – `camrad_motionRotor_N.dat` (aka `motion.txt`, coupled R/C)
  – `case_ref.scr`, `case_0.scr`, `case_N.scr` (coupled R/C)

• Output
  – `rotor_1.onerev.txt` (articulated R/C)
  – `motion_rotor_1.onerev.txt` (articulated R/C)
FAQ’s

• How long does it take (esp. as regards to coupled rotorcraft simulations)?
  – If you have to ask you can’t afford it!
  – Currently (April 2010), a 7 million node UH-60 simulation, which required 10 coupling cycles to converge to trim targets, takes approximately 72 hrs on 96(+1) processors of a 3.0 GHz P4 Dual Core 4GB GigE cluster - same cluster used in interactive sessions
  – Expect future speedup from implementation of parallel SUGGAR++ processing
What We Learned

• How to set up and run static and dynamic overset meshes in FUN3D
  – To fully utilize, requires knowledge of SUGGAR++, for which training will be provided tomorrow

• Rotorcraft simulations
  – Actuator disk models for basic influence of rotor
  – Moving, articulated blades for detailed airloads analysis - much more expensive and involved

• Assemble the composite grid with \texttt{dci\_gen}; takes most of the work out of setting up the SUGGAR++ XML file, using an input file you later need for FUN3D

• Rigid blades (untrimmed) can be run without coupling to a comprehensive code

• Coupled FUN3D / CAMRAD solutions a huge step up in complexity!