Suggar++ Capabilities and Introduction on Usage

Ralph Noack, Ph.D.
President

Celeritas Simulation Technology, LLC

www.CeleritasSimTech.com

Outline

- Brief Overview of Capabilities
- New Features
- Introduction to Suggar++ inputs
  - Body Hierarchy
  - Transformations
  - Grid Input
  - Boundary Surfaces
- Suggar++ and Pointwise

Overview of Suggar++ Capabilities

Suggar++

- The premier general overset grid assembly
- Useable with most any solver/grid system
- Available world wide
  - EAR-99 export license

Suggar++ Grid Types

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

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, hybrid, 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)
    - Distance-to-wall
      - Switch to d-to-wall near surfaces

Suggar++ Support for Overlapping Surfaces

- Integrated surface assembly
  - "Project" fringe grid onto donor grid
  - Structured and/or mixed element grids
    - Unstructured grid must have layers
  - Overlapping surfaces with relative motion
- Integrated USURP to support Force & Moment integration
  - Integration weights available via file, API to transfer without file I/O

Suggar++ Parallel Execution

- Threads for shared memory machines
- 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

- Decompose to 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 component 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
    - Improved capability to send DCI to flow ranks

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

Composite Grid Formats

- Structured grids
  - Plot3d
  - Gridgen
- Unstructured grids
  - Some restrictions depending upon input grids
  - VGRID
  - AFLR/UGRID
  - Cobalt
  - Flex

Suggar++
New Capabilities

New Capabilities
Speed and Robustness

- Numerous bug fixes and increased speed
- Improved robustness of creation of water-tight surface in <usurp>
- Improved dual-grid donor for cell-centered structured grids
  - Finds donor hex stencil of cell centers
  - Tri-linear interpolation
    - Monotonic except near boundaries
      - Option to reduce to quad at boundaries

New Capabilities
Hybrid hole cutting

- Different approaches for
  - Static hole cutting
    - Saves static holes so no need to cut within a dynamic group
  - Dynamic hole cutting
- Combine approaches for robustness
  - Example: Octree + donor search + direct cut

New Capabilities

- Improved approach for overlap minimization with embedded grids
- <body> and <volume_grid> can be siblings
- Plugin support for writing composite grid
- Input component and output composite grids can be compressed

Supported Platforms

- Linux
  - 64 bit
- Mac OS X
  - 64 bit
- Windows
  - 64 bit
  - No MPI

Suggar++ Input

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/>

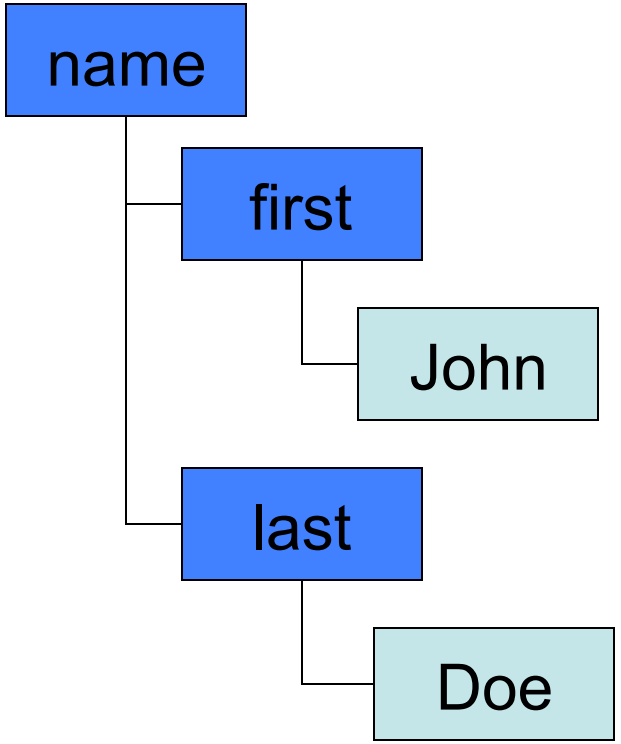
Hierarchies in XML

- 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--> />

Suggar++ Input

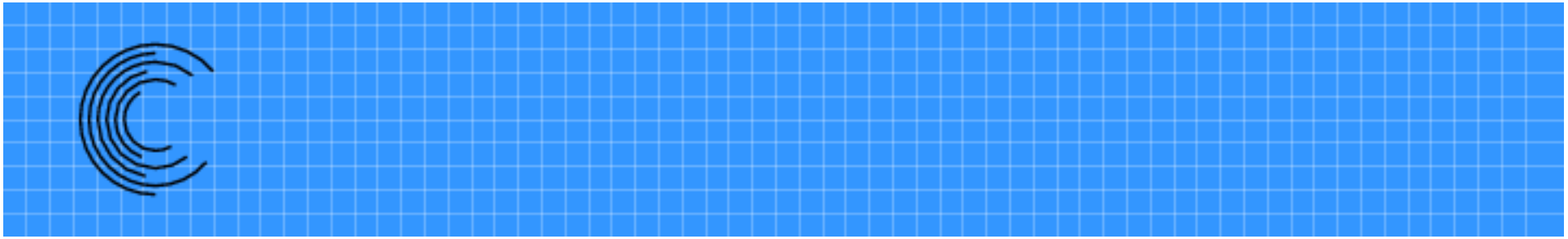
Input Sections

Input Has Three Main Sections

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

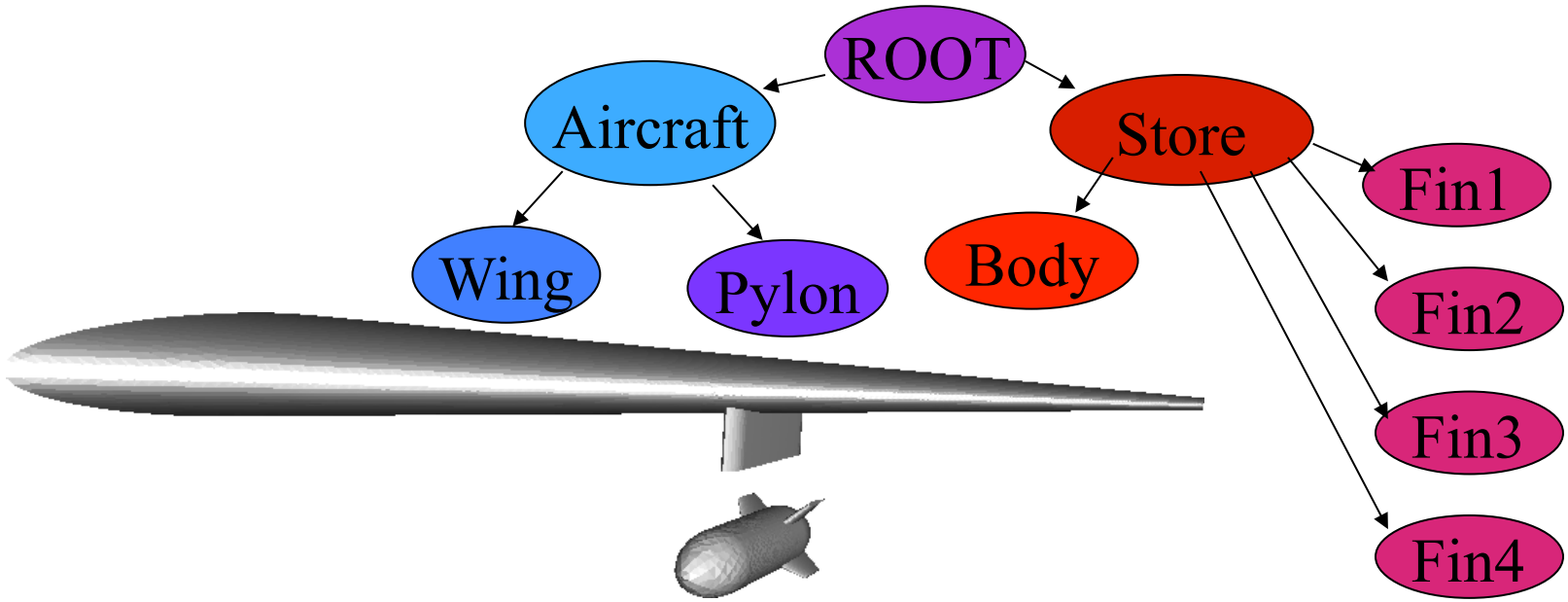


Suggar++ Input

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)



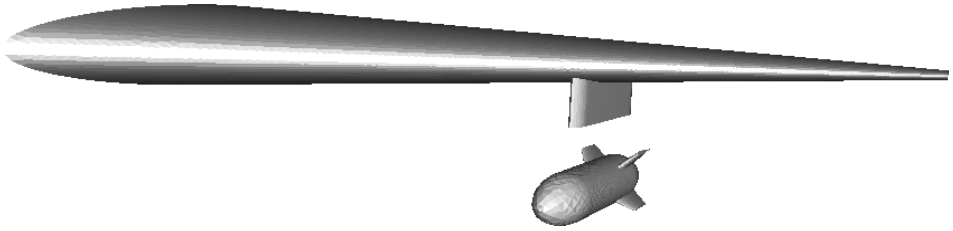
XML for Wing/Pylon/Store Hierarchy

```
<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>
```



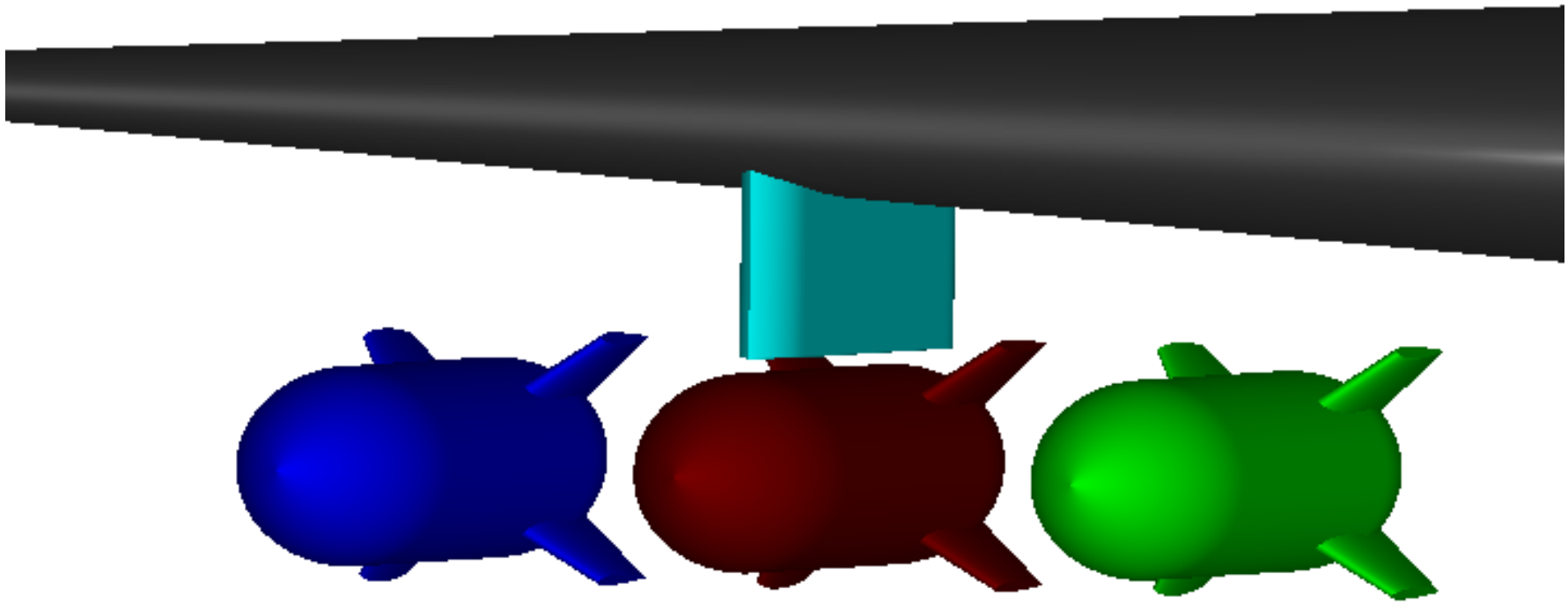
Suggar++ Input

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

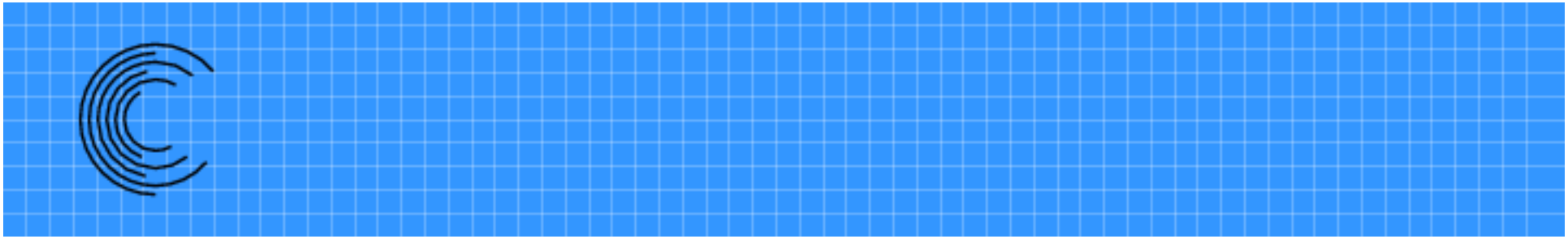


Wing/Pylon With 3 Stores
Input using includes

```
<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>
```



Suggar++ Input

Component Grid Input

Suggar++ Grid Types

- Structured
  - Curvilinear
  - Analytic
    - Cartesian (uniform and non-uniform)
      - Uniform can be defined in input file
    - Cylindrical
    - Spherical
    - Faster, less storage
- Unstructured
  - Tetrahedron
  - Mixed element
    - Tet, Hex, Prism, Pyramid
  - General polyhedral
  - 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
- 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”/>
```

Suggar++ Input

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

Suggar++ Boundary Surface Creation

- Automatically created for unstructured surface patches
- Must be explicitly defined for structured grids
  - If not defined a surface is created with a boundary condition of “overlap”

Specifying Boundary Conditions for Unstructured Grids

- Automatically set for VGRID files
  - Internal mapping between USM3D BCs and Suggar++ BCs
- Boundary conditions can be specified
  - In the input XML file
  - In auxiliary 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

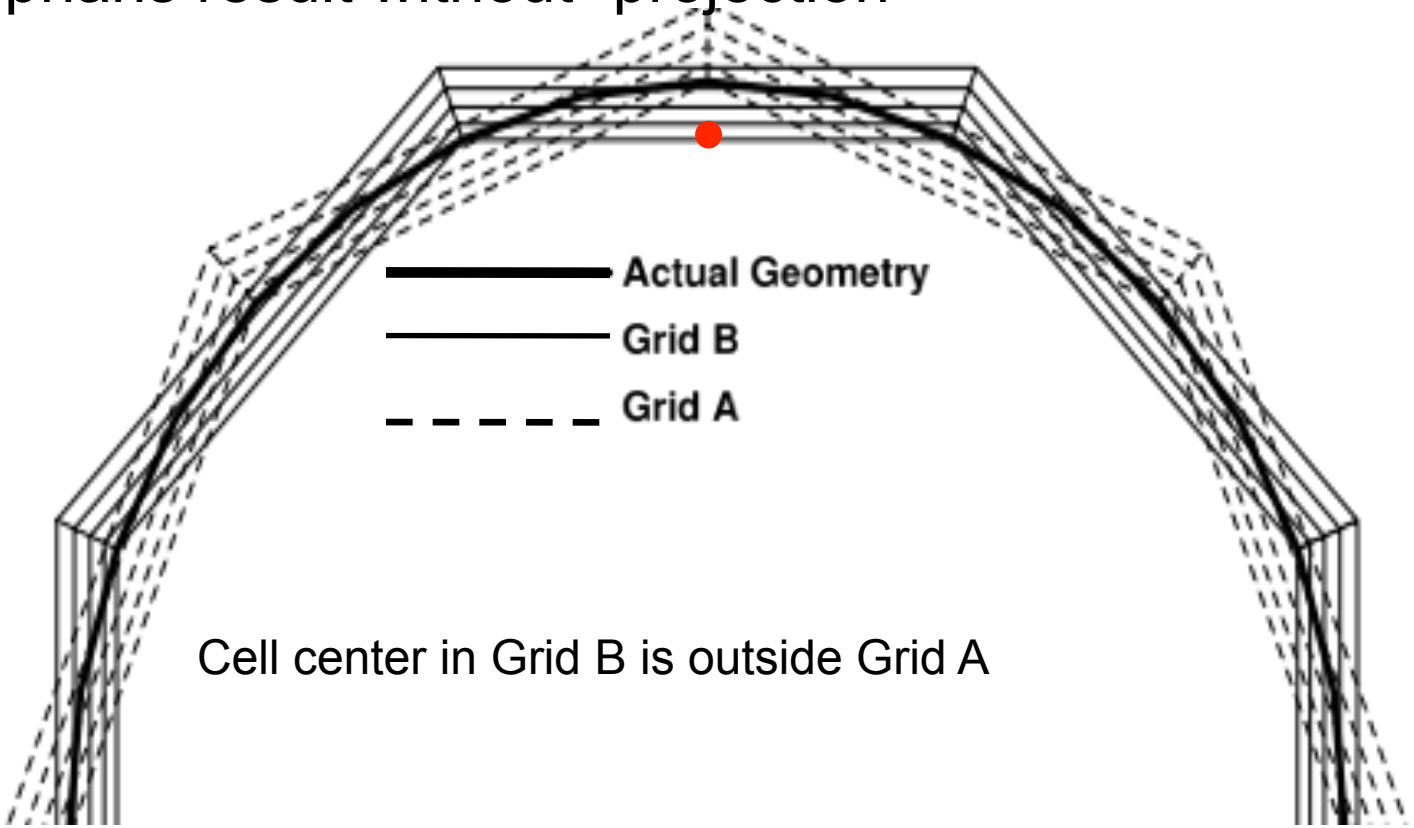
- Overlapping surface grids present several additional complexities
  - Surfaces in a grid can be associated with different geometry components
  - Overlapping surfaces will have different discrete representations
  - Overlapping surfaces require special treatment to eliminate double counting in Force 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 used 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++ performs the surface assembly internally
  - Enabled with <surface_assembly/> element
- Dynamic overlap is now supported
  - Static surfaces are assembled once
  - During motion only perform the assembly between surfaces in different dynamic grids

<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"/>

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 library 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/watertight grid

<usurp> Element

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

```
<global>
   <usurp/>
…
```

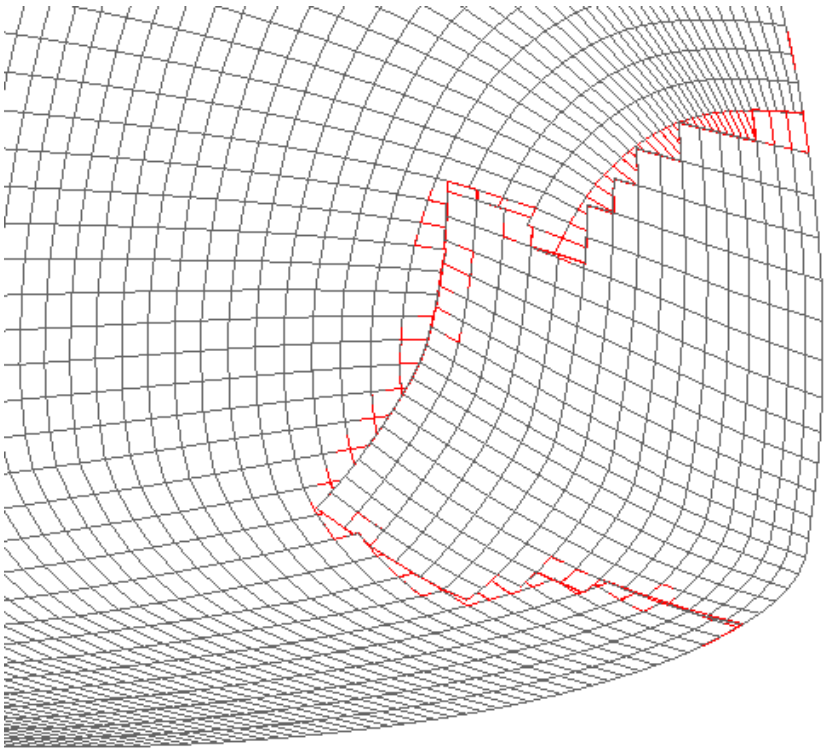
<usurp> Element Output Files

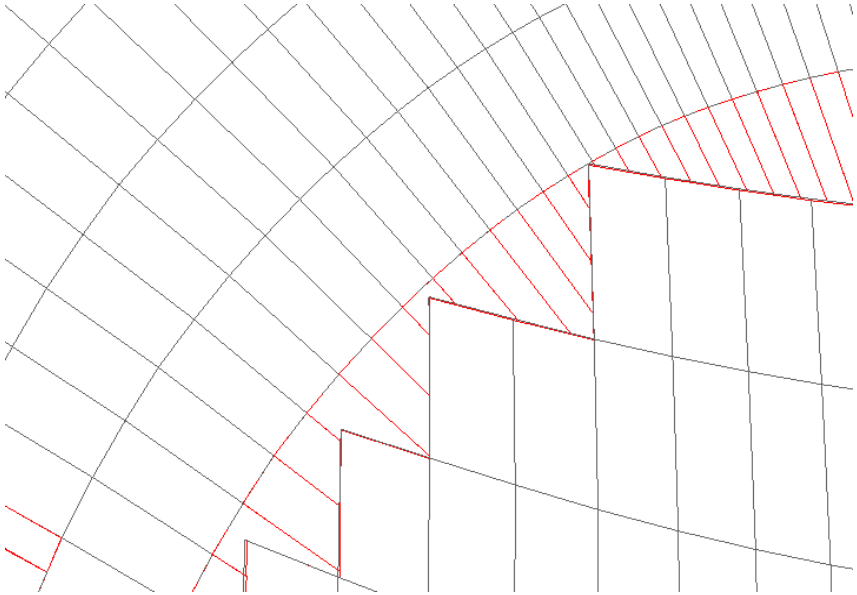
- panels_weights.txt
  - List of panel index, area_ratio, area, ratio*area, is_clipped, number_contours
- Surface panels and triangles
  - Tecplot file: usurp-surfaces.dat
  - Flex file for gviz: usurp-surfaces.flex
- Panels and clipped polygons
  - Flex file for gviz: usurp_panels.flex

<usurp> Element Output Files

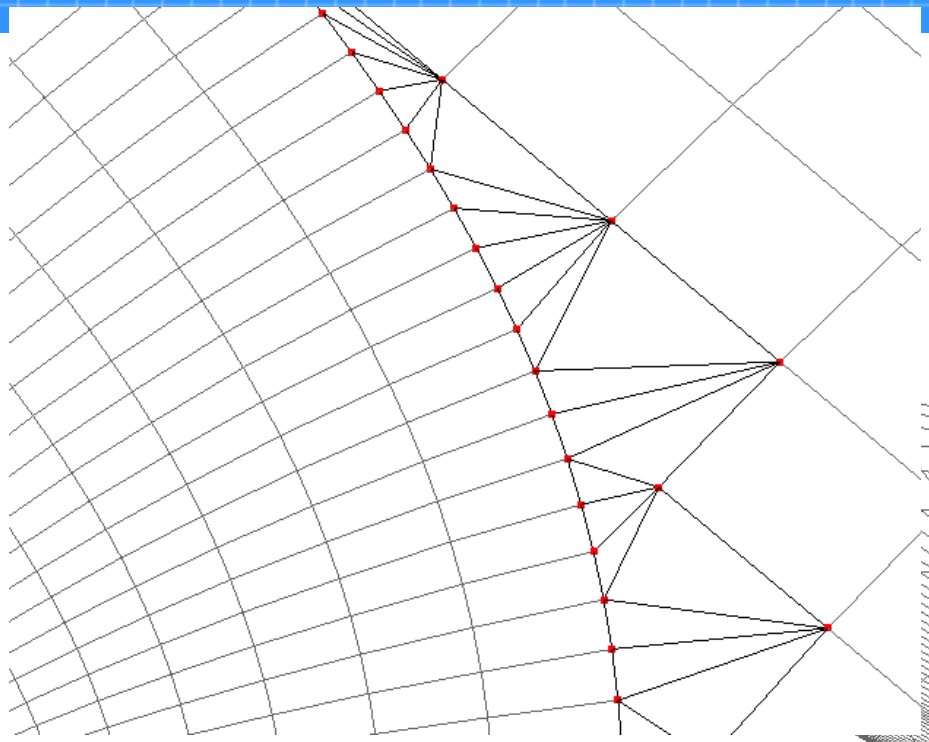
- If ***create_watertight_surfaces***="yes"
- Zipper grid:
  - Quads and zipper triangles
    - water_tight_surface_faces.flex
    - water_tight_surface_faces_dg_*.flex
  - Zipper triangles with quads replaced by triangles
    - water_tight_surface_faces_all_tris.flex
    - water_tight_surface_faces_all_tris_dg_*.flex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

Suggar++ USURP output for Robin Helicopter Fuselage

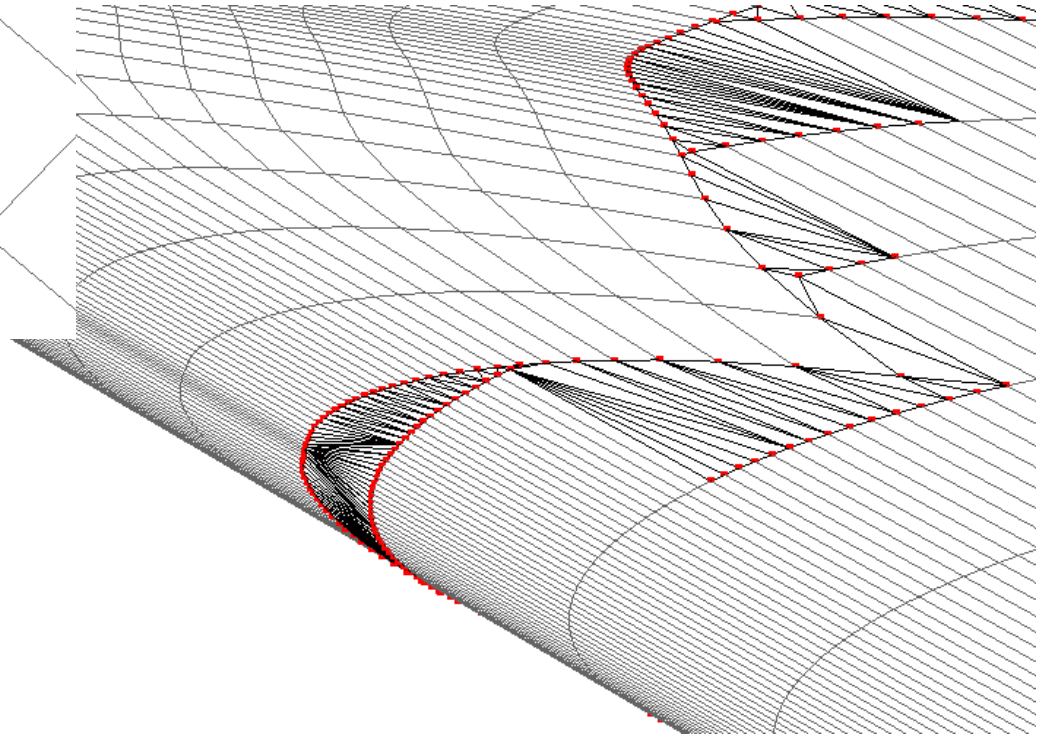


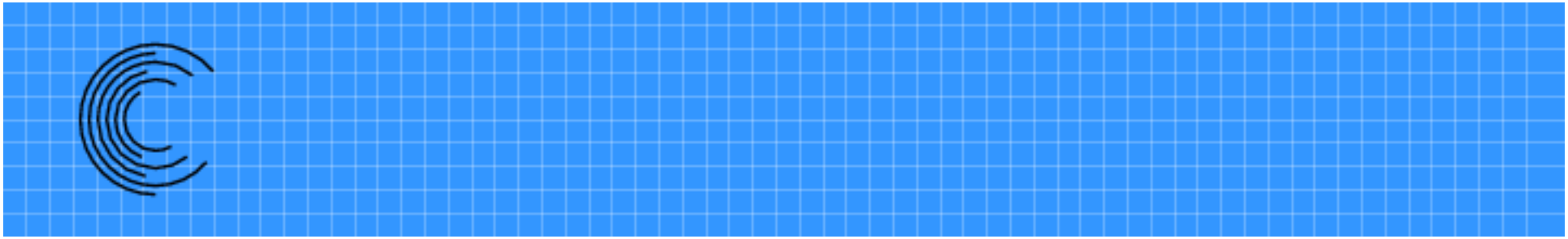


Suggar++ USURP output for WingBody: Watertight Surface Grid



Zipper grid:
Triangles contain only points in the original grid





Suggar++ and Pointwise

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>

Questions

Commercial distribution and support
for Suggar++provided by

Celeritas Simulation Technology, LLC

http://www.CeleritasSimTech.com

Exportable under an EAR-99 license