Suggar++ Capabilities and Introduction on Usage

Ralph Noack, Ph.D.
President

Celeritas Simulation Technology, LLC

www.CeleritasSimTech.com

Outline

- Brief Overview of Capabilities
- Introduction to Suggar++ inputs
  - Body Hierarchy
  - Transformations
  - Grid Input
  - Boundary Surfaces
- Overview of DiRTlib and LibSuggar

OVERVIEW OF SUGGAR++ CAPABILITIES

Suggar++ Overview

- Built upon experience with SUGGAR
  - Complete rewrite
  - Improved algorithms
- Significantly better than SUGGAR
  - Performance: memory and speed
  - New capabilities
- Integrated with new Pointwise OGA capability

Suggar++ Grid Types

- Structured
  - Curvilinear
  - Analytic
    - Cartesian, Sphere, Cylinder
    - Uniform and stretched
    - Faster, less storage
- Unstructured
  - Tetrahedral, Mixed element, Octree
  - General polyhedral currently in development

Suggar++ Solver Support

- Node- and/or cell-centered assembly
  - Has been used to couple different solvers
    - Overflow (node-centered) & Octree (cell-centered)
- Support for arbitrary structured solver stencil
  - Mark fringes required by flow solver spatial discretization
- High-order discretization support
  - Arbitrary number of fringes
  - High-order interpolation for structured grids

Suggar++ Overview

- Hole cutting
  - Direct cut, analytic, octree, manual
- Overlap minimization using general Donor Suitability Function
  - DSF: is this donor suitable for the fringe?
    - Element volume, diagonal, min edge length
    - Element size (bounding box diagonal)
    - New: distance-to-wall
      - Switch to d-to-wall near surfaces

Suggar++ Support for Overlapping Surfaces

- Integrated surface assembly
  - "Project" on fringe grid onto donor grid
  - Structured and/or mixed element grids
- Integrated USURP to support F&M integration
  - Integration weights available via file, API to transfer without file I/O

Suggar++ Parallel Execution

- Threads for shared memory machines
  - Future: dynamically adjust number of threads
- MPI for distributed memory machines
- Hybrid parallel execution
  - Use MPI to distribute memory across nodes
  - Use threads within a node

Suggar++ Parallel Execution Decomposition Of Grids

- Improve work distribution
  - Use more processors than original composite of grids
- Pre-processing step
  - Writes decomposed grids and input file
- Structured or unstructured grids
- DCI is combined back to original composite grids

Suggar++ Library

- Suggar++ is designed for moving body simulations
- Link into flow solver for integrated dynamic OGA
- libSuggar++ API
  - Control execution
  - Provide moving body transformations
  - Transfer DCI
    - With or without DiRTlib

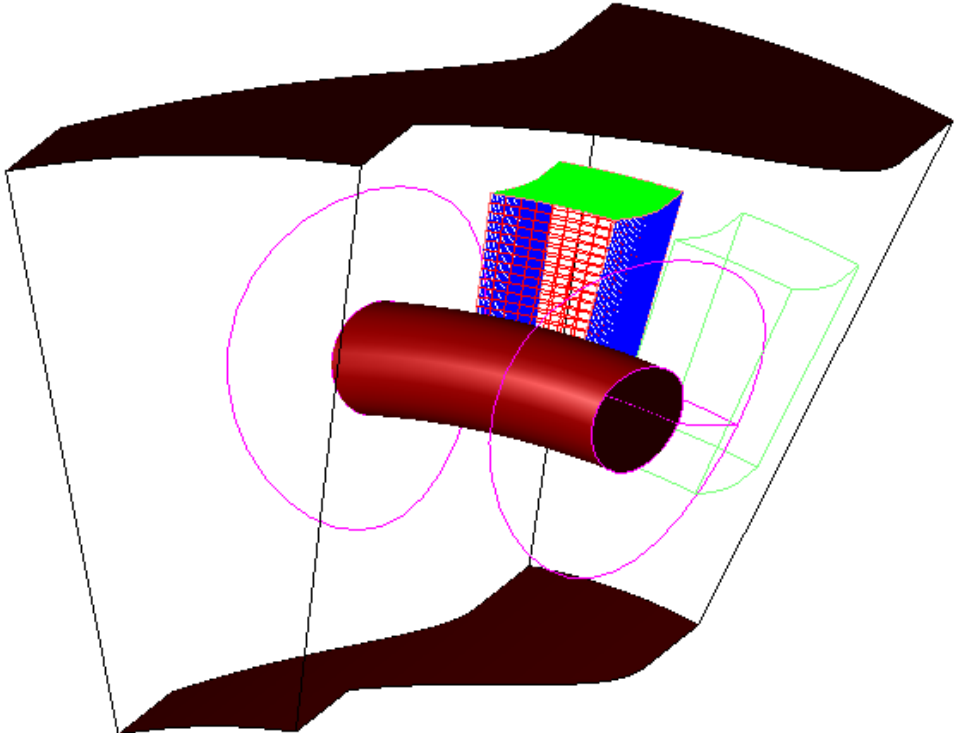
Suggar++ Library: Dynamic Groups

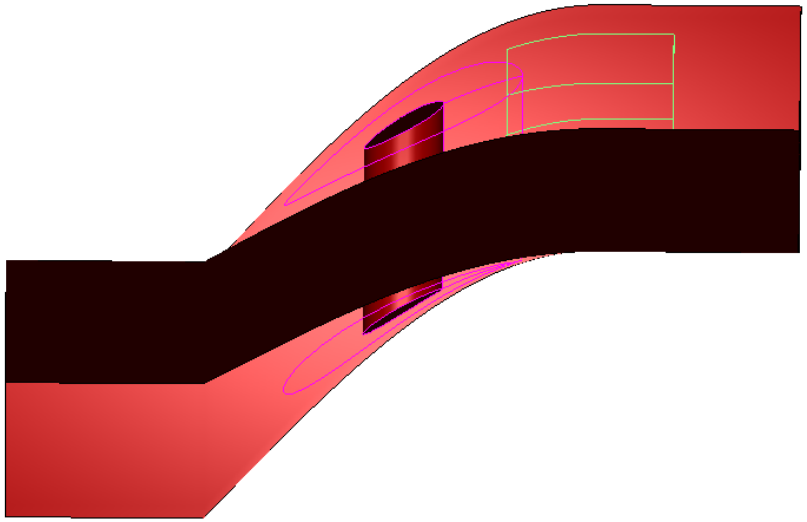
- Suggar++ Dynamic Groups
  - Parallel execution in time
  - One group assigned to T, another to T+1,…
- Overlap OGA execution with flow solution
  - Hide OGA execution time

Suggar++: Periodic Passages

- Turbomachinery simulations
  - Solve 1 blade with periodic boundary conditions instead of full wheel
- Suggar++ donor stencil reaches across periodic boundary to other side of passage
- “Virtual” grid index used to tell solver velocities need transformation

Periodic passage





Suggar++ Unstructured Grid Refinement

- Tetrahedral grids
- Mixed element grids
  - Tet, Hex, Prism, Pyramid
- Refine orphans and candidate donors
- List of elements
  - Could be provided by flow solutions
  - Refine a volume

Suggar++ Unstructured Grid Refinement: Two approaches

- New component grid
  - Copy elements to be refined
  - Adds overlap boundaries
    - Need more overlap
- Altered connectivity
  - Modifies original grid
  - No new overlap boundaries

Composite Grid Formats

- Structured grids
  - Plot3d
- Unstructured grids
  - Some restrictions depending upon input grids
  - VGRID
  - AFLR/UGRID
  - Cobalt
  - Fieldview Unstructured
  - OpenFOAM

Advanced Capabilities

- Deforming Grids
  - Grid point locations are transferred
    - File
    - API to transfer from flow solver
  - Recompute appropriate quantities
- Bodies in Close Proximity
  - Orphans result from insufficient overlap
  - Suggar++ will flag appropriate locations as Immersed
  - Solver must impose solid boundary on internal face
    - Immersed boundary condition

New Capabilities

- Numerous bug fixes and speed improvements
- Improved robustness of direct cut
- Improved performance/consistency for parallel execution
- PEGASUS 5 interpolated donor quality
- Direct DCI transfer for structured grids
  - Eliminates DCI gather to master rank

New Capabilities
Internal Grid Generation

- Additional analytic grids
  - Sphere, cylinder
- Offbody Cartesian grid generation
  - Octree Organized Collection of Cartesian grids
  - Meakin's Offbody Bricks
  - Berger AMR

SUGGAR++ INPUT FILE: XML

What is XML?

- XML stands for eXtensible Markup Language
  - Subset of SGML (Standard Generalized Markup Language)
- Text-based language used to “mark up” data
  - Add metadata (data about the data)
  - Self-describing
  - Not really a language but a set of syntax rules that let you create your own “language”

HTML vs XML

- HTML is designed for a specific application: Document display
  - Specific set of markup constructs
- XML has no specific application
  - It is designed for whatever you use it for.
- HTML syntax rules are sloppy
  - Some end tags can be omitted
- XML has very precise syntax rules

XML Tags/Markup Constructs

- An XML tag is enclosed in “< >”
  - <start>
- Must have an associated end tag
  - Same as start tag but with / after <
  - </start>

```
<name>
   <first>John</first>
   <last>Doe</last>
</name>
```

- Empty elements can have implicit end tag
  - <name></name> can be written as <name/>

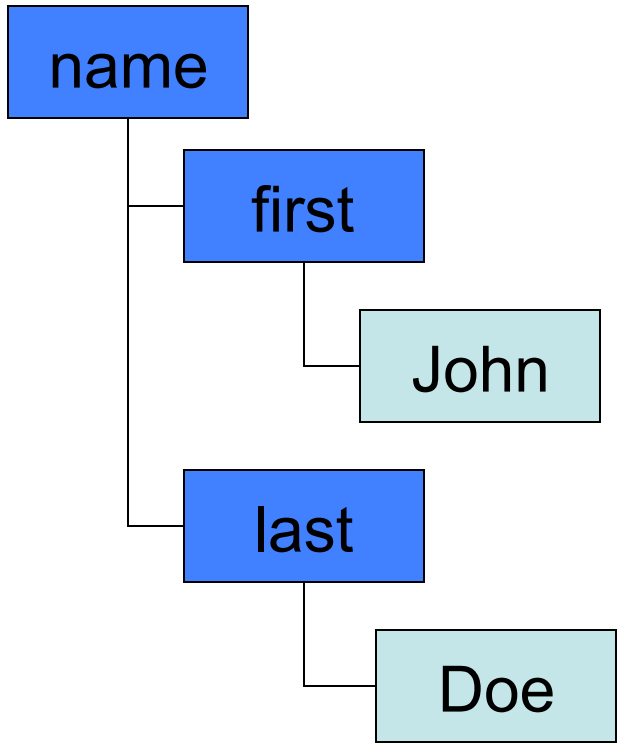
Hierarchies in XML

- Each XML tag defines an item or **element**
- Elements can be embedded inside start/end pair of another element
  - Creates a parent/child and sibling/sibling relationship
  - Children define element content
  - Child element must be closed before a parent can be closed
- Only one root element allowed

Example Hierarchy

- Hierarchy for <name> example

```
<name>
    <first>John</first>
    <last>Doe</last>
</name>
```



XML Elements Can Have Attributes

- Attributes
  - are name/value pairs associated with an element
  - are always attached to the start tag
  - must have a value enclosed in quotes
    (either single or double quotes)
- Place inside of start tag before closing “>”

<body **name=“store”**>

Comments in XML

- Comments in XML
  - start with **<!--** **and** end with →
  - cannot use -- in the comment string

    <!-- cannot embed double dashes -- →
  - cannot be within a tag

    <start <!-- this is illegal--> />

Input Sections

Input Has Three Main Sections

- Global parameter
  - Content of <global>
- Body Hierarchy
  - <body>
- Grid/Surface definition
  - <volume_grid>
    - <boundary_surface>

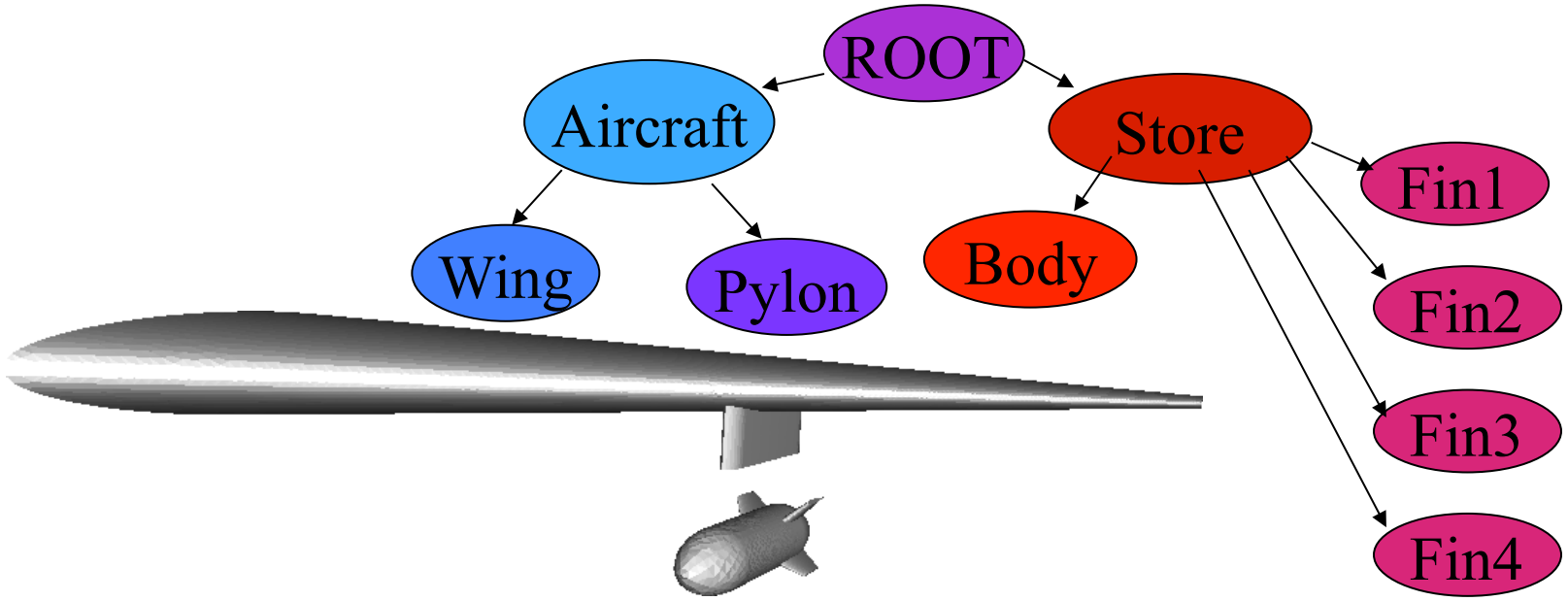
Values Specified by Attributes

- All input values are specified by element attributes
  - <body name="root">
  - Data between elements (PCDATA) is ignored
    - Can use as comments, some restricted characters
- Some attributes are required
  - Will abort if not present
- Other attributes are optional

Body Hierarchy

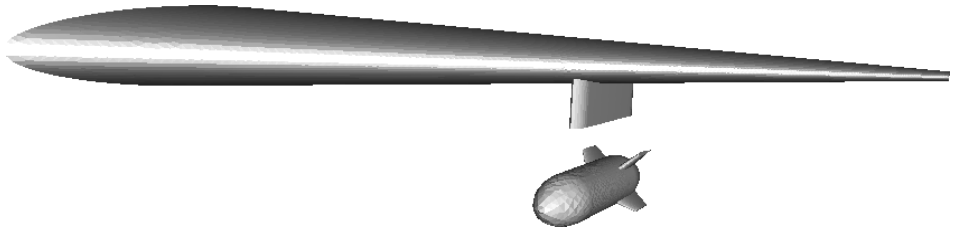
Body Hierarchy Controls Hole Cut

- A hierarchical grouping of grids/bodies minimizes user inputs and controls which grids are cut by which surfaces
- **Siblings cut each other**
  - Geometry in one body (including all children) cuts all grids in a sibling body (including all children)



XML for Wing/Pylon/Store Hierarchy

```
<body name="Root">
    <body name="Aircraft">
        <body name="Wing"/>
        <body name="Pylon"/>
    </body>
    <body name="Store">
        <body name="Body"/>
        <body name="Fin1"/>
        <body name="Fin2"/>
        <body name="Fin3"/>
        <body name="Fin4"/>
    </body>
</body>
```

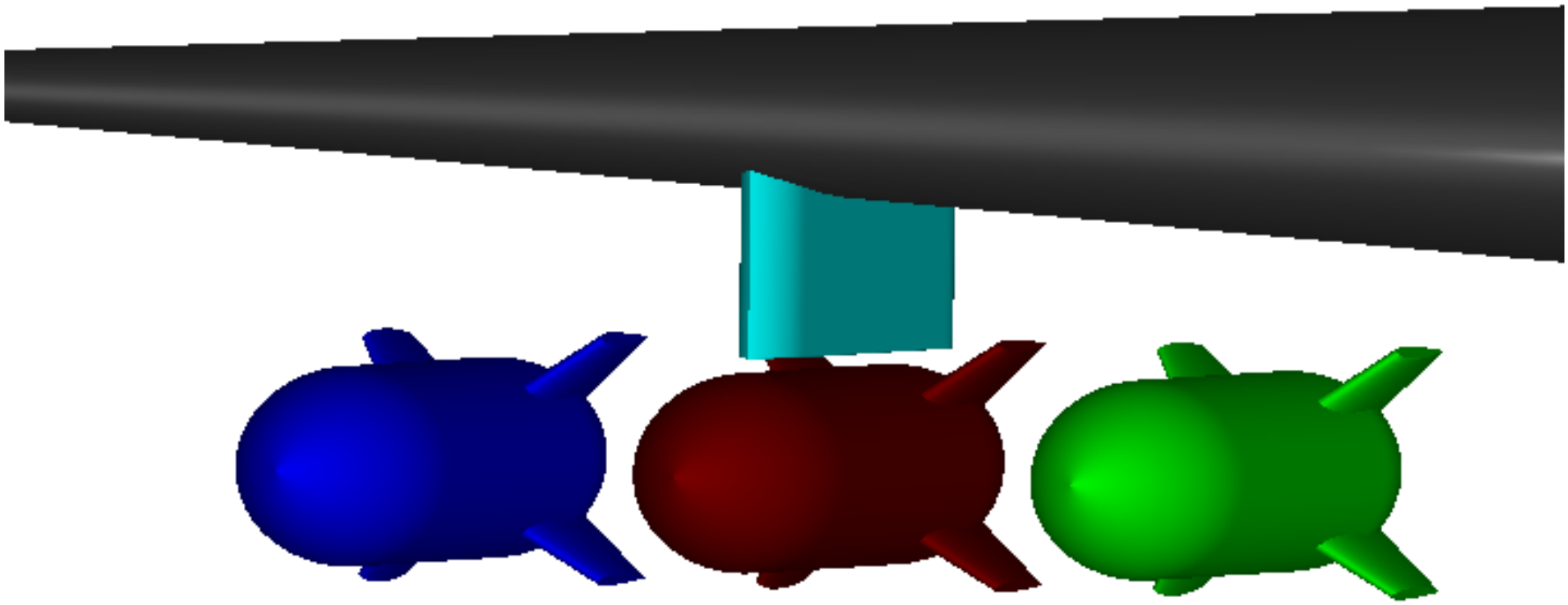


Transformations

Transformations

- Transformations are associated with a body
- Suggar++ has two different types of transformations
  - Static transformations
    - Applied to the grid coordinates on input
    - Original coordinates are replaced by transformed coordinates
  - Dynamic transformations
    - Flags the body as moving
    - Grid coordinates are left in original coordinates
      - Transformations are always from original coordinate system
      - Not cumulative
    - Transformations are used internally during execution
    - Output grids are transformed
- Transformations are hierarchical
  - Child body transformations are relative to the parent

Wing/Pylon With 3 Stores



Wing/Pylon With 3 Stores Input using includes

```
<body name="center-store">
    <include filename="Input/store.xml"/>
</body>

<body name="inboard-store">
    <transform> <translate axis="y" value="-2"/> </transform>
    <include name_suffix="-inboard" filename="Input/store.xml"/>
</body>

<body name="outboard-store">
    <transform> <translate axis="y" value="2"/> </transform>
    <include name_suffix="-outboard" filename="Input/store.xml"/>
</body>
```

Component Grid Input

Suggar++
Current Grid Types

- Structured
  - Curvilinear
  - Analytic
    - Cartesian (uniform and non-uniform)
      - Uniform can be defined in input file
    - Cylindrical
    - Spherical
- Unstructured
  - Tetrahedron
  - Mixed element
    - Tet, Hex, Prism, Pyramid
  - Octree-based Cartesian

<volume_grid> Element

- Parent element is <body>
- Associates a grid with a body
  - Actual grid to be used is specified with the filename attribute.
- A body can have more than one <volume_grid> child
  - Cannot have child <body> and child grids!
- Required attribute is name="grid name"

```
<body name="Wing">
   <volume_grid name="wing grid”>
   </volume_grid>
</body>
```

<volume_grid> filename, style attributes

- Grid file is specified with the attributes…
  - *filename*=*“file”*
  - *style*=*“style”*
- Both are required

```
<volume_grid name=“wing”
    filename=“Grids/wing.g” style=“p3d”/>
```

Boundary Surfaces

Suggar++ Boundary Conditions

- Suggar++ boundary conditions do not need to “match” flow solver boundary conditions
- Some cases where there may be a loose mapping
  - Flow solver “wall” ~ Suggar++ “solid”
  - Flow solver “farfield” ~ Suggar++ “farfield”
  - Block-to-Block, etc.

Suggar++ Boundary Conditions

- Many cases where they must be different than solver boundary conditions
  - Hole cutting geometry must be closed/“water tight”!!!
    - Surface is not solid geometry but must be used as hole cutting geometry
      - Inlet/Exhaust surface
  - Solver has solid surface but is not needed as cutting surface
    - Tunnel walls but no grids extend past tunnel walls
  - Suggar++ has a limited set of BCs

Boundary Surface Creation

- Boundary surfaces are automatically created for unstructured surface patches
  - Boundary conditions are automatically set for VGRID files
    - Internal mapping between USM3D BCs and Suggar++ BCs
- Must be explicitly defined for structured grids
  - If not defined surface is created with a boundary condition of “overlap”
- Multiply defining a surface is allowed
  - But is not recommended
  - Useful in limited circumstances

Specifying Boundary Conditions for Unstructured Grids

- Boundary surfaces are created automatically
- Boundary conditions can be specified
  - in the input XML file
  - in auxiliary files
    - for Vgrid file sets
      - projectName.suggarbc
    - for other unstructured grid files
      - gridFilename.suggar_surface_bc
      - gridFilename.suggar_mapbc
- An auxiliary file can also be used to specify solver BCs in the output composite grid
  - filename.solver_bc

<boundary_surface> Element

- Parent element is <volume_grid>, <cartesian_grid>,….
- It is a container element for content
- Specifies the surface and boundary condition type for boundary surfaces in the parent grid
- Required attribute is ***name***=*“surface name”*

<boundary_surface name=‘wing’>
</boundary_surface>

<region> Element

- Parent element is <boundary_surface>
- Specifies the boundary surface in a **structured** grid.
- Required attributes
  - ***range1***=*"start:end"*
    - Index range in the first index (I for IJK, J for JKL)
  - ***range2***=*"start:end"*
    - Index range in the second index (J for IJK, K for JKL)
  - ***range3***=*"start:end"*
    - Index range in the third index (K for IJK, L for JKL)
  - Negative number counts backwards from the end:
    - -1 is the same as max value, -2 is same as max-1 value, etc.
  - Can also use *min, max, all*

```
<boundary_surface name='wing'>
   <region range1='21:-21' range2='1:-1' range3='1:1'/>
</boundary_surface>
```

<boundary_condition> Element

- Parent element is <boundary_surface>
- Specifies the boundary condition to be applied at the boundary surface
- These are SUGGAR BCs and don't necessarily match the flow solver BCs
- Required attribute ***type***=*"boundary type"*

```
<boundary_surface name='wing'>
    <region range1='21:-21' range2='1:-1' range3='1:1'/>
    <boundary_condition type='solid'/>
</boundary_surface>
```

<boundary_condition> types

“overlap” An overset or overlap boundary surface.

“solid” A solid boundary and will be used to define the hole cutting geometry.

“symmetry” A symmetry non-overset boundary surface. The grid points on the symmetry boundary will be used to determine the value of the symmetry plane.

“axis” A singular axis where all the grid points in one of the computational coordinates are collapsed to a point.

“periodic” A periodic boundary in the structured grid. Both the min and max boundary surfaces should be specified.

“cut” The surface is a cut boundary in the structured grid. Both the min and max boundary surfaces should be specified.

“block-to-block”, “block-block”, “block2block” The surface is a block-to-block interface to another grid. Requires additional attributes.

“freestream” or “farfield” A freestream non-overset boundary surface

“non-overlap”, “non_overlap”, “nonoverlap”, “non-solid” , “non-*” The surface is an unspecified non-overset boundary.

<boundary_condition> optional attributes

- <boundary_condition> has an optional attribute ***solver_bc***=*"bc string"*
- Allows the user to specify a boundary condition for the surface to be output to a cobalt.bc file
- If ***solver_bc*** is not included, the SUGGAR BC is output.

```
<boundary_condition
    type='solid'
    solver_bc="viscous_wall"/>
```

Solver BCs for Unstructured Composite Grid

- Suggar++ will write selected solver boundary condition files for the composite grid
  - Vgrid

    project.mapbc file
  - Cobalt

    composite_grid_filename_cobalt_bc
  - Other unstructured grid formats

    composite_grid_filename.suggar_mapbc

Setting **Solver** BCs for Unstructured Composite Grid

- **Solver** BCs can be set from auxiliary files associated with each component grid
  - Vgrid

    project.mapbc file
  - Cobalt
    - grid_filename_cobalt_bc
    - basename.cobalt_bc
    - Where basename = grid_filename with trailing suffix removed
  - Other formats
    - grid_filename.solver_bc
    - grid_filename.suggar_mapbc

Overlapping Surface Grids

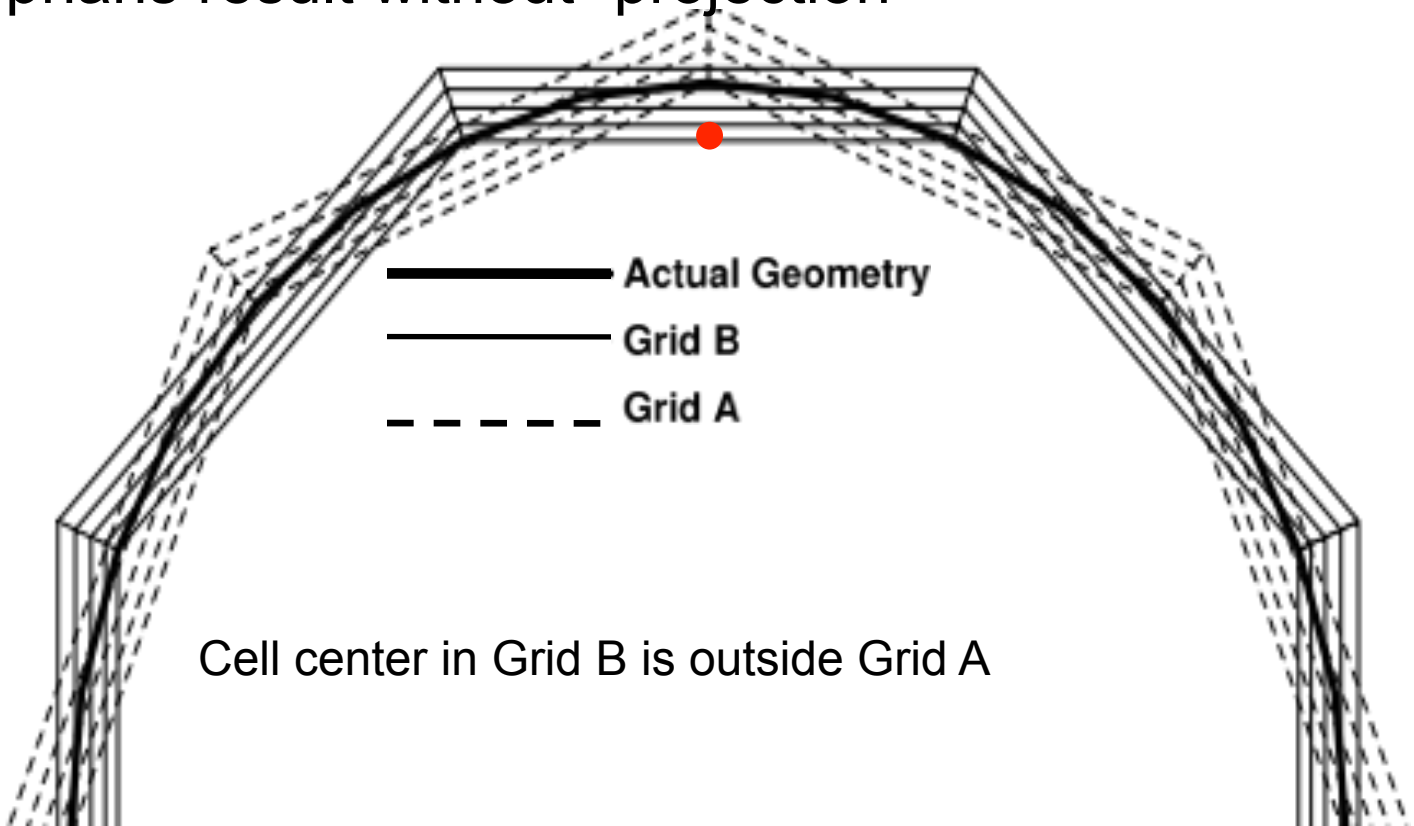
- Overlapping surface grids present several additional complexities
  - Surfaces in a grid can be associated with different geometry components
  - Overlapping surfaces will have different discrete representations
  - Overlapping surfaces require special treatment to eliminate double counting in Force and Moment integration

Overlapping Surface Grids: Different Discrete Representations

- Surfaces that overlap on geometry with curvature will have different discrete representations
- Difficulties arise when the tangential spacing is “large” relative to the curvature and the normal spacing
- Special procedures are required to properly find appropriate donors

Overlapping Surface Grids: Different Discrete Representations

- "Projection" of one surface onto the other is required to properly locate donors
- Orphans result without "projection"



Surface Assembly

- Grids are not actually projected
  - Grid points are not changed
- Fringe points will be shifted appropriately during the donor search
- Surface Assembly procedure is use to find the shift for each fringe point
  - Relative to overlapping surface in each donor grid
    - A fringe point will have different shifts/offsets for each donor grid

Surface Assembly Procedure

- For each surface grid point (node-centered) or face center (cell-centered)
  - Location appropriate donor faces in overlapping grid
  - Find normal distance from surface location to the surface donor face
    - Save deviation and the surface normal
  - Adjacent element is the volume donor for node-centered surface points

Volume Donor Search Uses Surface Assembly

- Volume fringes will be shifted using the surface assembly deviation
  - Shift will decay for points away from the surface
  - Interpolation deviation will be computed using the shifted fringe point
    - Flow solver will not have the shift so computing the interpolation deviation in the flow solver will not give the same result

Integral Surface Assembly

- SUGGAR uses a separate “surfasm” utility to obtain the deviation between surfaces
  - donors.xml contains surface donors and displacement
- Suggar++ performs the surface assembly internally
  - Enabled with <surface_assembly/> element

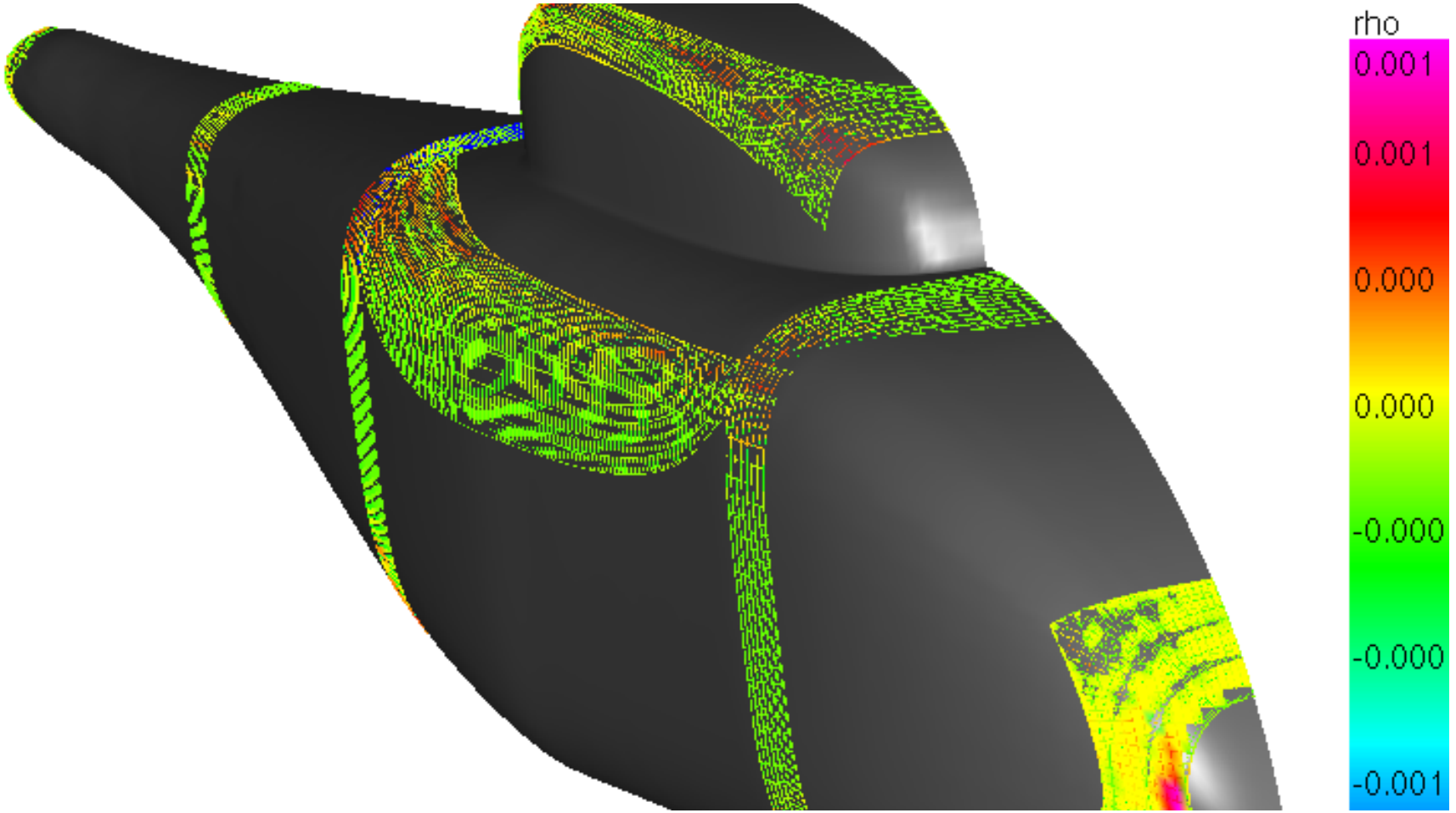
<surface_assembly/> element

- Parent element is <global>
- Required attribute
  - max_deviation_allowed="value in grid units"
    - Ignore surface overlap if deviation is larger than the specified value
- Optional attribute
    - max_angle_deviation_allowed="angle in degrees"
    - Ignore surface overlap if angle between donor face and normal at surface fringe point is larger than the specified value
- <surface_assembly max_deviation_allowed="0.0001"/>

Checking Surface Assembly

- Work/max_surface_assembly_deviation.txt
  - Surface deviation for each surface in all grids
- Work/SurfaceDeviation/Grid-#-name/surfname
  - # is the composite grid index
  - name is the grid name
  - surfname is surface name
  - Directory contains PLOT3D grid and Q file to visualize the deviation:
    - Grid is multi-block PLOT3D, with iblank, single precision, unformatted
      - DonorGrid-#-name.p3dwibu
    - Q is multi-block PLOT3D Overflow Q file, with iblank, single precision, unformatted, one dependent variable: surface deviation
      - DonorGrid-#-name.p3dqou

Visualizing Surface Deviation



Integrating Force And Moments On Overlapping Surfaces

- Special treatment to eliminate double counting in force and moment integration
  - Panel weights
    - Weight factor between 0 & 1 for each integration surface face/panel
  - Single valued (water tight) integration surface
    - Remove overlap, glue remaining portions of original surfaces together using new triangles
- Tools
  - FOMOCO
  - USURP/PolyMixsur

Suggar++ has integrated USURP capability

- Similar but not identical to the USURP utility
  - Different coding
  - Uses CLIPPER for polygon clipping
    - more robust than GPG used in USURP
  - Triangulation routines are different than USURP
- Panel weights
  - Included in DCI file: Can be retrieved via DiRTlib
  - Written to files
- Can create zipper grid
  - Not sufficiently robust

<usurp> Element

- Parent element is <global>
- No required attributes
- Lots of optional attributes

```
<global>
   <usurp/>
…
```

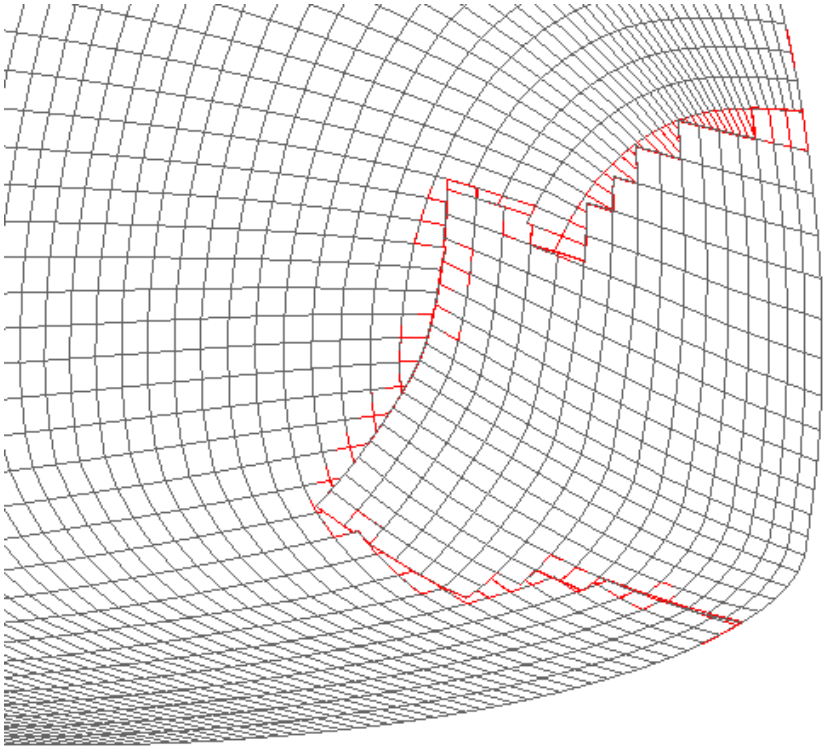
<usurp> Element Output Files

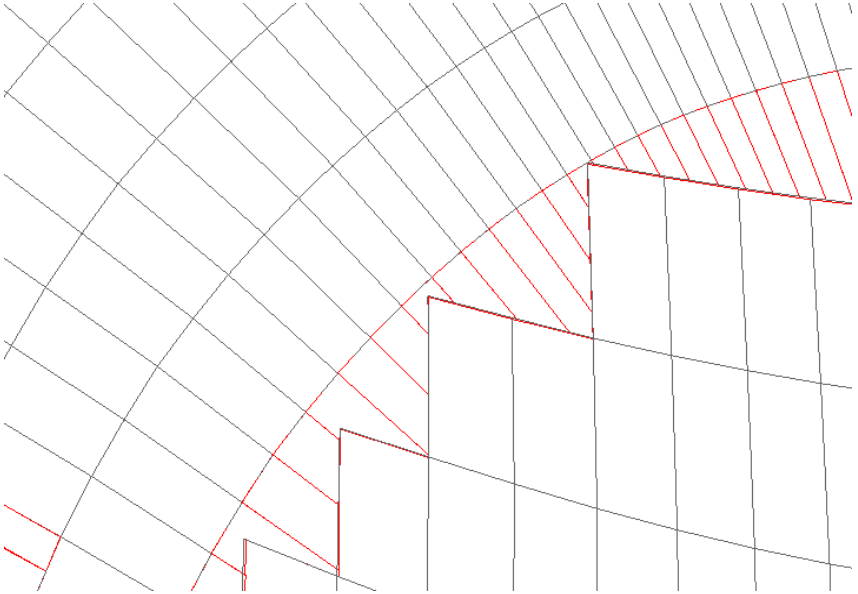
- panels_weights.txt
  - List of panel index, area_ratio, area, ratio*area, is_clipped, number_contours
- Surface panels and triangles
  - Tecplot file: usurp-surfaces.dat
  - Flex file for gviz: usurp-surfaces.flex
- Panels and clipped polygons
  - Flex file for gviz: usurp_panels.flex

<usurp> Element Output Files

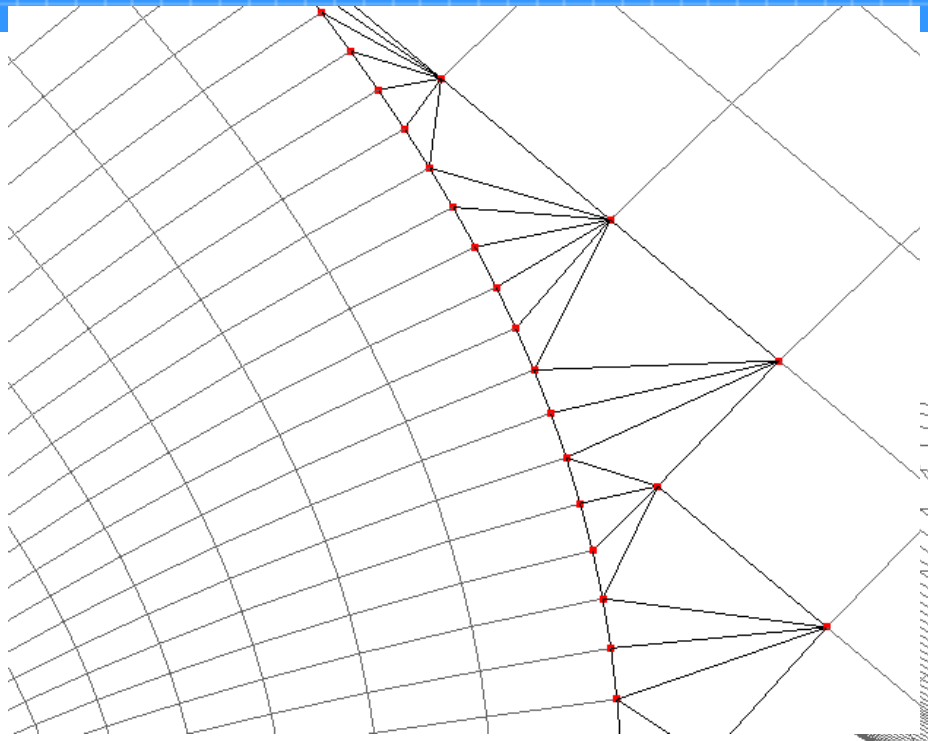
- If ***create_watertight_surfaces***="yes"
- Zipper grid:
  - Quads and zipper triangles
    - zipper_surface_faces.flex
  - Zipper triangles with quads replaced by triangles
    - zipper_surface_faces_all_tris.flex
    - usurp-triangles.dat (Tecplot file)

Suggar++ USURP output for Robin Helicopter Fuselage

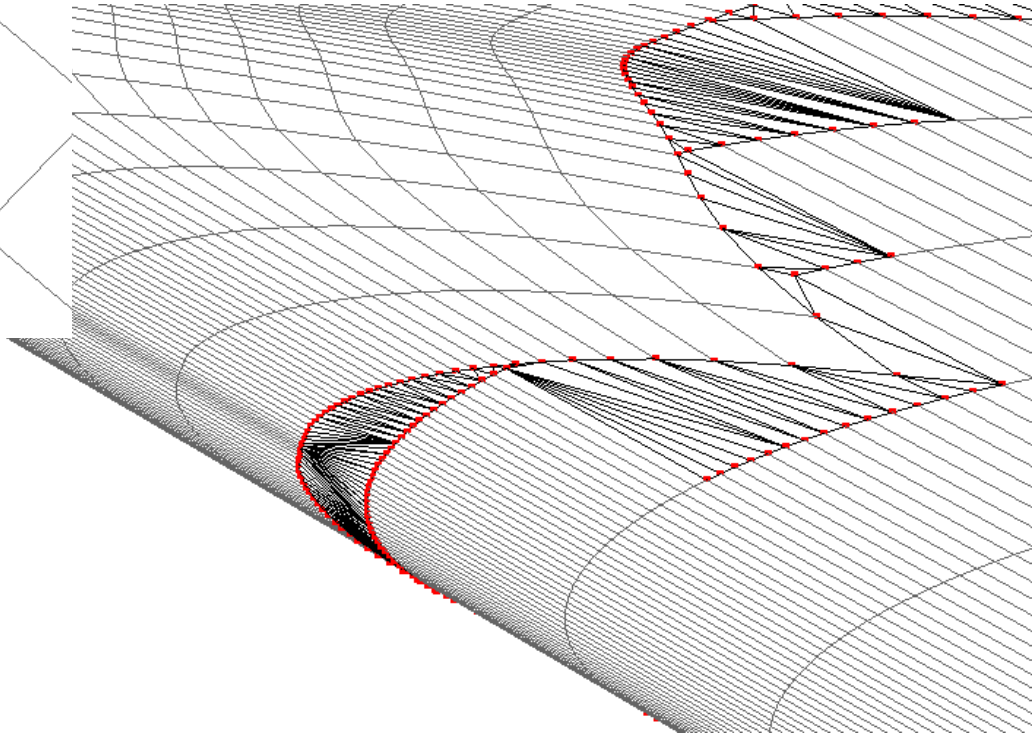




Suggar++ USURP output for WingBody

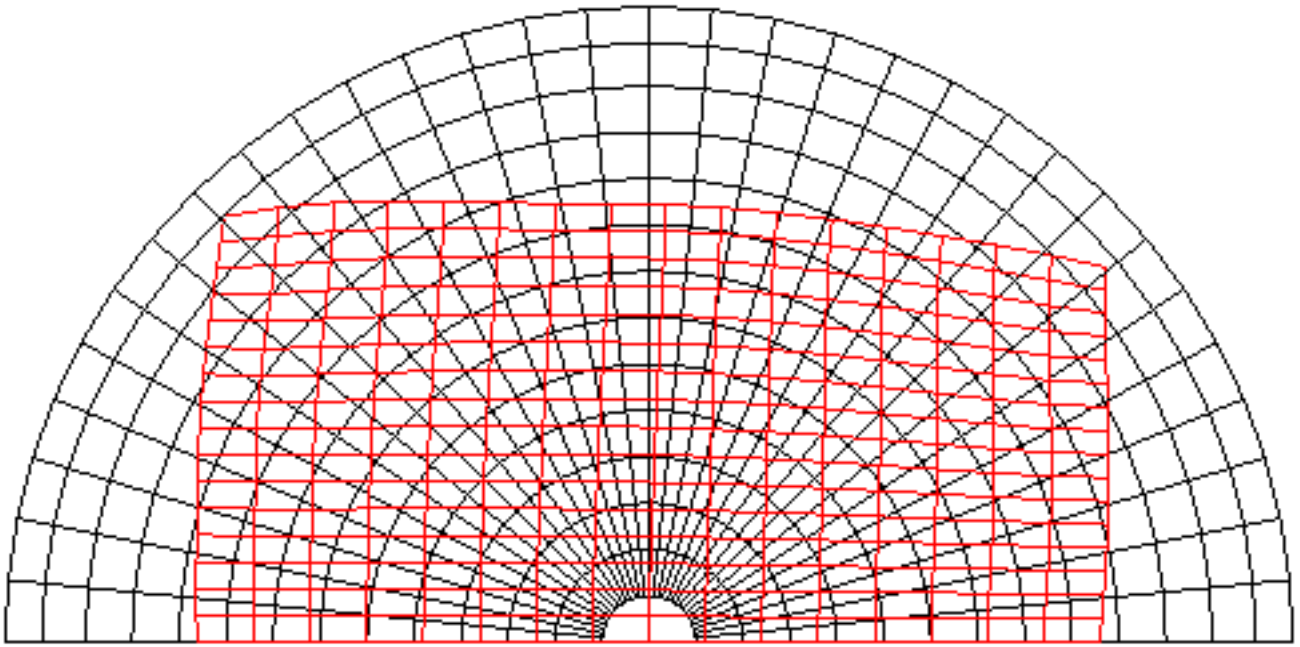


Zipper grid:
Triangle contain only points in the original grid



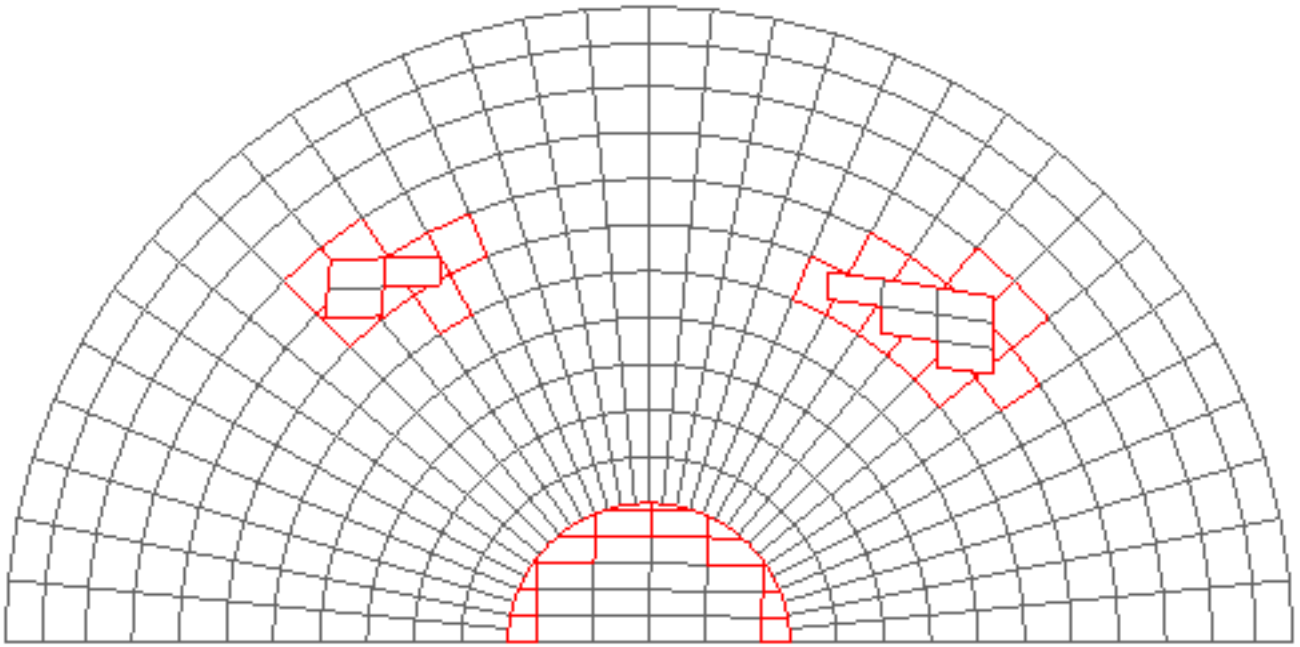
<usurp> Control Attributes

- ***polygon_ranking_basis***='panel|patch'
  - Select the approach for prioritizing the choice of panels. Default value is 'panel'.



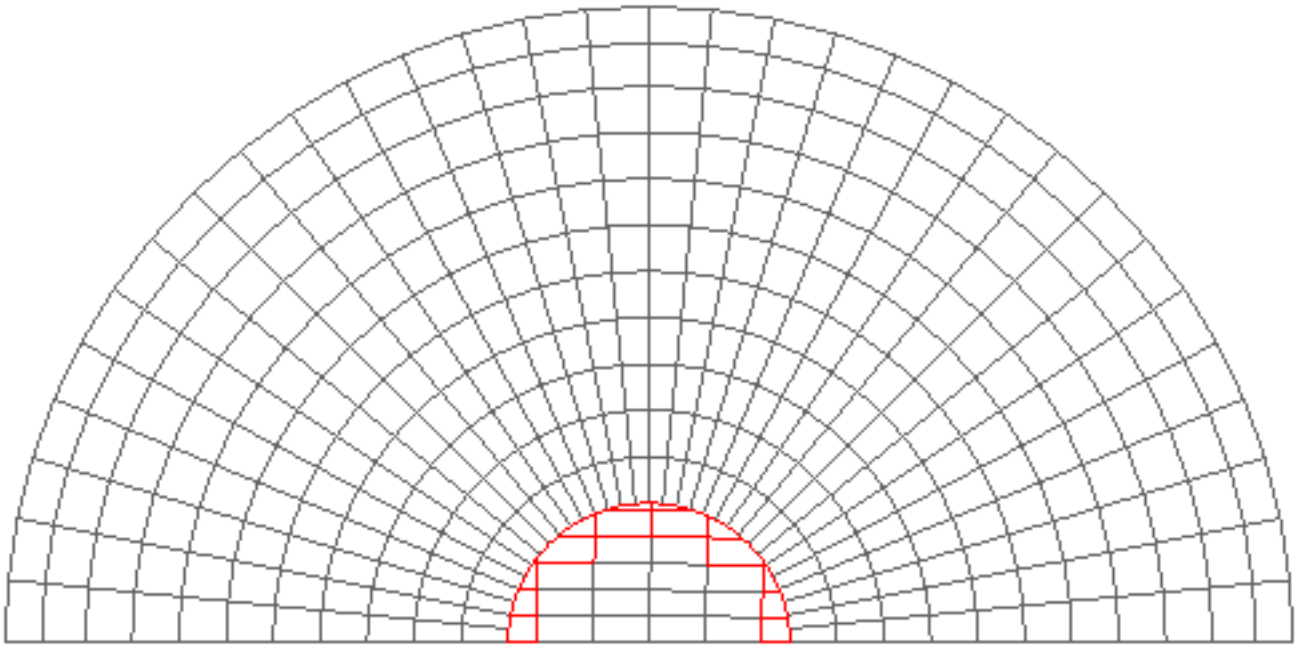
polygon_ranking_basis=’ panel'

- ***polygon_ranking_basis***=’panel'
  - Priority is local: panel/face with smallest area

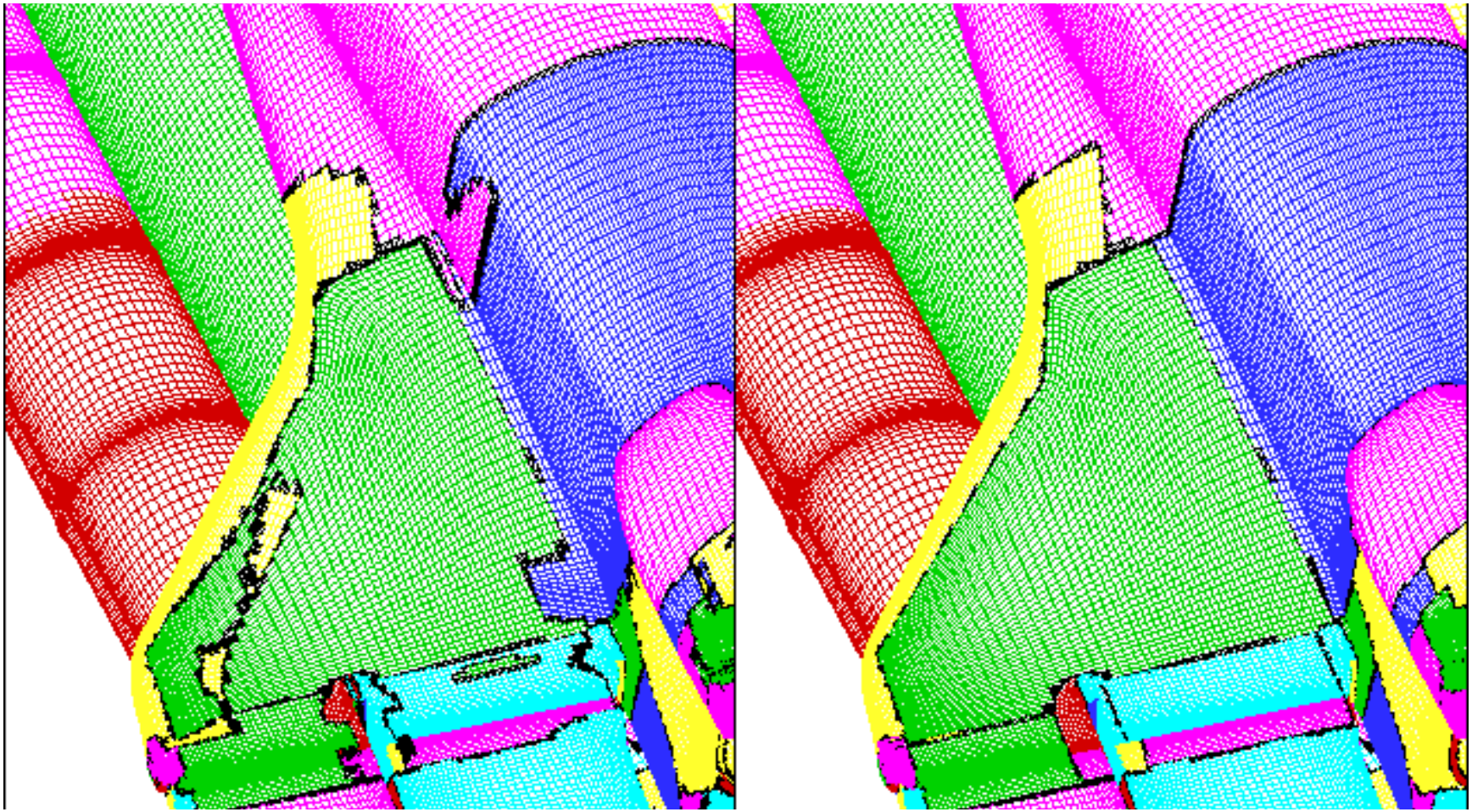


polygon_ranking_basis='patch’

- ***polygon_ranking_basis***='patch’
  - Priority is based upon the surface with the most surface fringes



More Complex Example



Utilities Provided With Suggar++

Utilities Provided With Suggar++ Grid Refinement/Derefinement

- RefineGrids
  - Refine structured grids by factor of 2
- DerefineGrids
  - Derefine structured grids by factor of 2
- Scripts to generate a sequence of derefined grids

Utilities Provided With Suggar++

- Convert
  - Convert between different unstructured grid formats
- Mirror
  - Mirror a set of structured grids and Input.xml
- report_number_grids
  - Output the number of component grids
- cmp_dci
  - Compare the DCI in two files

Suggested Work Process

General Suggestions
Building Input

- Build input in pieces
  - Or use <skip> </skip> to hide complete subtrees
- Check and Indent XML file
  - xmllint -format
  - xmlformat.pl
  - Emacs
- Visualize surfaces
  - Especially solid surfaces
  - Color collar surfaces differently
    - Put “collar” in surface name
      - <boundary_surface name=“kmin-solid-collar-with-sting”>

Be Very Careful With

- <boundary_surface const_coord=“”>
  - Make sure have right value on right surface
  - Look at composite grid
- Reorientation of grid blocks without appropriate changes to input
- Manual cutting and symmetry planes
  - Can cut wrong direction

General Suggestions
Run Suggar++

- Redirect the Suggar++ output
  - suggar++ -reopen
- During initial testing reduce wall clock time
  - suggar++ -ignore-composite-grid
  - suggar++ -ignore-minimize-overlap
- Check *suggar++progress* during execution
  - One line added at start of each stage of execution

Suggested Directory Structure

- We suggest putting critical input files in directories to minimize the chance of accidental removal
  - Put all your component grid files in Grids/
  - Put your input files in Input/
    - Suggar++ will default to read Input/Input.xml
      - *"suggar++ Input/Input.xml"* is same as *"suggar++"*

Suggest Use Scripts

- We suggest using standard scripts
  - Run
    - Execute Suggar++ and check for errors
  - Clean
    - Remove (LOTS) of files that Suggar++ can write

Example Run Script

```
#!/bin/bash

STDERR=out.stderr++

$SUGGARPP_OPT_EXE -reopen $*

EXIT_STATUS=$?
if [[ $EXIT_STATUS != 0 ]];
then
    echo "FAILURE: suggar++ has failed with exit status $EXIT_STATUS"
    grep "Error:" $STDERR

    exit $EXIT_STATUS
fi

if [[ -e summary_zipper.log ]]; then
    cat summary_zipper.log >> summary.log
fi
```

Example Clean Script

```
rm -f allgrids.p3dudl* *.dci* out* *log *gress
rm -f panels_weights.txt Suggar++Error.backtrace
rm -f usurp* zipper_*.flex cut_elements*
rm -rf Work
rm -rf *_trace_*
```

General Suggestions
Check Suggar++ Output

- Look at
  - summary.log
  - Standard error output file
    - -reopen will write to out.stderr++
- Visualize the DCI
  - Look at orphans
  - All blanked points
    - May have flood fill leak if entire grid is blanked out

Suggar++ and The New Pointwise Release

Pointwise Has Integrated Interface To Overset Grid Assembly!

- Currently supports PEGASUS 5 and Suggar++
- Within pointwise
  - Allows user to define inputs via GUI
    - Input definition is via XML file
  - Run OGA
  - Visualize results
  - Modify grid system
  - And more…

Suggar++ Support In Pointwise

- Some Suggar++ input elements are not visible in pointwise GUI
  - Handled internally in pointwise
    - <volume_grids>
    - <boundary_surface> and content
  - Not supported in pointwise
    - Analytic grids
      - <cartesian_grid>, <cylindrical_grid>, <spherical_grid>

Suggar++ Input Definition Support In Pointwise

- New input definition file can be provided with Suggar++ release
- Replace installed file or set an environment file

Overview of DiRTlib and LibSuggar

DiRTlib

- DiRTlib is: Donor interpolation Receptor Transaction library
- It is a solver neutral library to provide the required capability for using overset composite grids
  - Work with most ANY flow solver
  - Knows nothing of solver connectivity
  - Does not depend upon a specific solver storage

DiRTlib Design Goal

- Goal is to minimize modifications required to flow solver
  - Provide a few functions to DiRTlib
    - Interface to solver data
  - Insert a few function calls
- Most solvers utilize an IBLANK array
  - Not required but in most cases easiest approach

DiRTlib Capabilities

- Supports variable number of Dependent Variables
- Segregated Solvers
- Single Unstructured Grid
  - Unstructured grid solver sees a single composite grid.
  - Domain connectivity is based upon set of component grids
- Parallel Execution
  - Decomposition
    - Defined by solver
    - Can decompose structured grids

DiRTlib Capabilities

- Domain Connectivity Information
  - Files: SUGGAR/Suggar++, Pegasus 5
  - LibSuggar/libSuggar++
- Donor Details
  - Some solvers need to build interpolation into linear solution
- Relative Motion
  - What cells are moving
  - What is transformation to position body

Using DiRTlib

- Solver interface functions
  - DiRTlib does not (or rarely) directly access solver storage
  - Solver provides interface functions that DiRTlib calls to get/put values in solver storage
- Add a few calls to control execution
  - Initialize library
  - Perform interpolation/apply fringe values

Programming Language Support

- Library is written in C
  - Functions names start with **drt_**
- FORTRAN interface written in C
  - Functions names start with **drtf_**
  - Supports names with 0,1,2 appended underscores
  - Long function names are abbreviated
  - drt_fortran_interface.c provides FORTRAN wrappers
  - libdirt_interface.f90 can be compiled to provide module that provides function prototypes

libSuggar: DC API

- Domain Connectivity (DC) API (libSuggar) to allow integrated overset grid assembly process
- Flow solver calls DC API (libSuggar) to control execution
  - libSuggar can be called from dedicated rank
    - Required splitting MPI communicator
    - Modify solver to execute DC only on dedicated rank
    - Distributes SUGGAR memory usage
  - Can still write/read DCI file
- Domain Connectivity Exchange (DCX) calls allow DCI to be transferred via calls without writing/reading DCI file

Programming Language Support

- Library is written in C or C++
  - Functions names start with **dc_ or dcx_**
- FORTRAN interface written in C
  - Functions names start with **dcf_ or dcxf_**
  - Supports names with 0,1,2 appended underscores
  - Long function names are abbreviated
  - F90 module can be compiled to provide function prototypes

Example DiRTlib and LibSuggar++ Calls

- Will present a set of DiRTlib and LibSuggar++ function calls
- Illustrative of how few calls are required
  - Not necessarily all that are required or correct order
- Parallel execution requires conditionals so some calls are only executed on specific processors

Example DiRTlib and LibSuggar++ Calls Initialization

- drt_set_num_data_values_all_grids(N)
- drt_Init(PutDataValue,GetDataValue,...)
- dcx_set_dci_master_rank_in_group_comm(0)
- drt_rank_dci_only()
- drt_rank_flow_only()
- drt_pll_init(0,0)
- dc_init()

Example DiRTlib and LibSuggar++ Calls Provide DiRTlib with Solver Decomposition

- drt_init_str_subgrid_decomposition_map()
- drt_map_str_subgrid_to_rank(...)
- drt_end_str_subgrid_decomposition_map()

- Other calls for unstructured grids

Example DiRTlib and LibSuggar++ Calls Time Step: Specify Body Transformations

- dc_begin_motion_input()
- dc_add_motion_input(...)
- dc_end_motion_input()
- dc_parse_motion()

Example DiRTlib and LibSuggar++ Calls Time Step:

- dc_compute_dci()
- drt_get_dci()
- drt_generate_transmit_apply()
- dc_release_dci()

Commercial distribution and support
for Suggar++ provided by

Celeritas Simulation Technology, LLC

http://www.CeleritasSimTech.com

Exportable under an EAR-99 license