

PENNSSTATE



ARL

Applied Research Laboratory
The Pennsylvania State University

Current Status of Suggar++

Ralph Noack

David Boger

Penn State University Applied Research Laboratory

9th Symposium on Overset Composite Grid and Solution
Technology

State College, PA

14-16 October 2008

■ SUGGAR++

- Hole cutting
- Donor Search
- Parallel Execution
- DCI for Immersed Boundary in Solver
- Current Status

■ Summary

- **Object oriented design written in C++**
- **Goal is to have same broad capabilities as SUGGAR**
 - **Node and/or cell centered**
 - **Many different grid topologies**
 - **Suitable for moving body simulations**
- **Reusing some low level code when appropriate**
- **Add new capabilities and improved performance**

- **Explicit Cut:** user specifies these points are out
- **Query Cut:** Is this point inside a body?
 - **Examples:**
 - Object X-Rays
 - Analytic shapes/DCF3D
 - Ray casting: count intersections with body
 - SUGGAR Octree: Cartesian approximation to geometry
 - “Thin cut” problem: geometry is sub-grid feature
- **Direct Cut:** Is element intersected by geometry face?
 - **Beggar and Overture approach:**
 - Find element containing facet nodes
 - Refine facet until nodes are in neighboring elements
 - ◆ Can lead to excessive refinement
 - Requires a water tight set of surface faces
 - Flood/fill operation to mark interior/exterior
 - No “Thin cut” problem
- **Implicit Hole Cut:** What is not needed by solver?
 - Pundit approach

■ Octree in Suggar

- Simple and fast in use
- Multi-resolution
 - Can require excessive amount of memory in small gap regions
 - Often requires user input to cut a tighter hole
- Difficult for user to visualize geometry

■ Direct Cut

- **More complex, slower**
- **More accurate (tighter holes) than approximate approaches**
 - Very important in small gap regions
 - Eliminates “thin cut” problem
- **Easier for user to visualize geometry**
- **Less user input: should need just flow solver BC's**
- **Hopefully**
 - More reliable
 - Less user support

■ Cutting begins with Donor search

- Find elements in grid to be cut containing corners of cutting face
- If corners are not inside the grid
 - Use boundary element ADT to check for intersection with elements on boundary of grid

■ Cut element

- Cut faces of element
- Intersect faces of element by cutting face
 - Edges of element face intersect cutting face
 - Edges of cutting face intersect element face

■ Cut element Neighbors

- End recursion when neighbor was already cut by the face

- **Body hierarchy to determine the geometry that will cut a grid**
 - Same as in Suggar
- **“Unified cutter” also available**
 - Not using body hierarchy to control hole cutting
 - Simplifies input
 - Hierarchy required only for motions
 - Find cutting facets that overlap
 - Parent grids are protected from cutting by the *individual* facets
 - Inspired by USURP/POLYMIXSUR for F&M integration

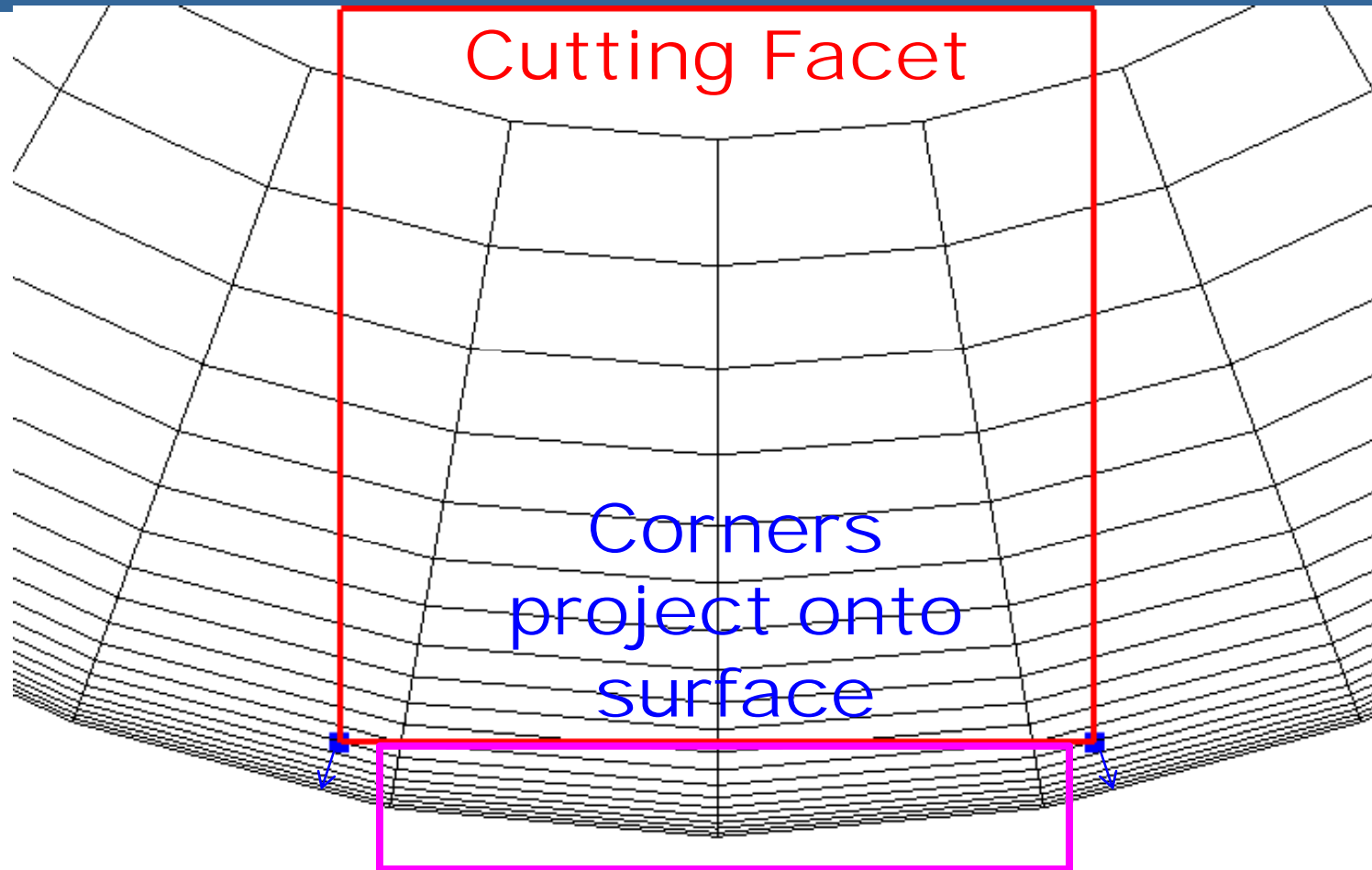
- **Floating point tolerances in intersection tests**
 - Allow fuzzy cut: intersection slightly outside of bounds
- **Quad faces:**
 - Non-planar
 - Treat as plane with finite thickness
 - Break into 2 triangles
- **Viscous layer cutting**
 - Large face perpendicular to surface
 - 2 Nodes “project” onto surface in surface assembly
 - Will leak without viscous layer/extended cutter
 - Need to implement procedure to extend cut to the wall

ARL

Penn State

COMPUTATIONAL MECHANICS

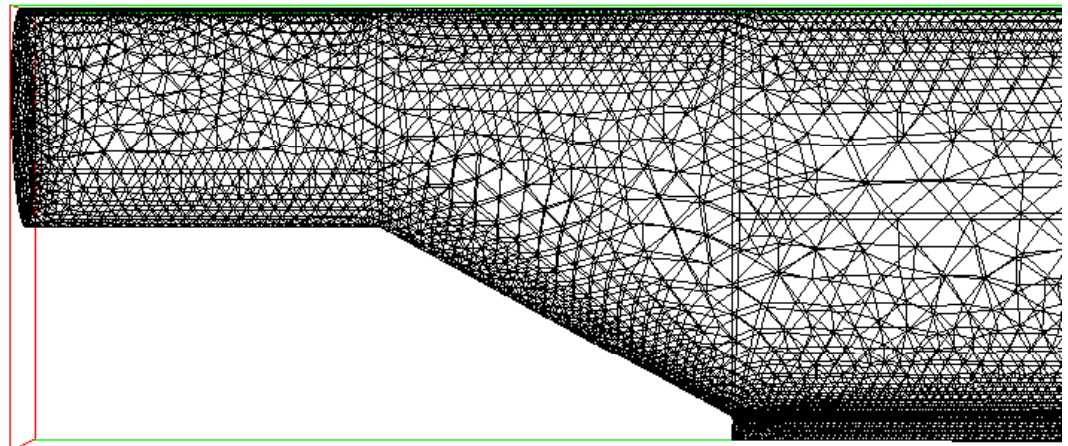
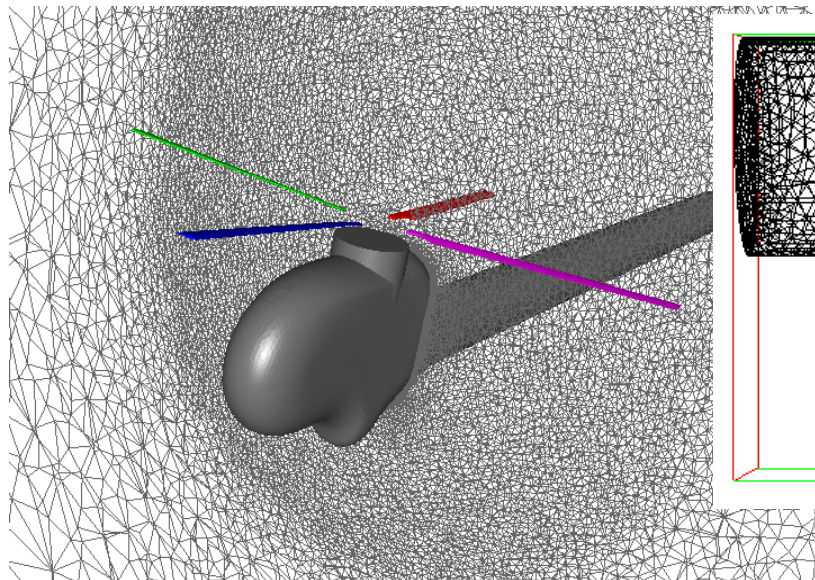
Viscous Layer Cutting



Perpendicular edges need to be cut

- **Reducing the number of faces in the cutting surface will reduce hole-cut CPU time**
 - **Rotorcraft grid system has 33k tri faces on each blade surface**
 - **Replace blade by bounding box <cutter_grid> with 6 quad faces = 12 tri faces**

<cutter_grid> for Rotor Blade



- **Plan to implement <replacement_cutter>**
 - **Any cutter face falling inside the replacement cutter is ignored**
 - **Surfaces of replacement cutter will be used as cutter**
 - **Coarse approximation in region of replacement cutter, original cutter when it protrudes from replacement cutter**

■ FringeSet contains set of fringes in one grid

- Sent to other appropriate ranks so they can search their grids for donors
- Results are returned to original fringe grid rank

■ Donor search

- Neighbor walk/Stencil jump used to find a donor
- Search initiated from:
 - Hint based on my donor from previous time step
 - Hint based on my "sponsor's" donor from this time step
 - ◆ For min fringe only
 - Hints based on donors found by my neighbors within my fringe set
 - Hint found using Inverse Map

■ Donor search tries to chain searches

- Attempt to always have donor from neighbor as hint to start search
- Inner/Outer fringe donor search:
 - Neighboring fringes are put on a stack along with donor
 - Stack is used for next fringe
 - ◆ When stack is empty go back to FringeSet for next fringe without donor
- Overlap Minimization:
 - Starts from Inner/Outer fringes
 - If fringe is min fringe (donor is smaller than fringe) put neighbors into new fringe set along with hint from sponsor

■ Often have 100% efficiency: all searches start from neighbor donor

- **Overlapping surface grids with viscous spacing require “projection” of one surface onto the other**
- **Suggar uses SURFASM utility to obtain deviation between surfaces**
- **Suggar++ performs surface assembly internally**
 - **Overlapping faces are also used in “Unified cutter”**

- **Parallel decomposition is by grid**
 - Grid is assigned to a rank
 - Heavy data (grid points, connectivity) stored on only that rank
 - Light data (includes surface/cutter data) is stored on all ranks
- ***Par_apply_to_grid(func)* is utility routine that will call `grid->func()` for the appropriate list of grids**
 - Execute in parallel using threads and/or MPI
 - Allows an MPI rank to use threads to parallel process its set of grids
 - Used whenever possible: speed up initialization in addition to computing DCI

	Rotor1 Coarse			Rotor1 Medium			Rotor2			
	Suggar		Suggar++	Suggar		Suggar++	Suggar		Suggar++	
		Full	Box	With Reuse	Full	Box	No Reuse	With Reuse	Full	Box
Hole cut	2.6	4.9	0.4	4.1	7.2		2.8	2.8	8.7	0.2
Donor search	0.4	0.7	0.7	1.7	2.8		56.5	3.4	4.8	3.3
Minimization	6.2	0.3	0.2	23.4	2.4		77.0	38.0	33.0	25.2
Assembly Step	15.5	7.9	3.0	39.2	16.5		150.0	57.0	52.0	33.3

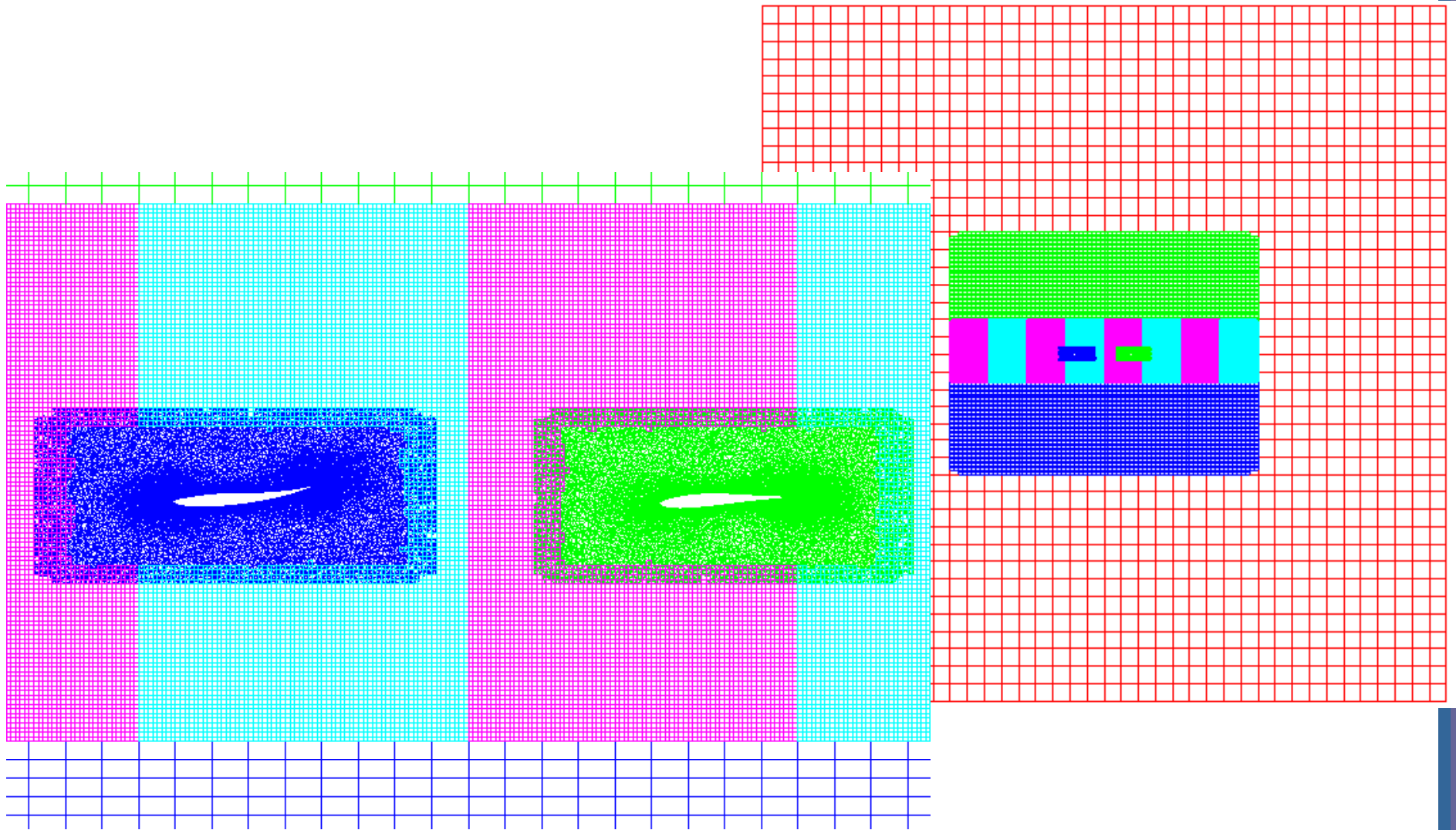
Wall clock time in seconds/step

Severely imbalanced

Machine was loaded

Timings eyeball averaged over several time steps

Rotor System + Cartesian Background



	Suggar	Suggar++		
	Serial	Serial	2 threads	4 threads
	With Reuse	Full	Full	Full
Hole cut	32	12	7	4.8
Donor search	7	19	13	7.5
Minimization	74	50	30	20.0
Assembly Step	140	150	110	90.0

Wall clock time in
seconds/step

I/O time for Suggar++ dci file ~ 56 seconds

Total 40 million points, 62 million elements

Background grid: 35567215 nodes, 34697728
elements

41x41x41, 65x73x47, 65x73x44, (513x73x117)*8

3 Blade grids: Each 1544344 nodes, 9072229
tet elements

Machine was loaded

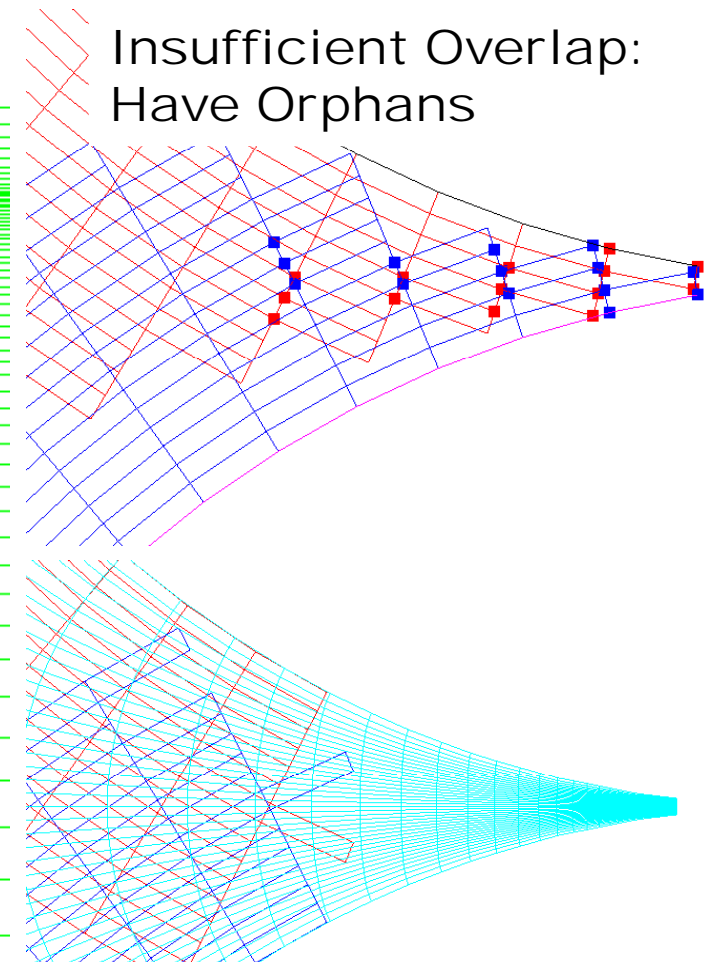
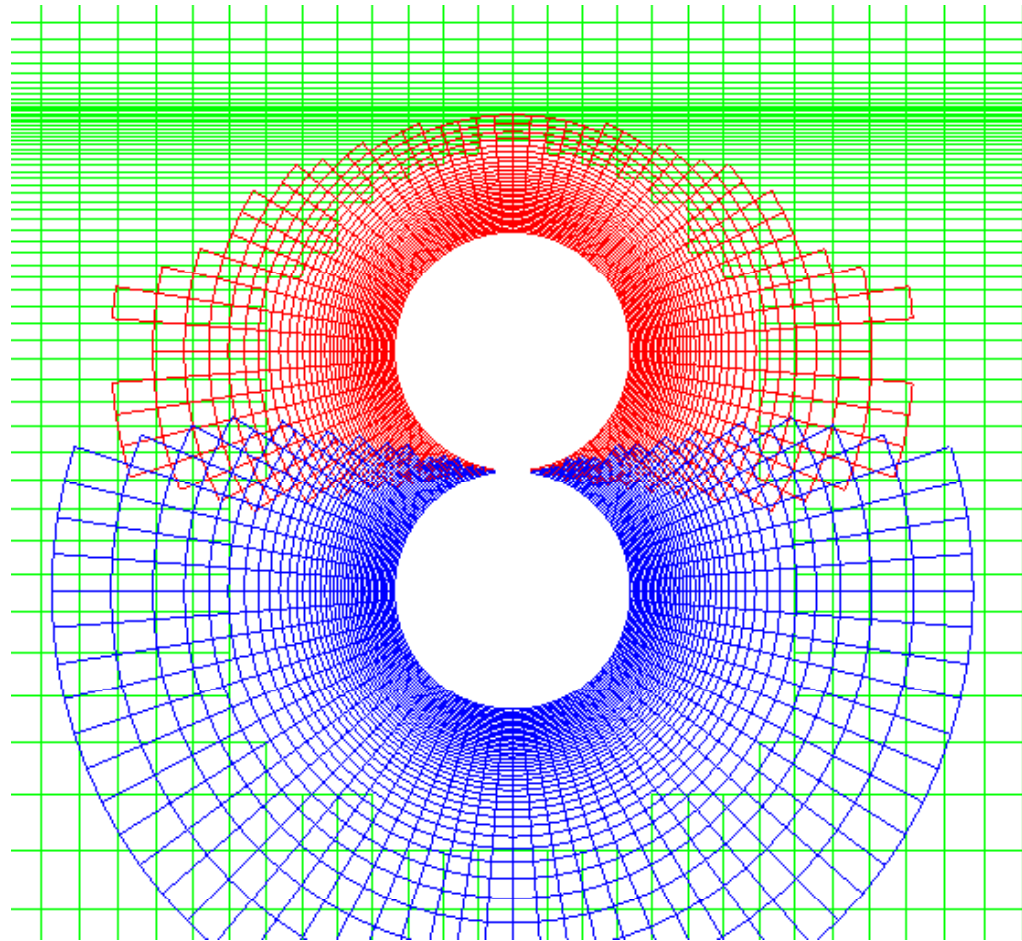
Timings eyeball averaged over several time steps

- **Grids of bodies in close proximity may not have sufficient overlap**
 - Results in orphans
- **Need to add a collar grid that conforms to both bodies: Not cut by either body**
 - Difficult for moving bodies
- **Typical approach is to leave a small gap with both grid adequately refined**
 - Can result in significant increase in grid points/decrease in element size in region of low interest
 - May be very difficult to grid properly

- **Immersed Boundary approach imposes a “solid boundary condition” at locations other than a grid block boundary**
- **New capability: Suggar identifies locations to be solved as immersed boundary in flow solver to eliminate orphans due to insufficient overlap**
- **Imposes local “Stair Stepped” approximation to the geometry**

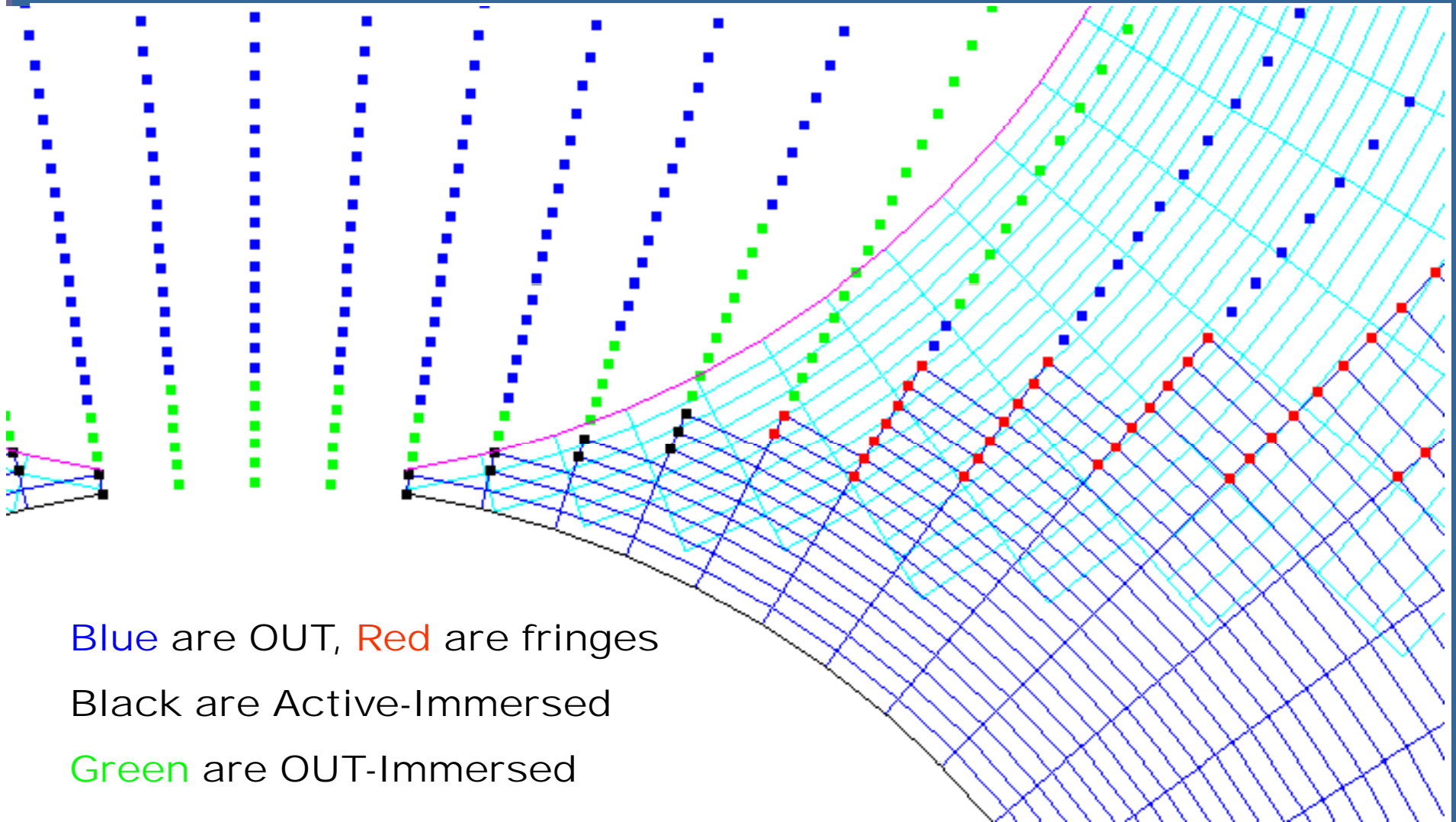
- **Mark body as “Immersed Boundary Cutter”**
 - **Hole cutting marks**
 - Grid edges as cut by immersed boundary cutter
 - OUT points as inside immersed boundary cutter body
 - Other point as adjacent to immersed boundary
- **Mark fringes adjacent to holes (InnerFringe1) and search for donors**
 - If no donor change to Active-Immersed
- **Mark second level fringe (InnerFringe2) and search for donors**
 - If no donor and next to InnerFringe1 adjacent to immersed boundary
 - Change InnerFringe1 to Active-Immersed
 - Change InnerFringe2 to Active

Example: Two Cylinders in Contact



Collar Grid To Eliminate Orphans
Cannot be used for relative motion

Example: Two Cylinders in Contact



Blue are OUT, Red are fringes

Black are Active-Immersed

Green are OUT-Immersed

■ **Structured Grids**

- **Cartesian: Uniform and Stretched**
- **Curvilinear**

■ **Unstructured:**

- **Tetrahedron**
 - VGRID grid file
- **Mixed element: tet, hex, prism, pyramid**
 - Fieldview unstructured grid file

■ **Other grid types will be added as interest/funding dictates**

- **New hole cut approach: better for small gaps, more intuitive**
- **Will write XINTOUT output file**
 - Allows preliminary testing by CFDShip for static problems
- **Projection using surface assembly**
 - Surface assembly is integral/internal to Suggar++
- **Parallel**
 - Threads and/or MPI
 - MPI rank may have a set of grids, which can be threaded
 - Domain decomposition to distribute memory for MPI
 - MPI execution can have each rank also run multiple threads
 - Suggar++ -partition_mesh
 - Currently creates simple splitting and writes decomposed grids
 - Need to write input file for split grids
- **LibSuggar++ API is partially implemented**
 - Soon be able to use as replacement for libSuggar
- **Alpha status-Need to implement**
 - Viscous layer cutting
 - Decompose grids

- **SUGGAR while useful has several weaknesses**
- **SUGGAR++ is a rewrite in C++**
 - Intend to have same basic capabilities SUGGAR
 - Object oriented design and implementation should assist development and maintenance
- **Direct cut instead of octree approximation**
 - Still working on hole cut robustness
- **Preliminary performance is encouraging**
 - Need to decompose grids for load balancing
- **Investigating immersed boundary procedure for insufficient overlap**