Session 10:
SUGGAR++ Basics

Bob Biedron

Learning Goals

- What this will teach you
  - Very rudimentary SUGGAR++ operation
- What you will not learn
  - All the useful stuff that Ralph Noack would teach you
  - GVIZ (Ralph's own viewer for overset grid assembly - useful for debugging/assessing hole cutting)
- What should you already know
  - Basic concept of overset meshes

Setting

- Background
  - Use of overset grids in FUN3D requires either SUGGAR++ or SUGGAR (predecessor)
    - SUGGAR++ and SUGGAR are very similar in functionality and usage; will generally use "SUGGAR" and "SUGGAR++" interchangeably here; in one or two spots the differences are noted
    - Disclaimer: I am not a SUGGAR expert - just a user for limited applications; this presentation may contain factual errors or other misinformation
- Compatibility
  - FUN3D requires both DiRTlib and SUGGAR codes from PSU
  - Grid formats: VGRID, AFLR3, FieldView (FV)
- Status
  - Overset simulations done with FUN3D and SUGGAR++ on a frequent basis, primarily for rotorcraft applications.

SUGGAR++ Documentation

- User’s Guide: doc/UsersGuide/UsersGuide.pdf
  - Documents list of input elements (the rules, not much of the “why”)
  - Documents command-line options for SUGGAR++
- Programmer’s Guide: doc/ProgrammersGuide/ProgrammersGuide.pdf
  - Compilation
  - How to integrate libSUGGAR++ into a flow solver
- Training slides presented by Ralph Noack and Dave Boger at the April 2010 FUN3D Training Session will eventually make it on to the FUN3D website
  - Much of the material here is a distillation of the April slides - but they had a full day to cover this

Nomenclature (1/4)

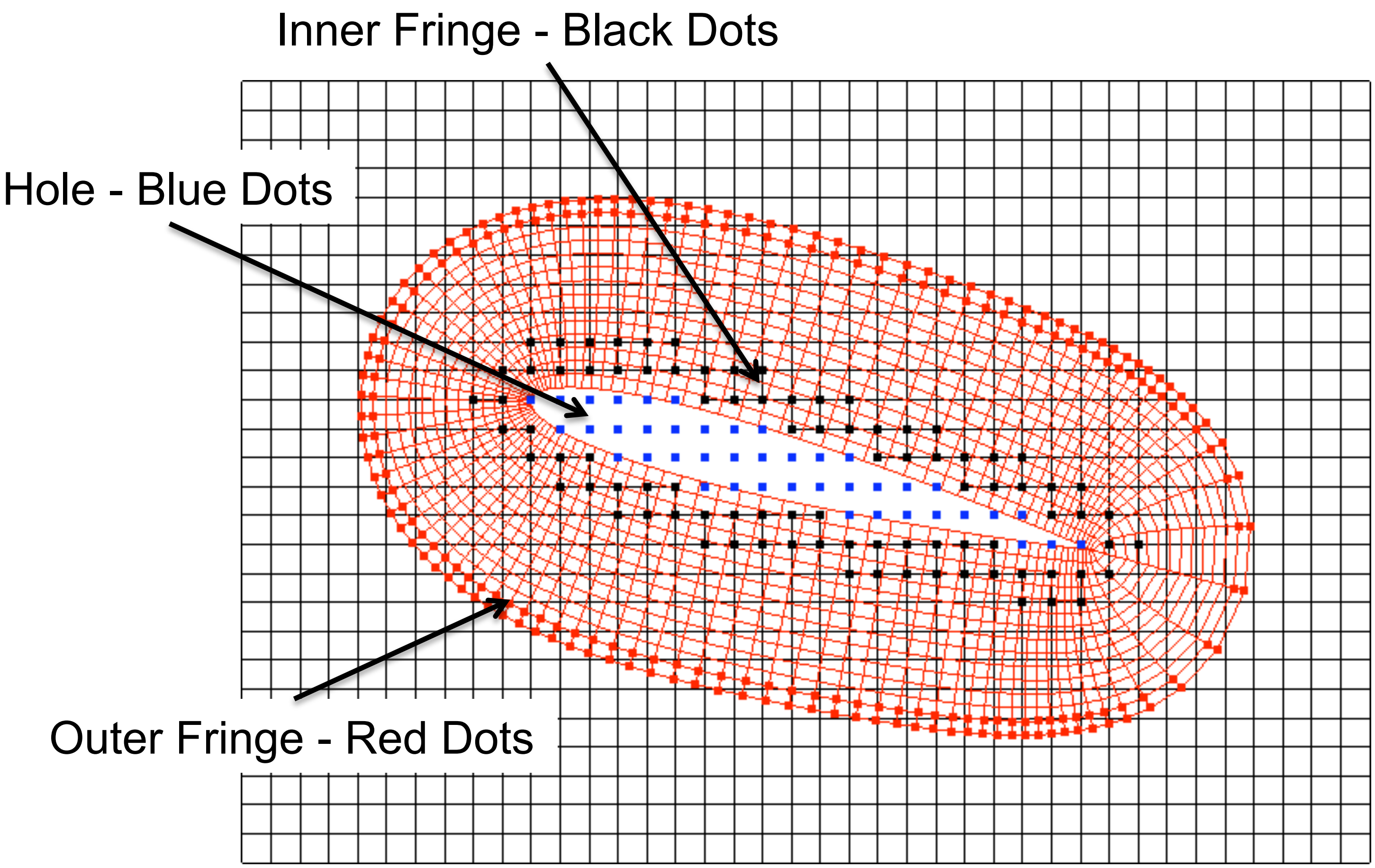
- SUGGAR: Structured, Unstructured, Generalized overset Grid AssembleR SUGGAR++ is the next-generation version
  - PEGASUS-like capability for general grids
  - Stand-alone versions for static grids; library versions for dynamic grids
- DiRTlib: Donor interpolation/Receptor Transaction library - used by flow solver to handle data provided by SUGGAR++; no user input (just compile and link to flow solver)
- Component Grid
  - "Independently" generated grid for one piece of the configuration
  - Up to you to create these
- Composite Grid
  - An assembly of component grids
  - Created by SUGGAR++ based on your input

Nomenclature (2/4)

- Overset grid point classification
  - In or Active: flow solver updates these points by solving the governing equations at these locations
  - Out or Hole: flow solver need not update these points as they have been removed from the domain
    - In practice, especially for moving grids, the flow solver fills in data at these points by averaging neighboring points - done so that as points move from “out” to “in”, they have “reasonable” data
  - Fringe: these points are updated by interpolation from “in” points; fringe points border a hole (inner fringe) or lie along an outer boundary (outer fringe)
  - Donor: the “in” points that supply data to fringe points
  - Orphan: fringe points for which too few or no donor points can be found; undesirable; solver fills in data at these points by averaging solution at neighboring points

Nomenclature (3/4)

- Flow solver point classification - example



Nomenclature (4/4)

- DCI file
  - Domain Connectivity Information file
  - Created by SUGGAR; contains information about point classifications (hole, fringe, etc) for points in composite mesh, plus interpolation stencil data
  - Calls to DiRTlib within FUN3D read the DCI file and utilize the data within to update the solution at fringe points via interpolation from donor points
  - If grid is static, only need one DCI file
  - If grid is dynamic, must either have pre-computed DCI files available for the grid positions at each time step, or utilize libSUGGAR calls within FUN3D to compute DCI data "on the fly"

XML Basics (1/2)

- SUGGAR/SUGGAR++ input based on XML
  - eXtensible Markup Language (HTML-like, but not web-centric)
  - XML element is enclosed in a tag “< >” , with corresponding end tag **<body>** … **</body>** (start and end can also span multiple lines)
  - Elements can have attributes/data: **<body name=“wing”>**
  - Elements can have an implicit end tag; elements can be empty - no attributes: **<dynamic/>**
  - XML elements can be embedded in other XML elements to create parent-child relationships (wing and store are children of aircraft)

```
<body name="aircraft">
  <body name="wing">
  </body>
  <body name="store">
  </body>
</body>
```

XML Basics (2/2)

- Element attributes are name/value pairs associated with an element
  - Always in the start tag, value must be in quotes (single or double)

```
<body name='blade_1'> … </body>
<translate axis="x" value="1.0e0"/>
```

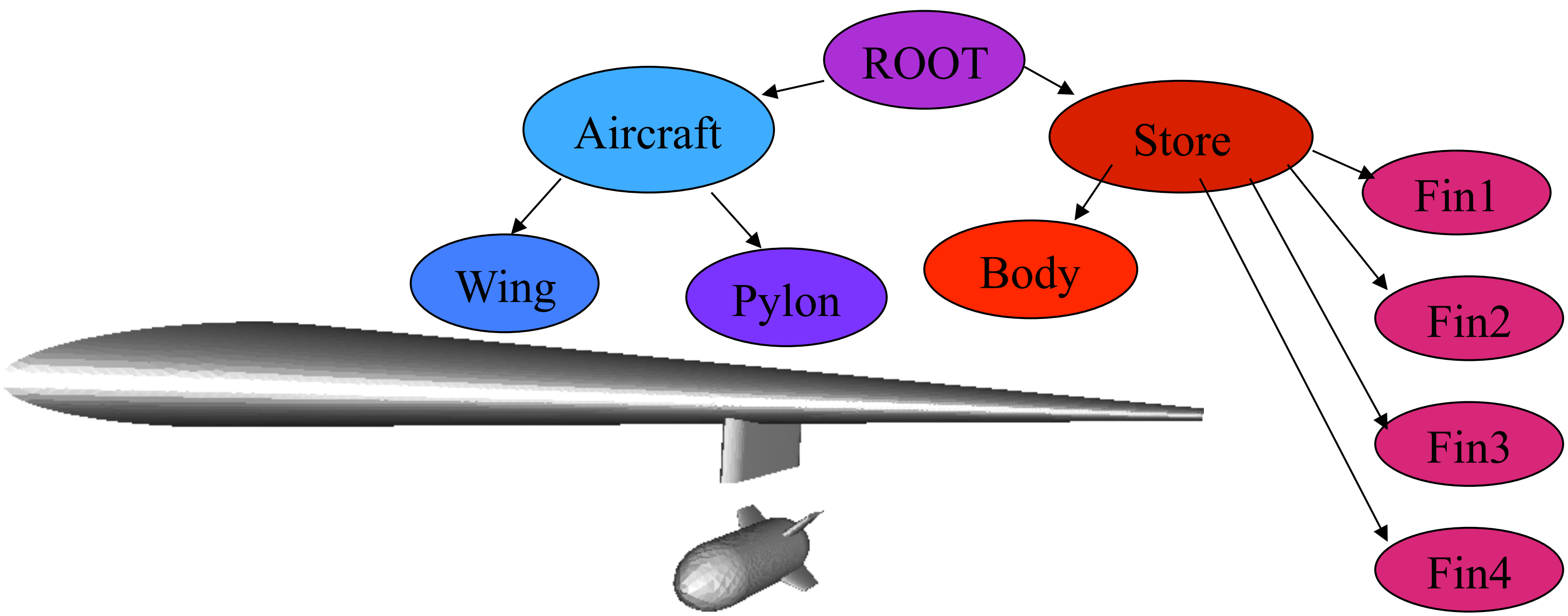
- Comments start with `<!--` and end with `-->` and cannot be within a tag

`<!-- <body name="aircraft"/> -->`	Correct
`<body <!-- name="aircraft" --> />`	Incorrect

- XML syntax must be precise: xmllint is on most(?) systems and can be used to check XML syntax before using SUGGAR
  - Usage: xmllint myfile.xml
  - If syntax is OK, will simply echo XML file to screen; otherwise it reports the error
- Indentation helps keep XML input readable; xmllint can help here too
  - Usage: xmllint -format my_messy_file.xml > my_neat_file.xml

Hole-Cutting: Hierarchy

- Parent-Child hierarchy established in XML file minimizes additional input to control hole cutting
- Basic rule: siblings cut each other
  - Geometry in one body (including all children) cut all grids in a sibling body (including all children); Aircraft cuts hole in Store and vice versa



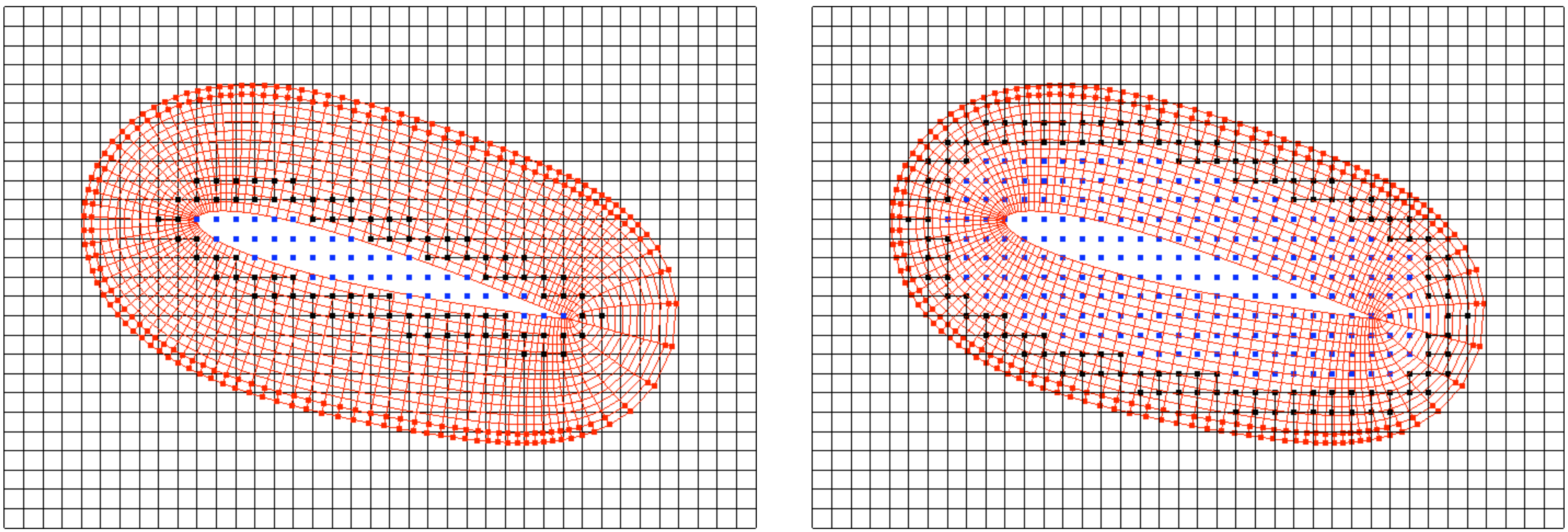
Hole-Cutting: SUGGAR vs SUGGAR++

- Older SUGGAR code relies (primarily) on Octree hole cutting - uses Cartesian representation of geometry; hole cutting based on a query approach: Is this point inside (the Cartesian representation of) the body?

- In my experience, the Octree hole cutting approach often needs a lot of tweaking beyond the default behavior
- Newer SUGGAR++ code relies (primarily) on a direct hole cutting approach: Find intersections of geometry and grid; requires watertight geometry
- In my experience very little tweaking has been required with SUGGAR++
- SUGGAR++ supports the older Octree approach too; other hole-cutting options are available in both codes but are beyond the scope here
- There are pros and cons to any approach…

Hole Cutting: Overlap Minimization

- Solution quality usually improved by reducing amount of overlap
- Goal is to have donors and receptors of similar size
- Enabled by element **<minimize_overlap>**
- For moving grids: **<minimize_overlap keep_inner_fringe="yes"/>**
  - Instead of blanking out points removed in overlap minimization, keeps them as fringes so they are *interpolated* rather than *averaged* - presumably better for when these points later emerge from the hole



Building Up A SUGGAR Input File (1/7)

- **`<global>`** element serves as the root (parent) element for *every* SUGGAR input file: first line in file is **`<global>`** and last line is **`</global>`**
- Child elements of **`<global>`** specify various global parameters, and the body hierarchy
- So on a high level an input file for an aircraft composed of a wing and a store would look something like:

```
<global>
 <!-- global parameters here -->
 <body name="aircraft">
   <body name="wing">
   </body>
   <body name="store">
   </body>
 </body>
</global>
```

Building Up A SUGGAR Input File (2/7)

- Common child elements of **`<global>`** (see documentation for more info)
  - **`<donor_quality value="0.9"/>`** (lower stencil quality standard to reduce number of orphans)
  - **`<minimize_overlap keep_inner_fringe="yes"/>`**
  - **`<output>`** (governs output of composite mesh and DCI file)
  - Principle children of **`<output>`**
    - **`<composite_grid filename="file" style="style"/>`**
    - **`<domain_connectivity filename="file" style="style"/>`**
    - Note: **`<composite_grid>`** and **`<donor_receptor_file>`** are for SUGGAR++; SUGGAR uses different element names, but accomplish the same thing
    - **`<composite_grid/>`** style attributes compatible with FUN3D: **`"vgrid_set", "unsorted_vgrid_set", "fvuns", "aflr3", "ugrid"`**

Building Up A SUGGAR Input File (3/7)

- `<body>` element can be child of `<global>` or another `<body>`
  - Required attribute `is name="body_name"`
- Common child elements of `<body>` (see documentation for more info)
  - `<volume_grid name="wing" filename="Grids/wing" style="vgrid_set"/>` (associates a volume grid with a body)
  - `<dynamic>` (declares a body as moving; also determines how the element `<transform>` is handled)
  - `<transform>` (used to manipulate body - scale, rotate, translate, etc.)
    - If `<transform>` is child of `<body>`, transform is "static" - input grid coordinates are actually altered by the transform specified
    - If `<transform>` is child of `<dynamic>`, transform is "dynamic" - input grid coordinates are *not* altered by the transform; the transform is only used internally
    - I find this more than a little confusing…please see the documentation for yourself

Building Up A SUGGAR Input File (4/7)

- I deal with the **<transform>** duality by adopting the following fixed strategy for moving-body cases:
  - Always make it a child of **<body>** and not a child of **<dynamic>**
  - Add a "self-terminating" **<dynamic/>** child to any body I want to have in motion:

```
<body name="store">
  <dynamic/>
  <transform>
    <scale value= '1.666666666667'/>
  </transform>
</body>
```

  - Because the **<dynamic/>** element self terminates, **<transform>** is not a child of it
  - I don't claim this is the "right" way…but it works for my applications
  - Not an issue for non-moving bodies in the composite grid

Building Up A SUGGAR Input File (5/7)

- Children of **<transform>**:
  - **<translate>**
  - **<rotate>** (used to rotate about x, y, or z)
  - **<rotate_about_v>** (used to rotate about arbitrary vector axis)
  - **<scale>**

```
<body name="store">
  <dynamic/>
  <transform>
    <scale value= '1.666666666667'/>
  </transform>
</body>
```

  - The order of transforms is important; transforms applied in order specified in the input file
- Refer to documentation for complete rules about which elements are allowed as children, which are allowed as parent, allowable attributes, etc.

Building Up A SUGGAR Input File (6/7)

- More complex example of `<transform>` from rotorcraft application

```
<body name="rotor1_blade2">
  <dynamic/>
    <transform>
      <translate axis="x" value="  7.6520E-01"/>
      <translate axis="y" value="  0.0000E+00"/>
      <translate axis="z" value="  7.9600E-01"/>
      <rotate_about_v axis_vector="0.0E+00,  1.0E+00, 0.0E+00" value="0.0E+00”
   originx="7.652E-01" originy="0.0E+00" originz ="7.96E-01"/>
      <rotate_about_v axis_vector="1.0E+00,  0.0E+00, 0.0E+00" value="0.0E+00"
   originx="7.652E-01" originy="0.0E+00" originz ="7.96E-01"/>
      <rotate_about_v axis_vector="0.0E+00,  0.0E+00, 1.0E+00" value="0.0E+00"
   originx="7.652E-01" originy="0.0E+00" originz ="7.96E-01"/>
      <rotate_about_v axis_vector="0.0E+00, -1.0E+00, 0.0E+00" value="0.0E+00"
   originx="7.652E-01" originy="0.0E+00" originz ="7.96E-01"/>
      <rotate_about_v axis_vector="0.0E+00,  0.0E+00, 1.0E+00" value="9.0E+01"
   originx="7.652E-01" originy="0.0E+00" originz ="7.96E-01"/>
    </transform>
  <volume_grid name="rotor_w_cutout_1_correct_pitch" style="vgrid_set"
   filename="rotor_w_cutout_1_correct_pitch" format="unformatted"
   precision="double">
  </volume_grid>
</body>
```

Building Up A SUGGAR Input File (7/7)

- Boundary conditions
  - SUGGAR needs to know some boundary condition information, e.g. which are the solid (body) boundaries, which outer boundaries need to be interpolated from other grids
  - SUGGAR input has provision for specifying the required SUGGAR BC's via XML elements
  - An alternative is to provide SUGGAR with a separate file with the BC data
  - I always use the separate file and so will not cover the xml input file approach - this is by far the most expedient way for VGRID meshes
- Pretty much wraps up this very brief overview of what goes into the input XML file for SUGGAR; documentation goes into much more and you should consult it in detail
- Next, look at how to set those BC's for SUGGAR via a file

Boundary Condition Files For SUGGAR

- 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
```

Running SUGGAR/SUGGAR++ (1/3)

- Ralph recommends creating a "Grids" subdirectory and an "Input" subdirectory for each case
  - I never make an Input directory but do use a separate directory to hold the component grids
  - By default SUGGAR will look to read Input/Input.xml, so if you don't have this you simply have to explicitly give the input file name
- You will want to redirect stdout and stderr (stdout has LOTS of output); below, file name **`Input.xml_0`** is explicitly given
  - c-shell

    ```
    (./suggar++ Input.xml_0 > suggar++.output) > & suggar++.error
    ```

  - bourne-shell

    ```
    ./suggar++ Input.xml_0 1> suggar++.output 2> suggar++.error
    ```

  - Simpler trick (just learned): **`./suggar++ -reopen Input.xml_0`**
    - stdout and stderr automatically go to **`out.stdout++ and out.stderr++`**

Running SUGGAR/SUGGAR++ (2/3)

- Principle output: DCI and composite grid files specified in the XML file
- A concise summary of SUGGAR++ info is written to **`summary.log`**

```
start time: Wed Jul  7 18:49:17 2010
host: i16n1
last git commit:
command line: ./suggar++ Input.xml_0
number of processors: 1
number of threads: 1
total number of out: 9657
total number of fringes: 166124
total number of min fringes: 145265
total number of orphans: 199
number of orphans due to poor quality donors: 199
wall clock to perform assembly (seconds): 4.98748
memory used (MB): 1018.83
max interpolation deviation: 7.32747e-15
fringe donor quality: 0.904761
min fringe donor quality: 1
```

Running SUGGAR/SUGGAR++ (3/3)

- For FUN3D applications, SUGGAR++ itself is typically only run one time, to create the composite mesh and initial DCI file
- For moving-body cases, FUN3D calls libSUGGAR++ to compute the DCI data "on the fly"; however the libSUGGAR++ functionality is identical to SUGGAR++
- SUGGAR++ *can* be run in parallel
  - So far scaling achieved has been fairly poor - nowhere near linear, even for small (~8) processor counts
  - Requires a separate partitioning step, which is at odds with current FUN3D parallel-processing paradigm; "optimum" SUGGAR++ partitioning bears no resemblance to optimal flow solver partitioning
  - For these reasons, and since libSUGGAR++ exhibits the same parallel issues, there has been minimal incentive to utilize the parallel capability for SUGGAR++ processing
  - Hopefully SUGGAR++ parallel scaling will improve in the future

Running SUGGAR/SUGGAR++ (3/3)

- Ralph has a “home-brew” interactive visualizer for looking at the overset grid assembly, called GVIZ
  - Allows visualization of the meshes, hole points, fringe points, etc.
  - Very useful for debugging
  - I don’t have enough skill with GVIZ to even begin to explain how to use it

List of Key Input/Output Files

- Input
  - **`Input/Input.xml`** (default; any name OK if explicitly specified)
  - Component grids (name and grid format vary; for FUN3D: vgrid, aflr3, fieldview formats)
- Output
  - Composite grid; name and grid format vary
  - **`filename.dci`** (name set in XML file)
  - **`summary.log`**

FAQ's

- Where do I go to get the correct information on all this and not just your lame, watered-down interpretation?
  - SUGGAR/SUGGAR++ documentation
  - Ralph Noack
  - Sign up for the Google User Group: http://groups.google.com/group/overset-grid-tools/topics (may require invitation from Ralph)

What We Learned

- Very basic SUGGAR/SUGGAR++ XML input
- Setting up a suggar_mapbc file
- SUGGAR/SUGGAR++ execution