OSCAR - An Overset Grid Assembler for Overlapping Strand/CARtesian Mesh Systems

Jay Sitaraman **Beatrice Roget**
University of Wyoming
Andrew M. Wissink
U.S Army Aeroflightdynamics directorate

Outline

- Motivation
- Problem definition and grid topologies
- Strand/Cartesian overset grid assembly
  - Preprocessing
  - Search
  - Communication
  - Application examples
- Gridding issues at sharp corners
  - Removal of warped volumes
  - Clipping strategies to clear self-intersections
- Ongoing and future work

Motivation for Strand/Cartesian framework

- Body-conforming volume grids take time to generate and require hands-on expertise for complex geometry
  - In descending order of difficult (1)Structured multi-block, (2)structured overset, (3)unstructured

- Parallel Overset Grid Assembly becomes imbalanced for any type of partitioned grid systems – rebalancing can improve this, but at the expense of larger communication overhead

- Compact data structures for grids – maintainable concisely on each process can remarkably improve efficiency

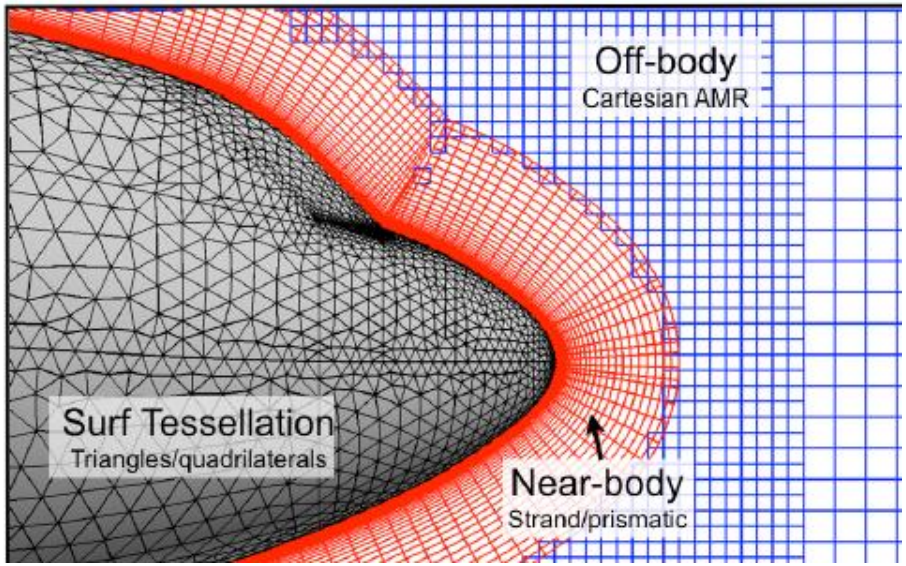
- Software design of the future should support the most efficient work flow from CAD surface to aerodynamic loading metrics (thrust of CREATE A/V)

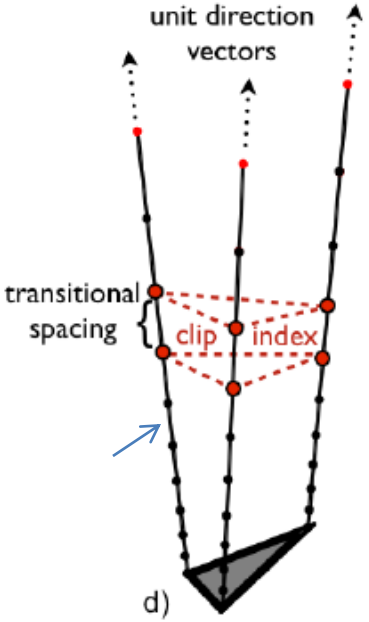
Motivation for dedicated Overset Grid Assembler (OSCAR)

- Strand grids are semi-structured – have to use efficient search strategies that can take advantage of this
- Compact representation of global mesh data known in each process. Have to take advantage of this.
- Data flow is different from traditional approach :
  - every process searches for donor cells suitable for its query points in other processes (automatically scalable)
  - In traditional approach, each process searches for donor cells in its mesh-data for query points from other processes

Meakin, AIAA (2007), Katz JCP (2011), Wissink, AIAA (2012)

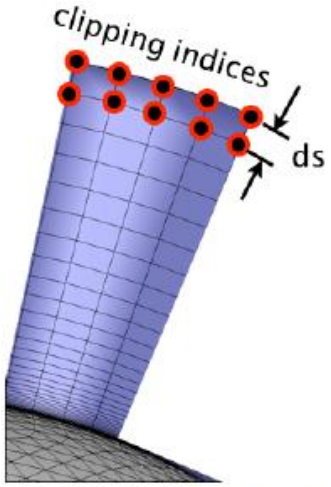
Strand/Cartesian mesh system



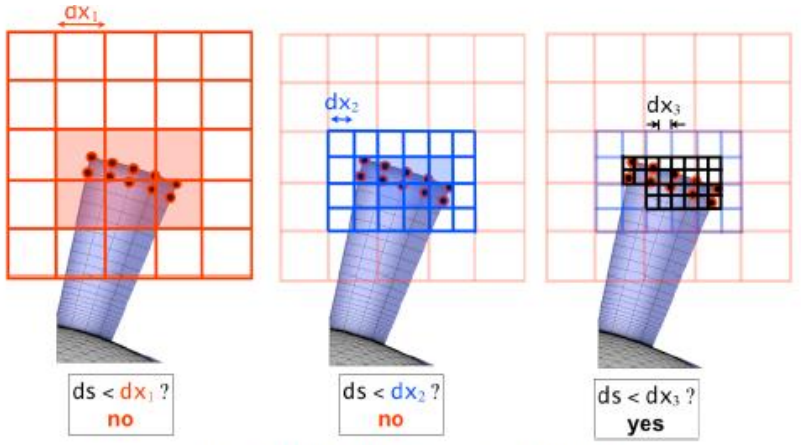


Strand structure – normals for each corner of a strand stack is fixed at root (different from standard hyperbolic prismatic mesh generation)

Off-body mesh generation adapting to strand spacing



(a) Transitional spacing at strand clip points

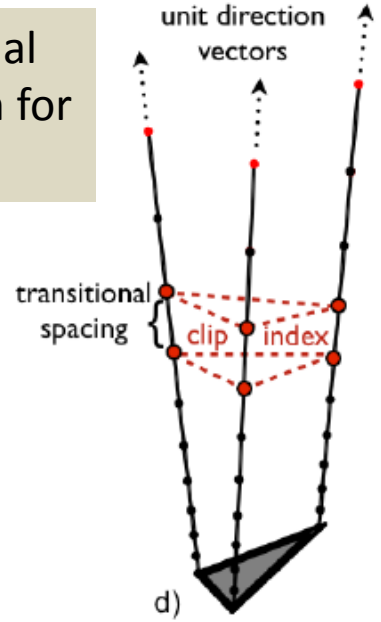


(b) Cartesian refinement to match spacing

Strand and Cartesian Mesh data structures

- Strands

Same normal distribution for all strands

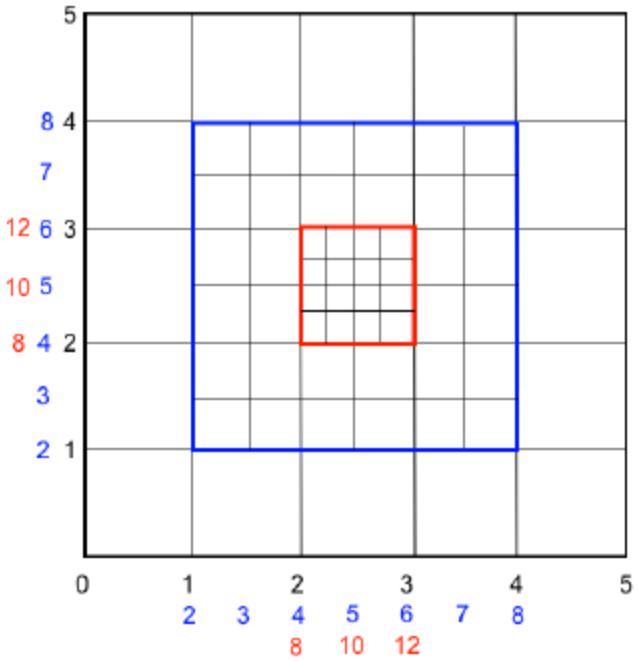


Type	Integers (4 byte)	Floats (8 bytes)
Surface Nodes	0	3*nnodes
Surface connectivity	3*nfaces	
Node normals		3*nnodes
Clipping index	nfaces	
Distance function		nlayers
glob2loc map	2*nfaces	

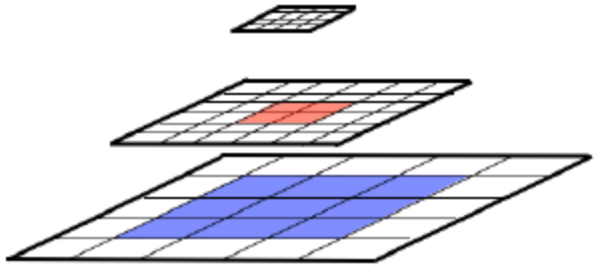
Total memory in Bytes = 24*nfaces + 48*nnodes + 8*nlayers
Only 9.6 MB of storage for mesh with 199,996 faces, 100,000 nodes and 50 layers (total grid cells close to 10 million)

Cartesian mesh system

- Nested block format



Level 0:	global index space ..	(0,0),(5,5)
(coarse)	block ilo/ihi.....	(0,0),(5,5)
Level 1:	global index space ..	(0,0),(10,10)
	block ilo/ihi.....	(2,2),(8,8)
Level 2:	global index space ..	(0,0),(20,20)
(fine)	block ilo/ihi.....	(8,8),(12,12)



Compact storage:

7 ints per block (ilo(3), ihi(3), level)
Global DX (spacing)
Global XLO (lower corner coords)

For 1000 blocks this is only 28kB of data

Problem definition

- An overlapping grid system is composed of Strand and Cartesian mesh types and is to be solved by partitioning into multiple processes

- For each grid node (or cell centroid), the overset grid assembler must identify and designate an unique point type, i.e. it should a field point, receptor point or hole point. For Strands, receptors are specified based on “clipping”

- For each receptor point, it has to also identify the donor cells and the weights

Parallel Strand/Cartesian Overset Grid Assembly (overview)

- Data resident in each process
  - global strand grid and Cartesian grid data
    - Multiple strand grids (one for each body) is allowed
  - Local solution data, i.e. of the few strand blocks and Cartesian blocks assigned to this process

Overset Grid Assembly: overview

- In each process
  1. Tag the query points* above clipping as mandatory receptors
  2. For all query points, search all strand grids (including itself) for possible donors (donors must have better resolution than the query point itself)
  3. If a donor cell is located, notify the process that own the solution data of this donor cell with appropriate information for data interpolation
  4. Generate off-body grids using the list of mandatory receptor points that didn't find donors
  5. Repeat steps 2 and 3, but search domain now includes both the strand and Cartesian grid systems and query points include ones from both Strand and Cartesian grids
  6. Finalize a communication map : each process knows its donor map and receptor map
     - Donor map – where to interpolate from and whom to send this data
     - Receptor map – whom to receive data from and which locally owned data to update

*Points were solution data is stored (could be cell centers or nodes)

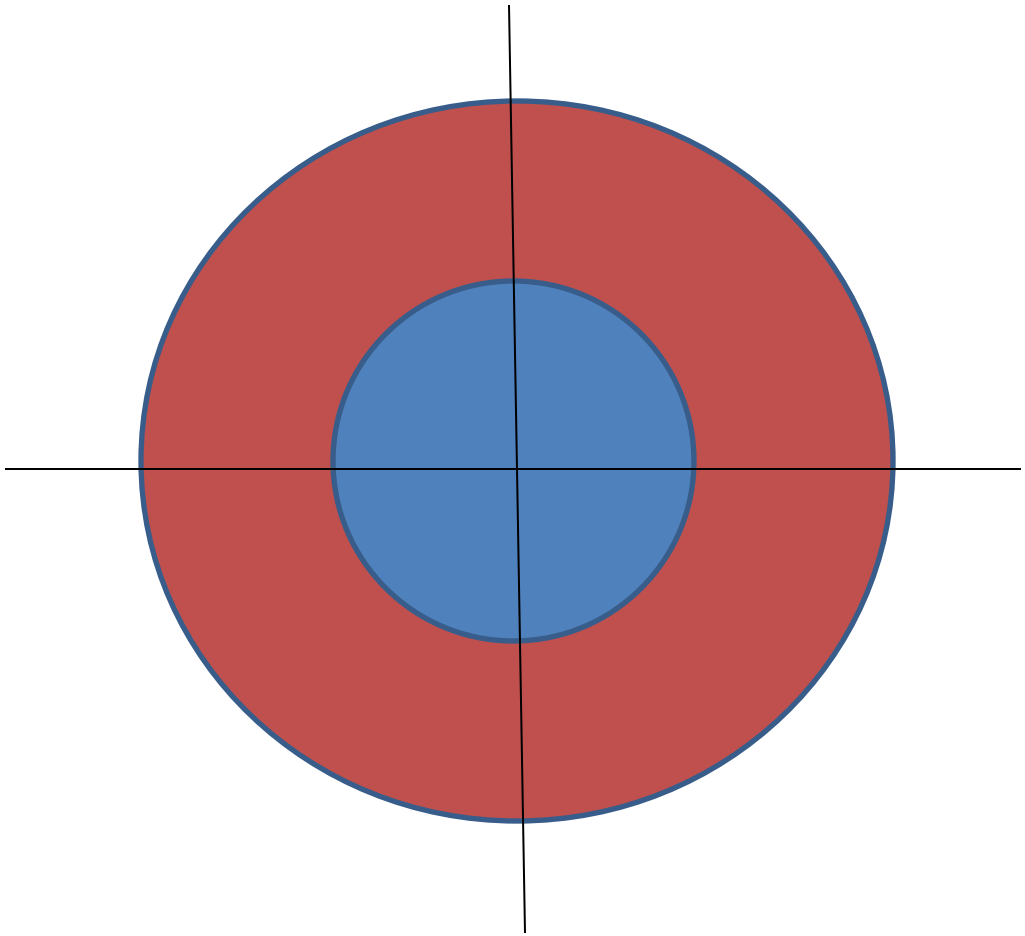
Overset Grid Assembly (Difference from traditional approach)

1. Search donor cells for self-owned query points in all grids
2. Total number of searches are directly proportional to number of query points in each process
3. Communication needs to happen only after the best donor cell has been picked

1. Search donor cells for query points from other partitions
2. Total number of searches are directly proportional to the physical volume of each unstructured partition
3. Communication needs to happen before the best donor has been picked

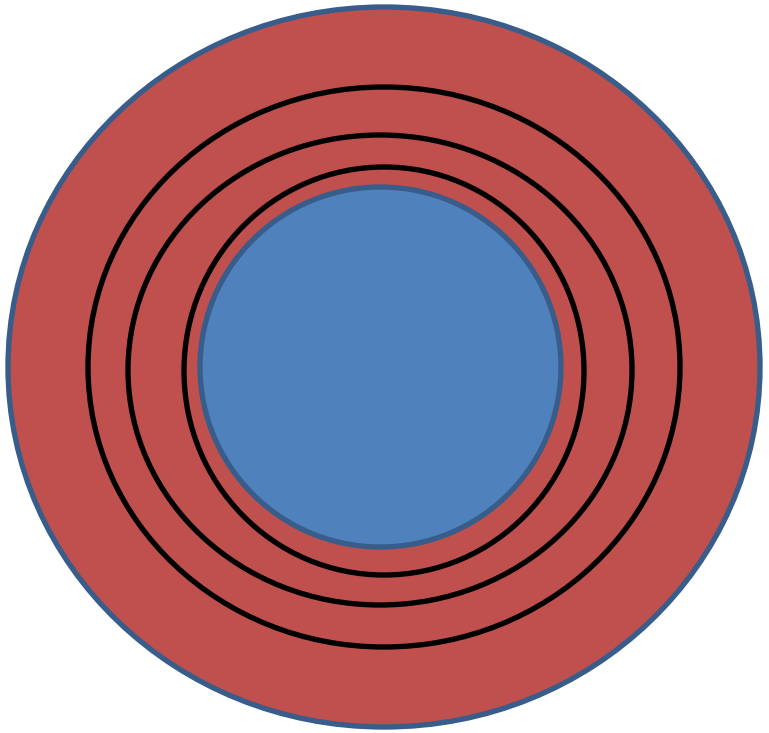
Strand/Cartesian grids are much more scalable since number of query points per process can be equalized easily with partitioning. In traditional case, variance between physical volumes has to be minimized as a constraint to improve load-balance

Scalability issues



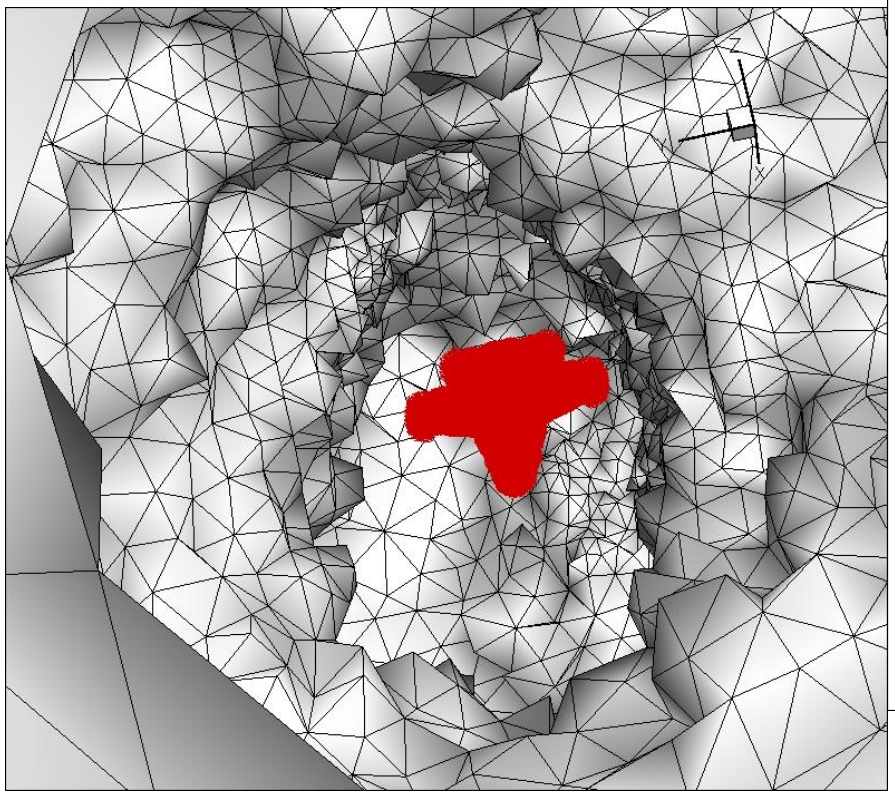
More Balanced for domain connectivity

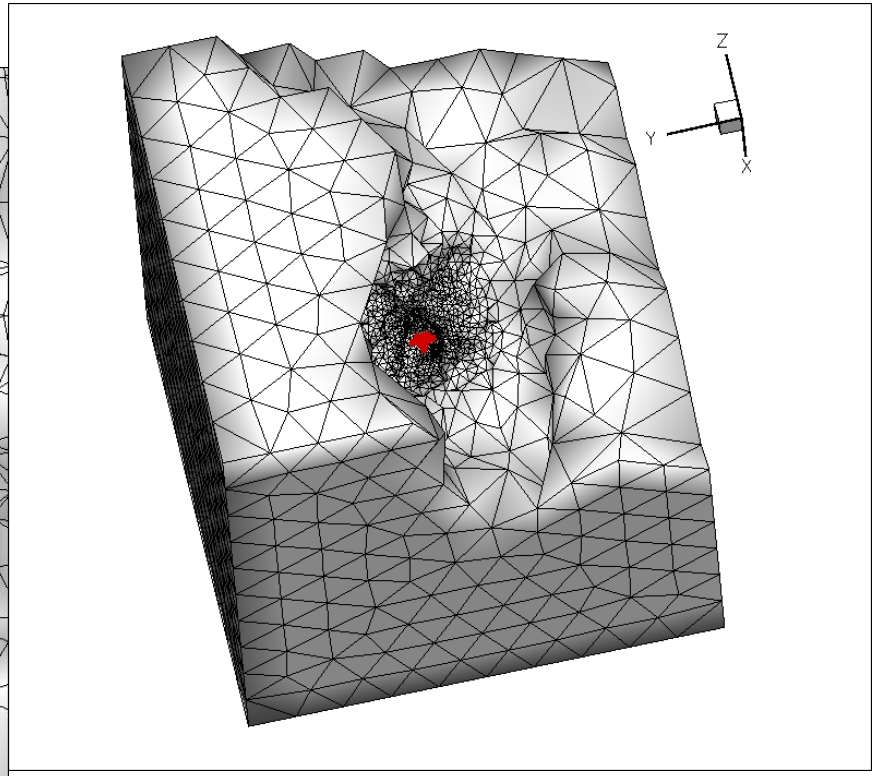
Strands are always partitioned this way since the surface grid is partitioned



Imbalanced for domain connectivity

Scalability issues

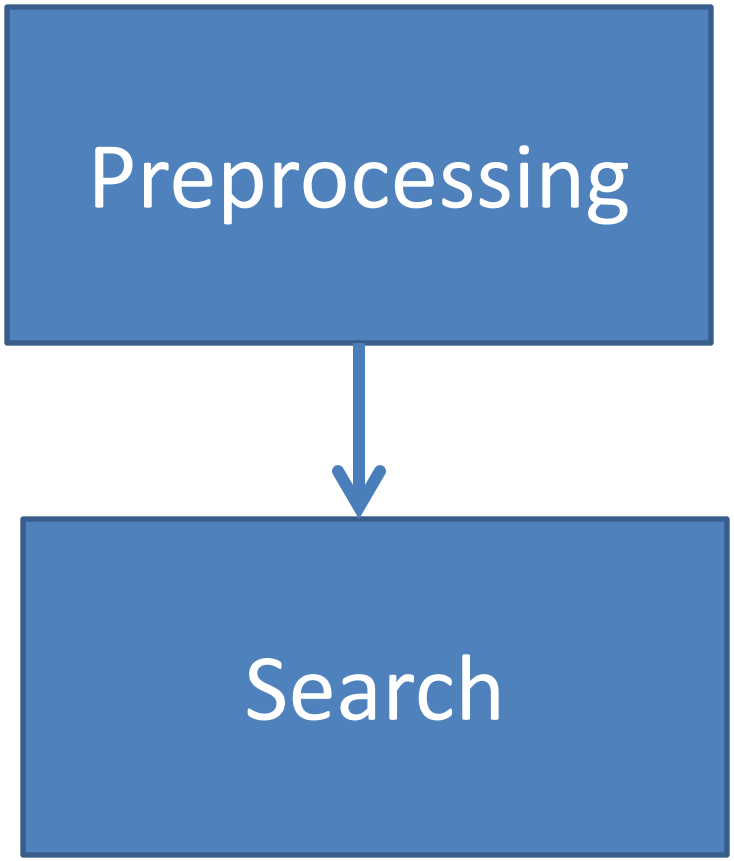




Traditional non-weighted partitioning of the nodal graph

- all partitions have equal number of cells but very different volumes
- donor search will be imbalanced

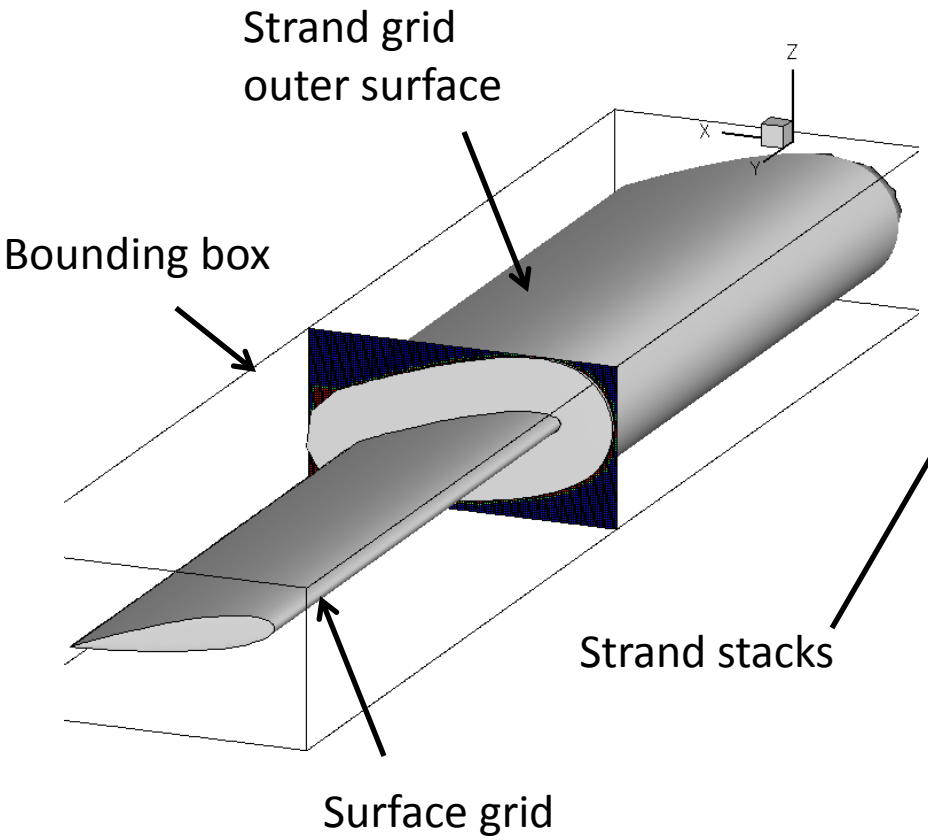
Strand grid search process

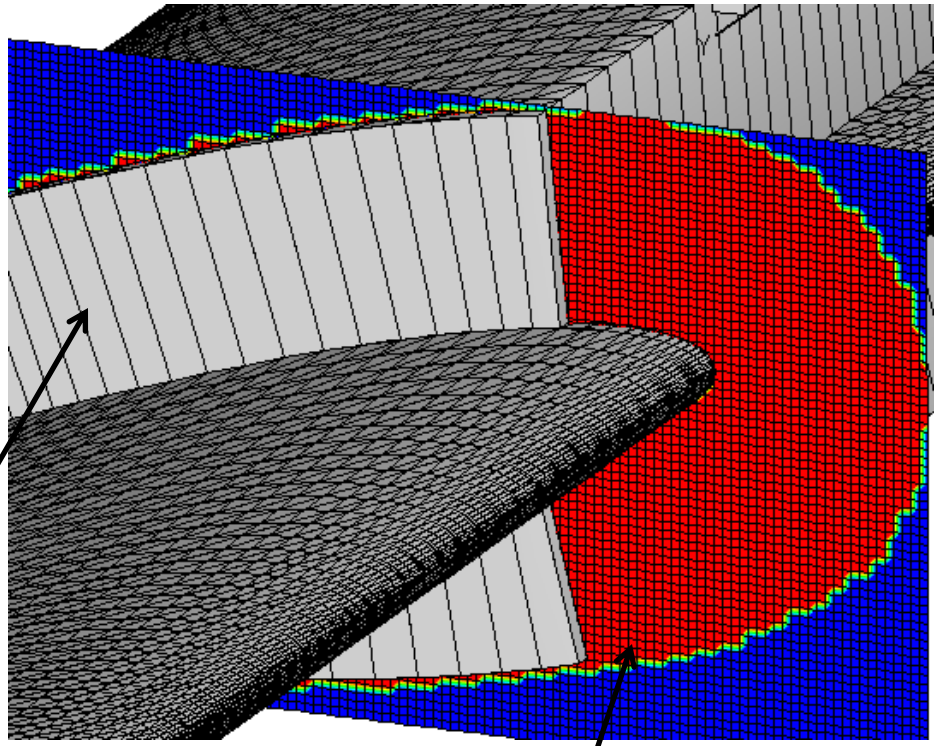


- Create an auxiliary grid by dividing the bounding box of each strand grid
- Make an inverse map that each auxiliary grid cell to a “strand super cell” or “strand stack”

- Find a starting guess using inverse map
- Use line-search to walk into the right stack
- Use another map in the strand direction to identify the exact containing cell

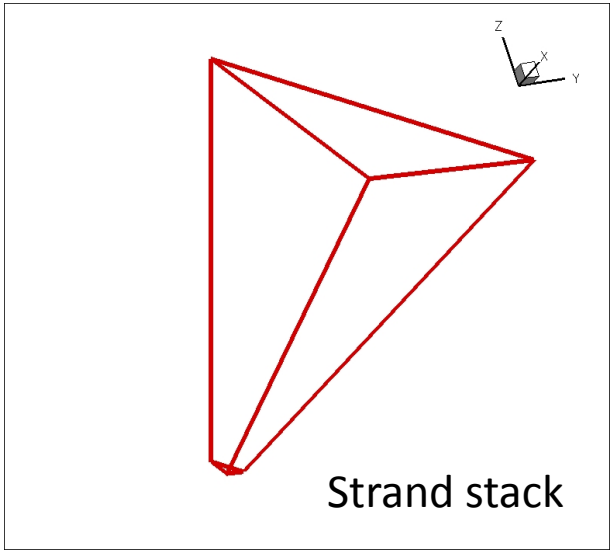
Preprocessing (1)

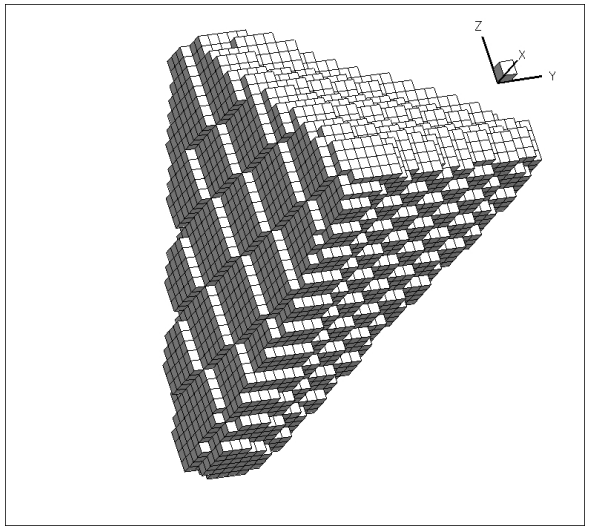




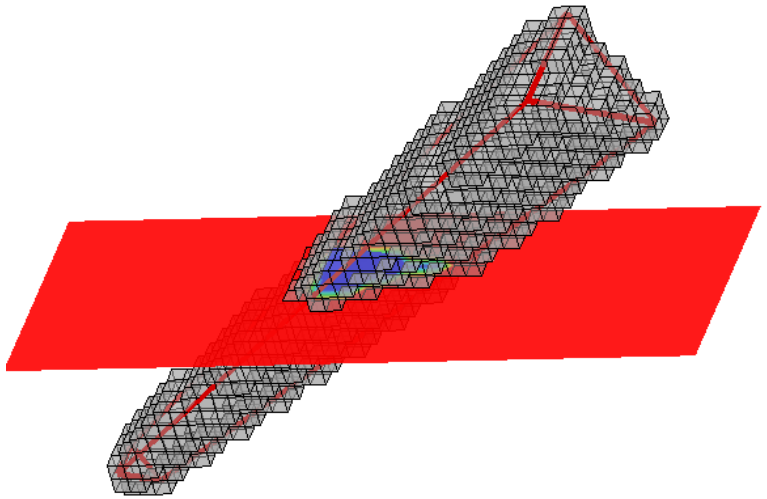
Inverse map to assist searching :
boxes marked red can potentially
contain strand stacks

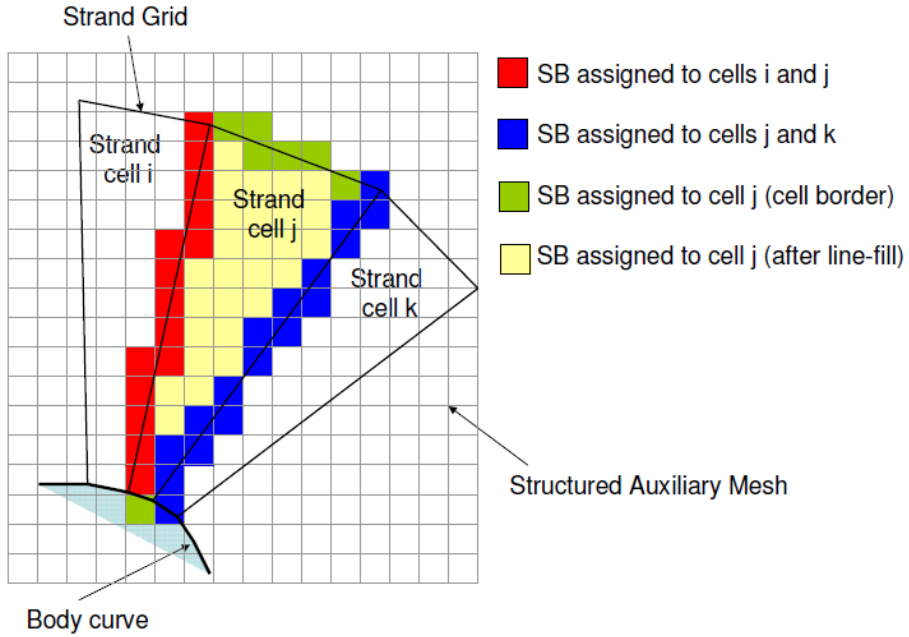
Preprocessing(2)



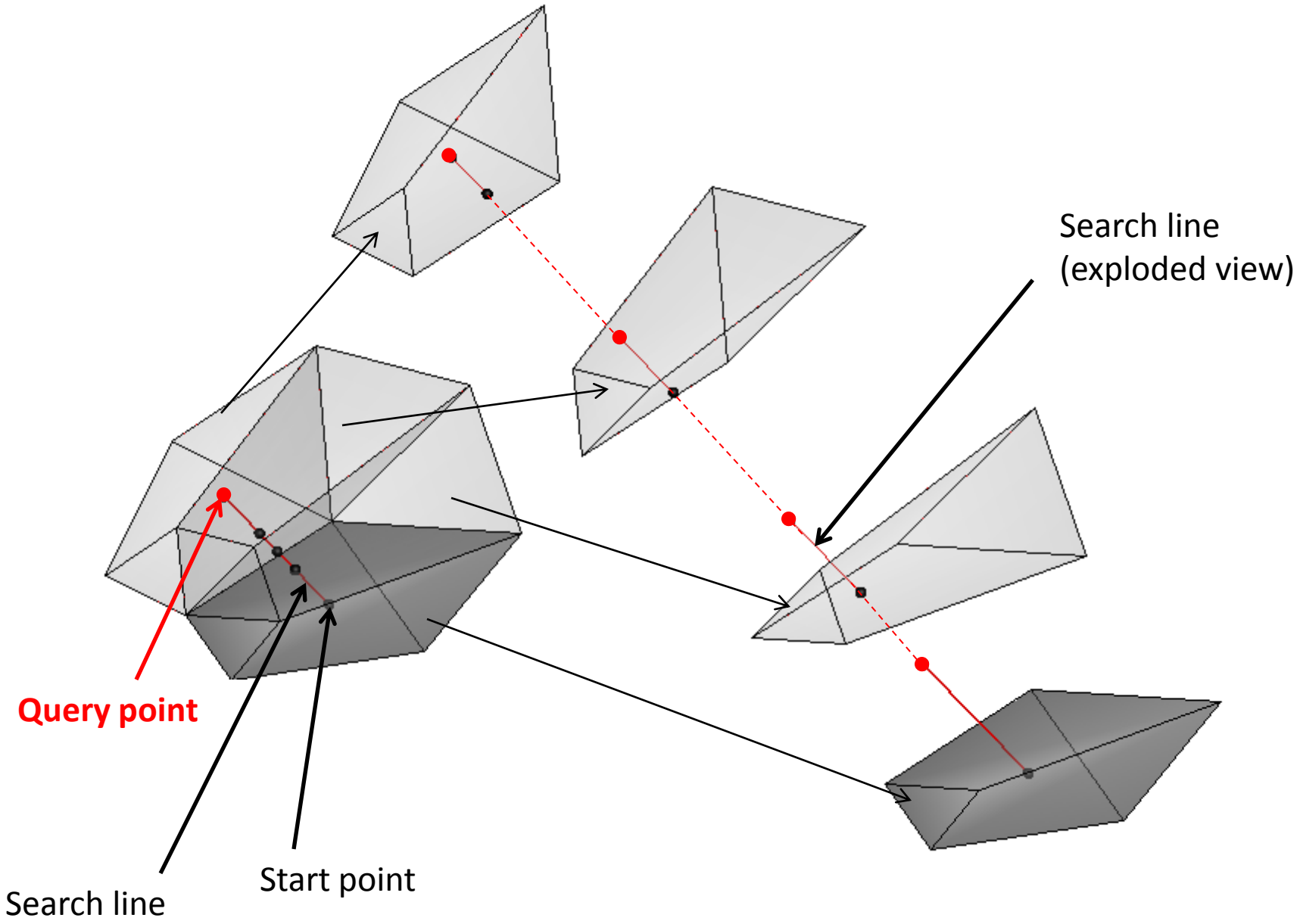


Inverse-map by rasterization

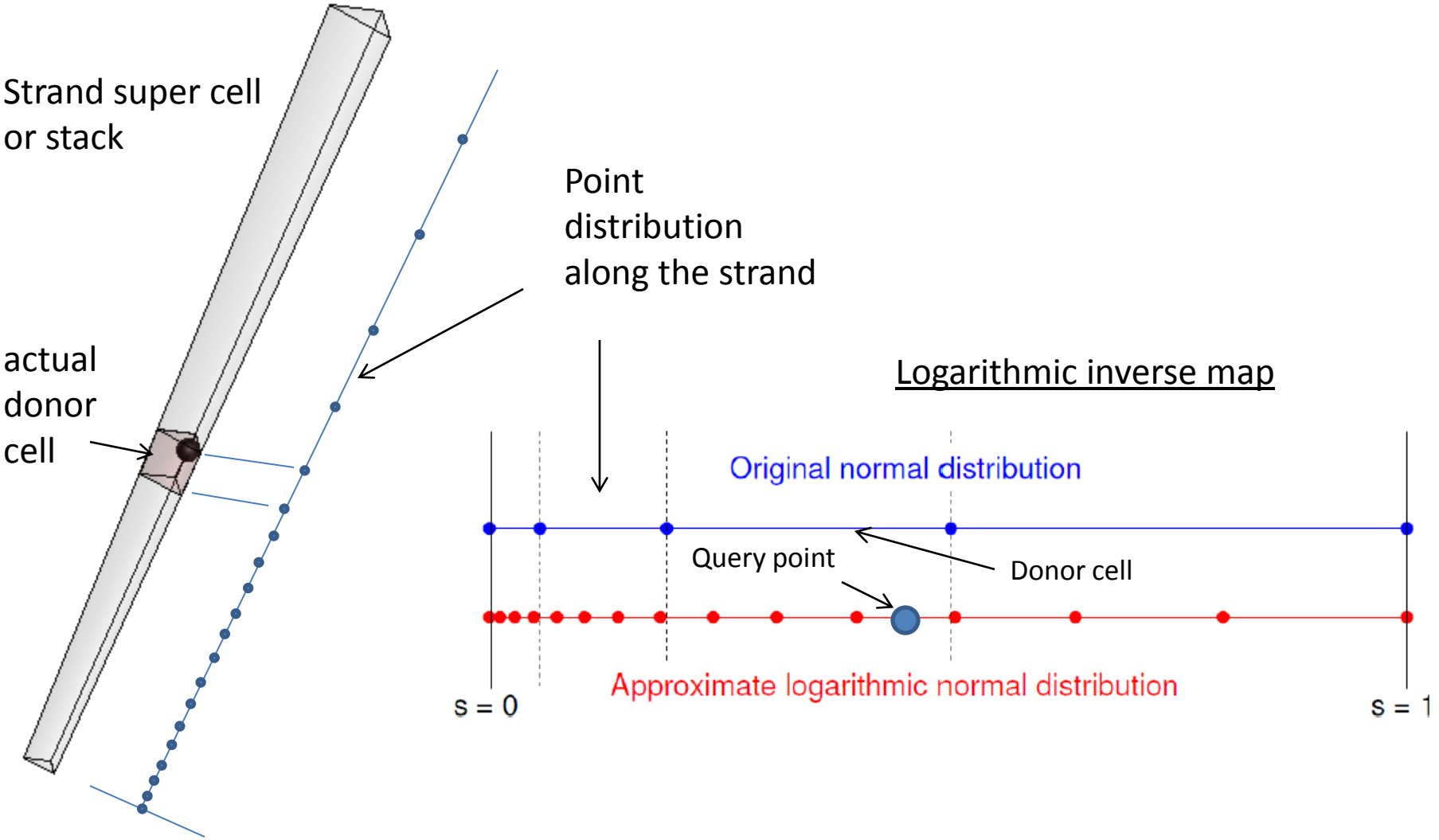




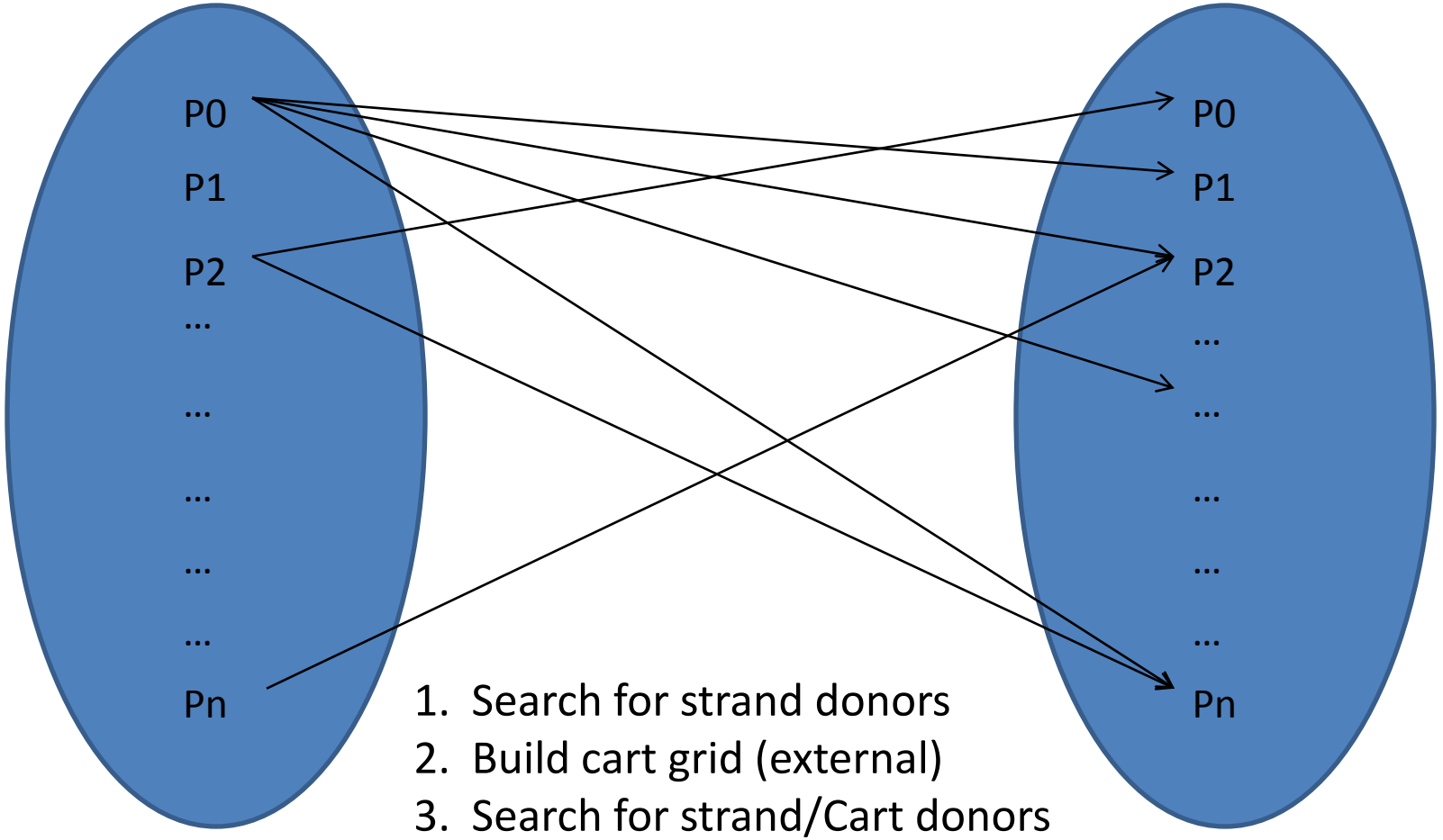
Donor search using line-walk process



Actual donor cell using secondary map

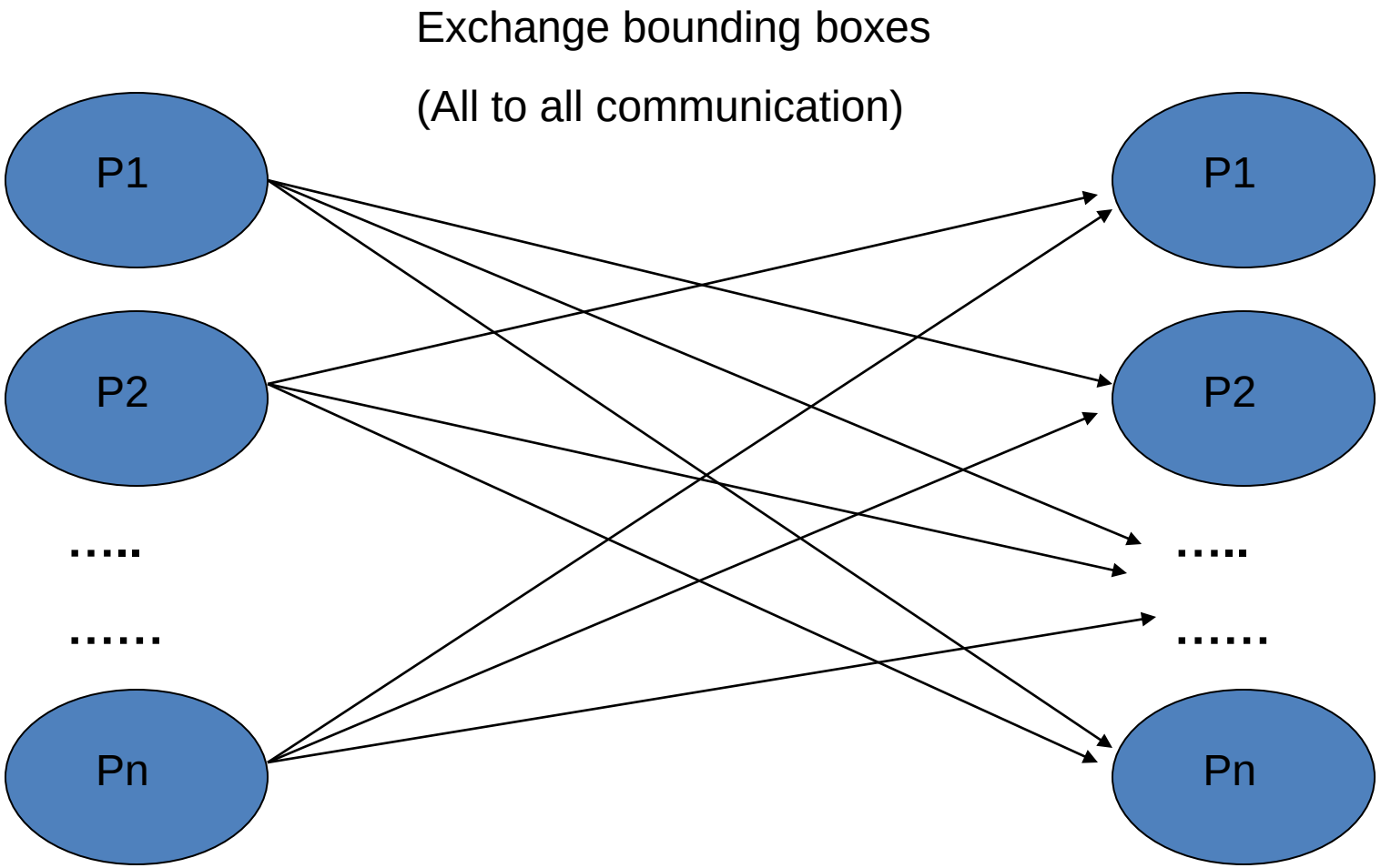


Communication in Strand/Cart overset grid assembly (OSCAR)

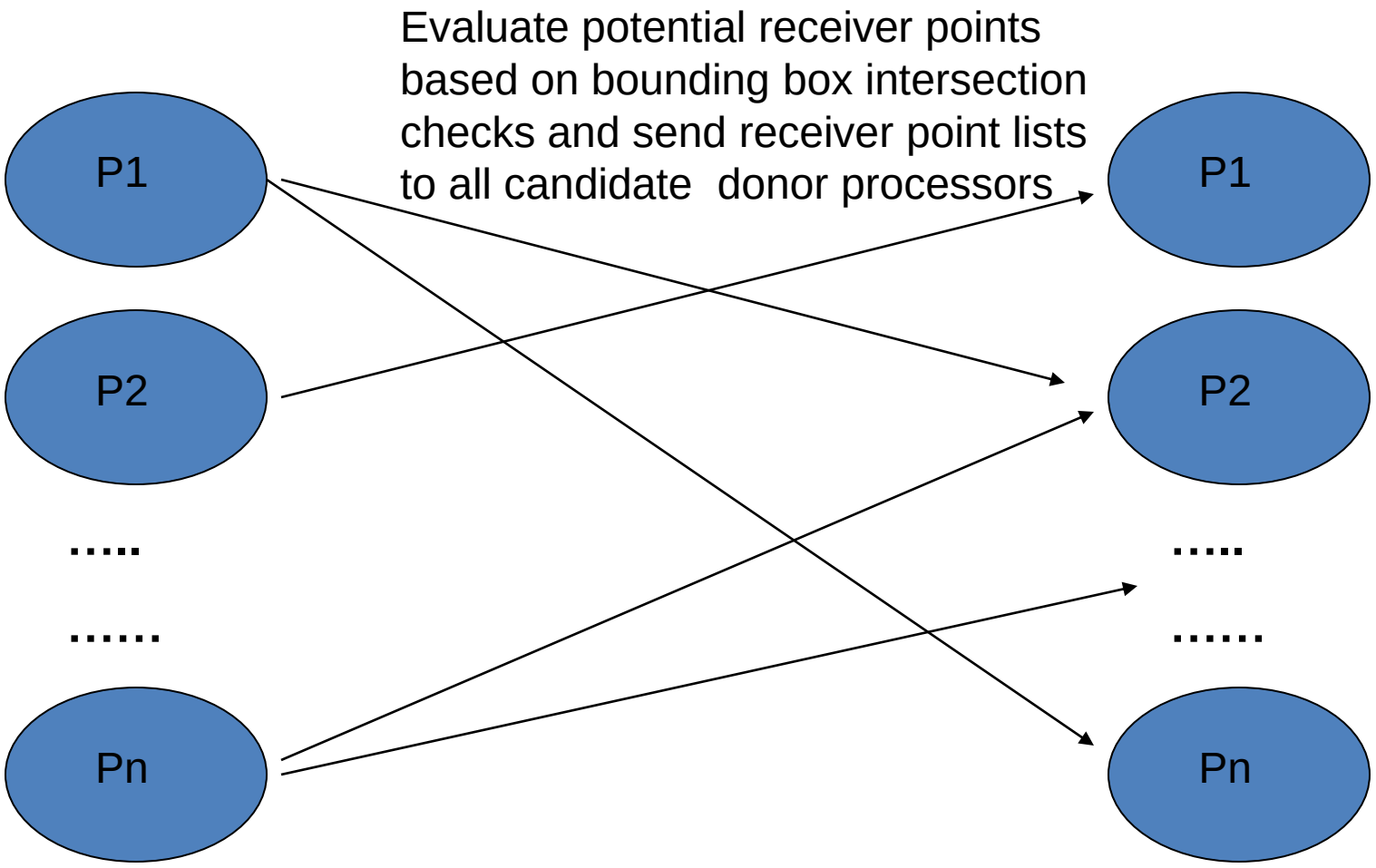


1. Search for strand donors
2. Build cart grid (external)
3. Search for strand/Cart donors
4. **Send donor cells to host partition that owns data**
5. Create donor/receptor lists
6. **Interpolate actual data and exchange**

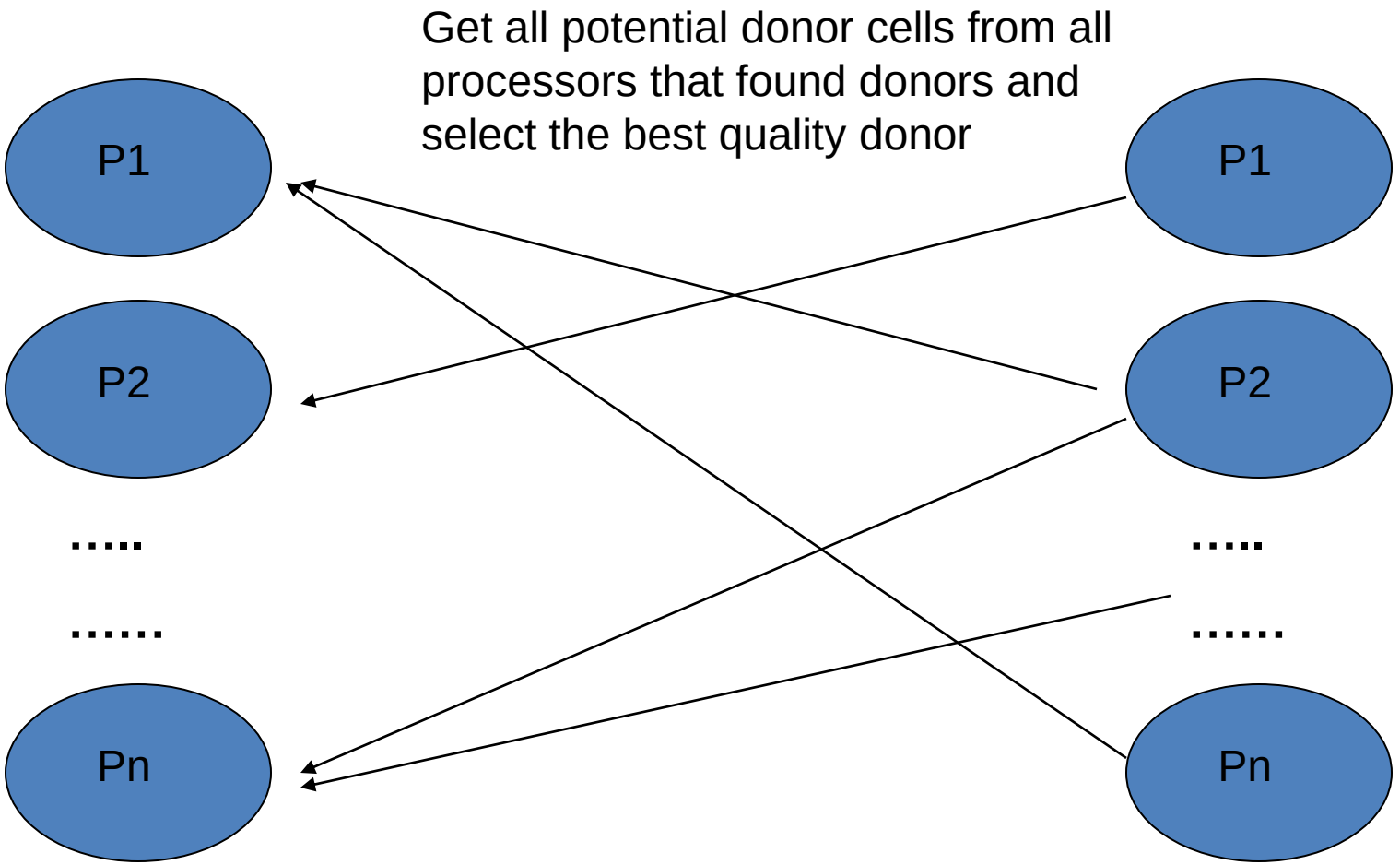
Communication in traditional overset grid assembly (PUNDIT)



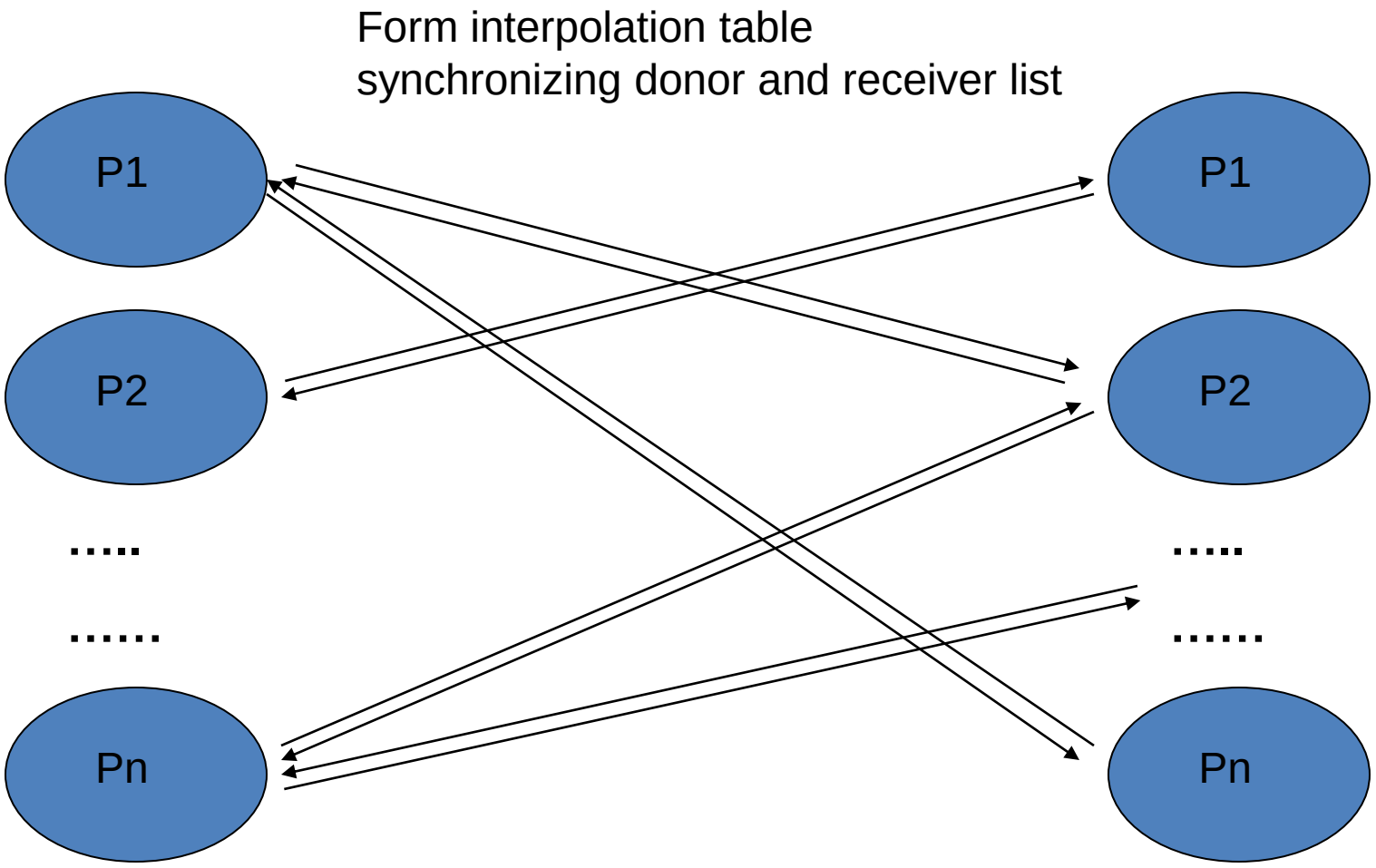
Communication in traditional overset grid assembly (PUNDIT)



Communication in traditional overset grid assembly (PUNDIT)



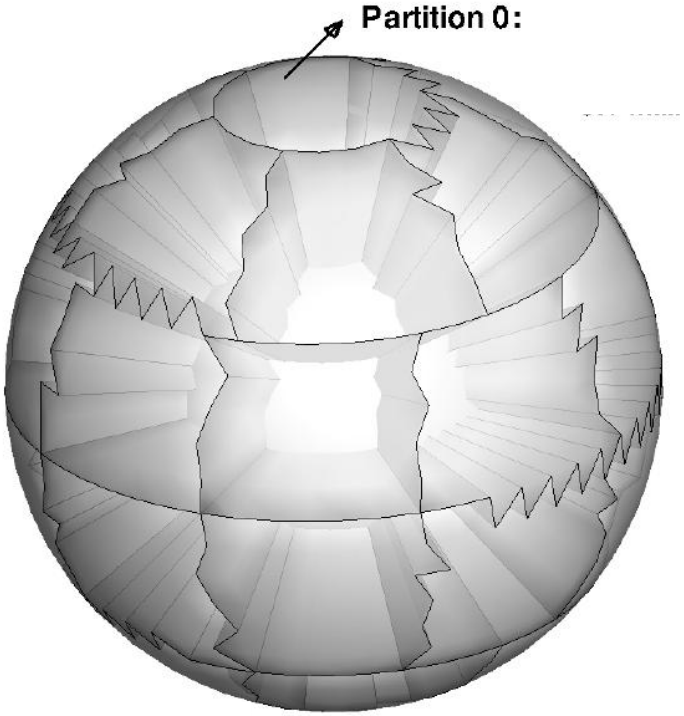
Communication in traditional overset grid assembly (PUNDIT)

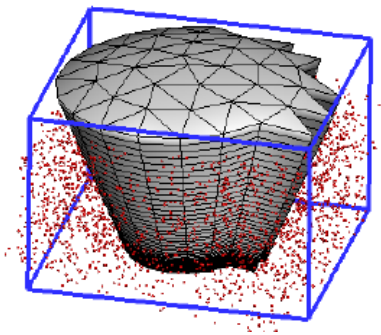


Application Cases

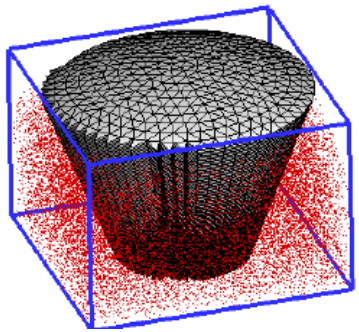
- Order analysis using Sphere geometry
- NACA0015 wing overset connectivity and flow solutions

Sphere problem





0.2 million cells
0.1 million QP

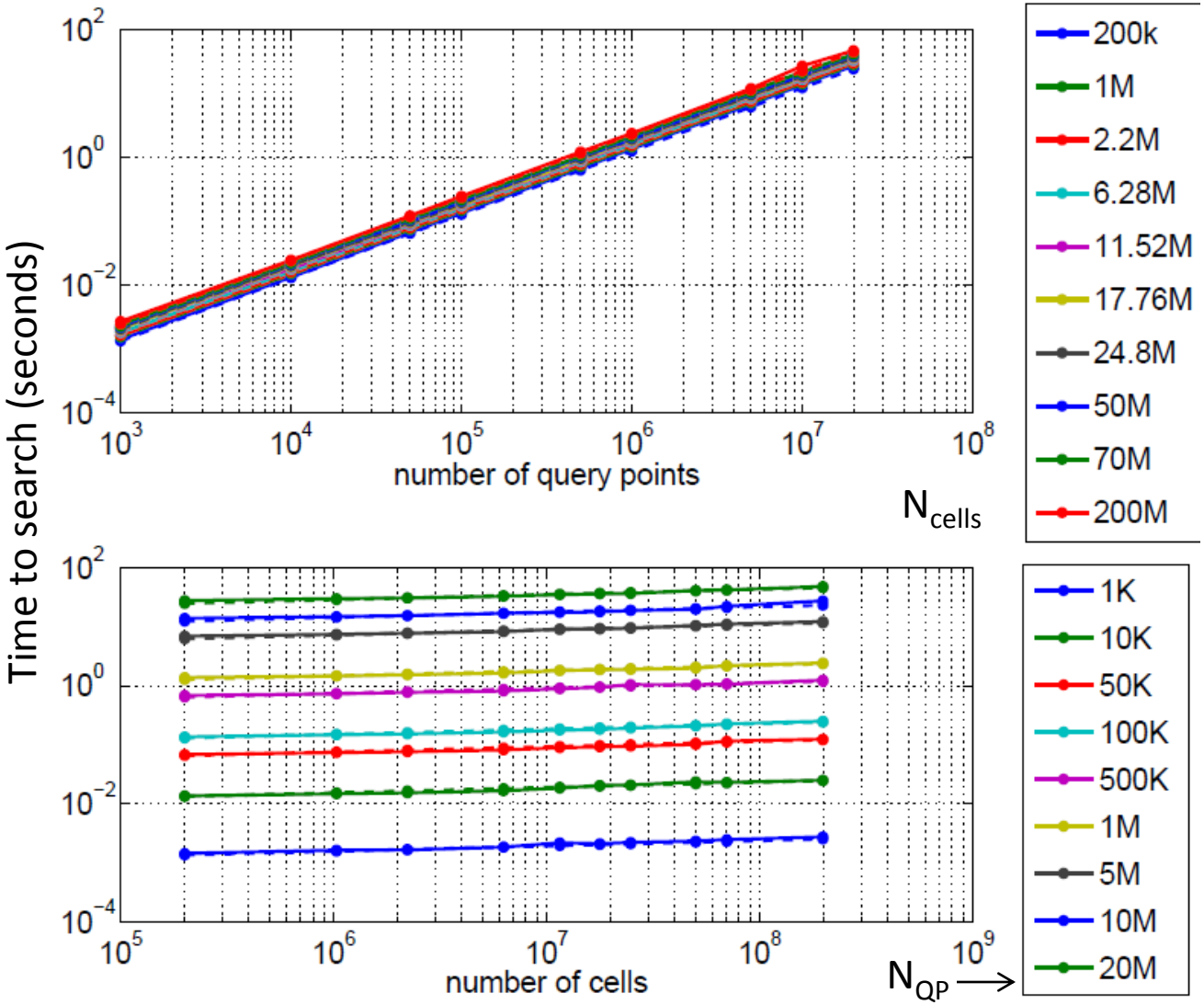


6.3 millions cells
1 million QP

1. Create a surface triangulation on a sphere using Delaunay approach (N triangles can be parameterized)
2. Create Strand grids by choosing a normal distribution with n layers
3. Create random query points (number can be parameterized)
4. Search for suitable donors for each of these query points
5. Perform the same search problem using the same grid in unstructured prismatic grid format using PUNDIT

-> Can quantify order of the search process by performing parameter sweeps

Wall-clock-time for donor search for varying sets of parameters

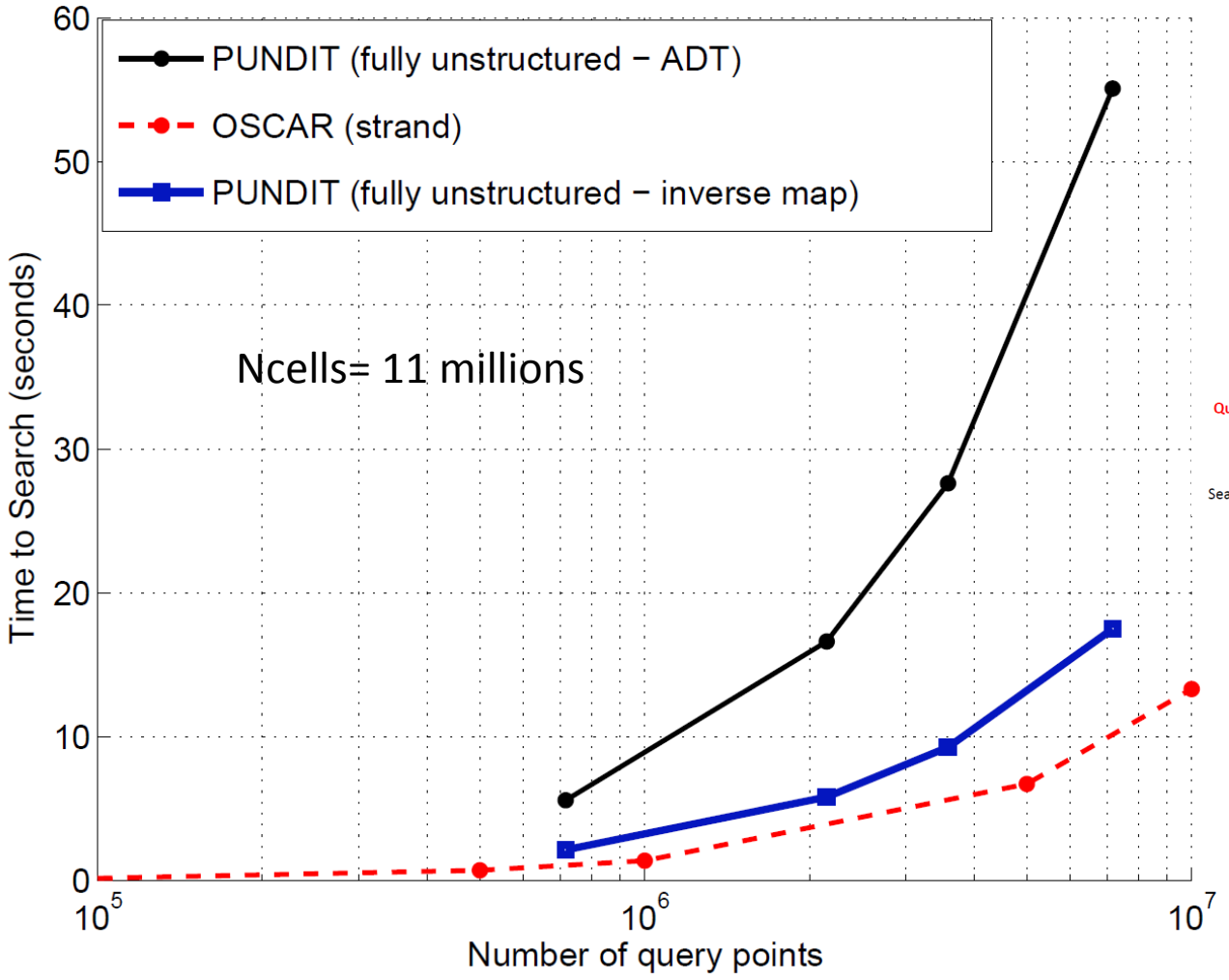


Time to search follows power rule: $Ts \approx K\ N_{QP}^{\ \alpha}\ N_{cells}^{\beta}$ (dashed lines)

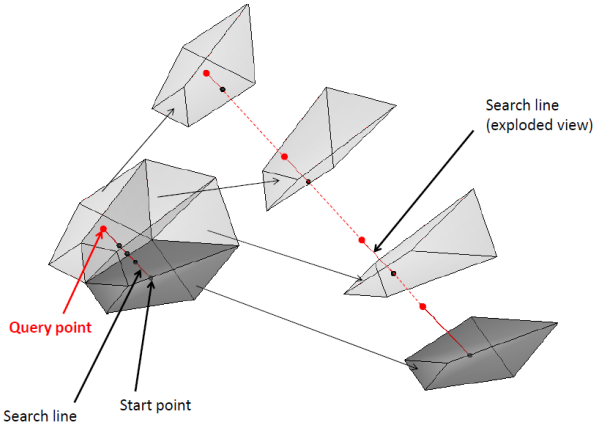
Strong function of the number of query points ($\alpha \approx 1$)

Weak function of the number of cells ($\beta \approx 0.1$)

PUNDIT vs. OSCAR comparison of search times



OSCAR is faster but not by much - why?



Time to search depends on number of line-walks, average line-walk length is the same (2 cells) for both codes

PUNDIT vs OSCAR order analysis

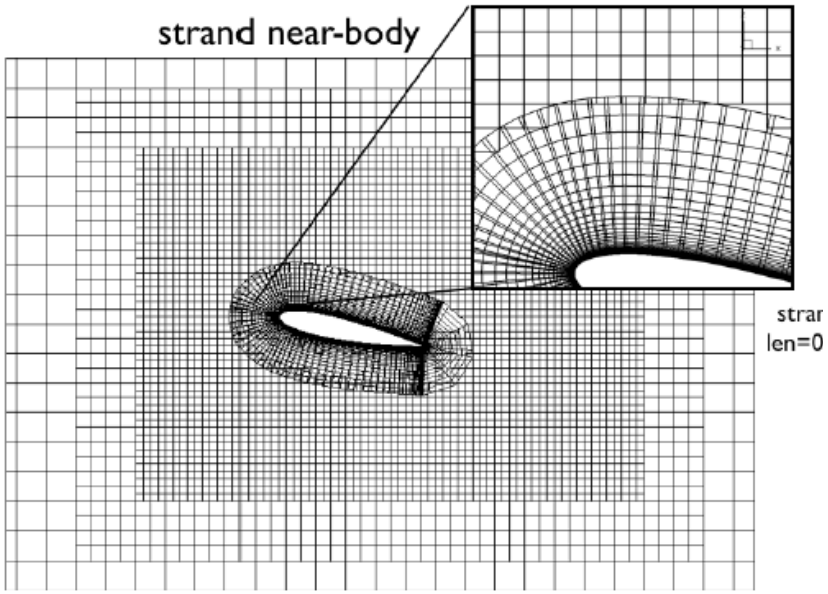
- PUNDIT search order

$$t_s = 1.9\ {N_p}^{0.95} {N_c}^{0.05}$$

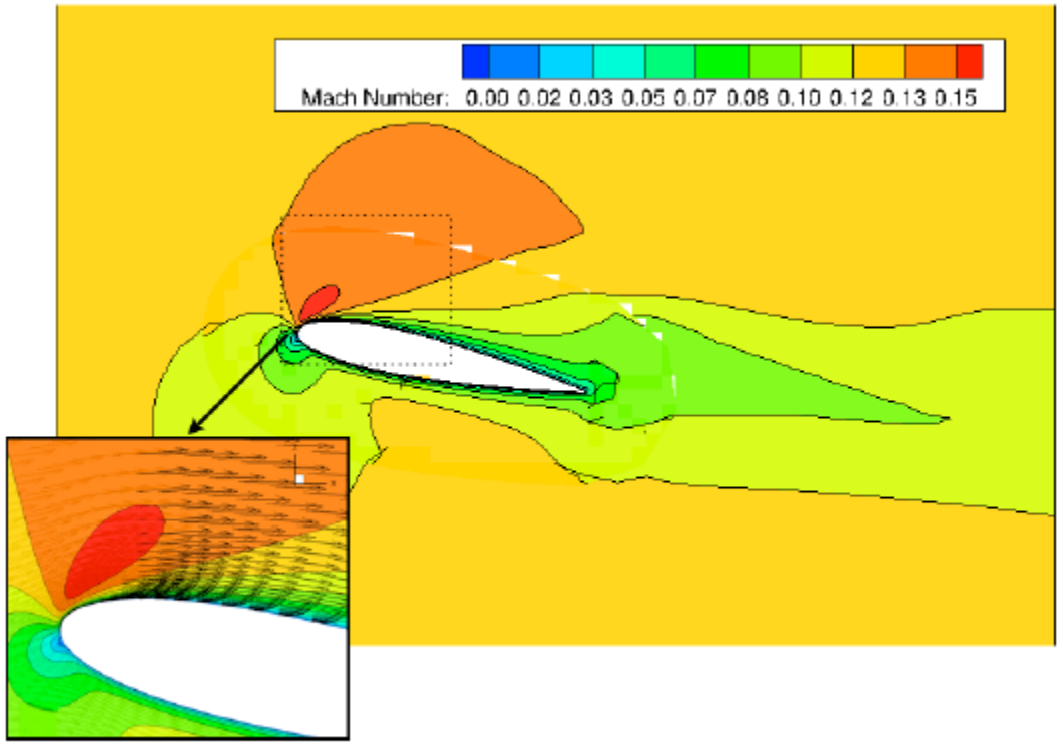
- OSCAR search order

$$t_s = 0.55\ {N_p}^{0.99} {N_c}^{0.07}$$

NACA 0015 wing



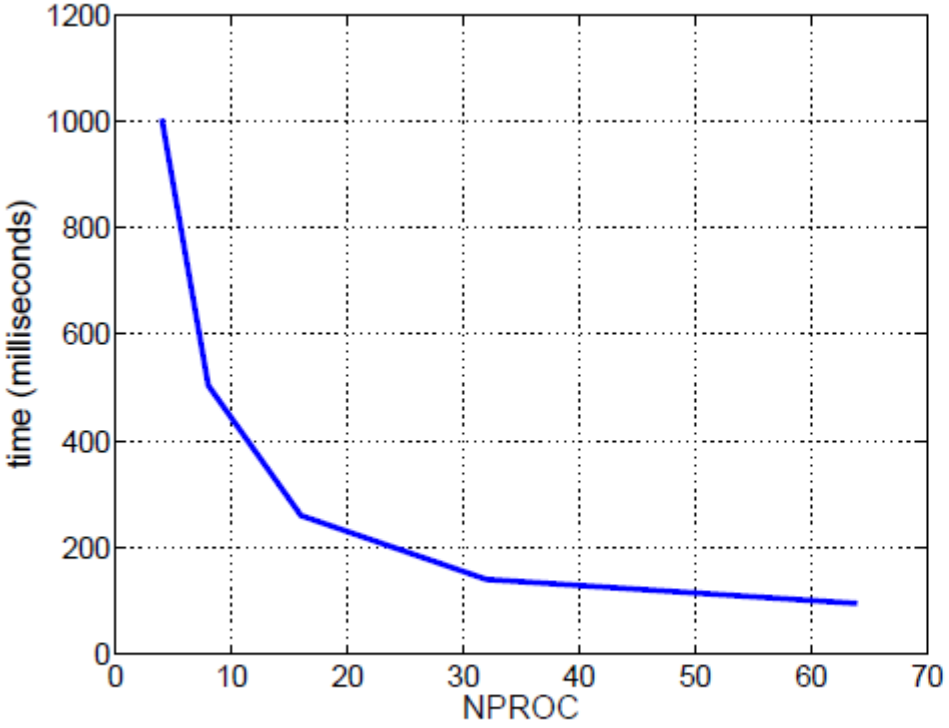
(a) strand len = 1.0r



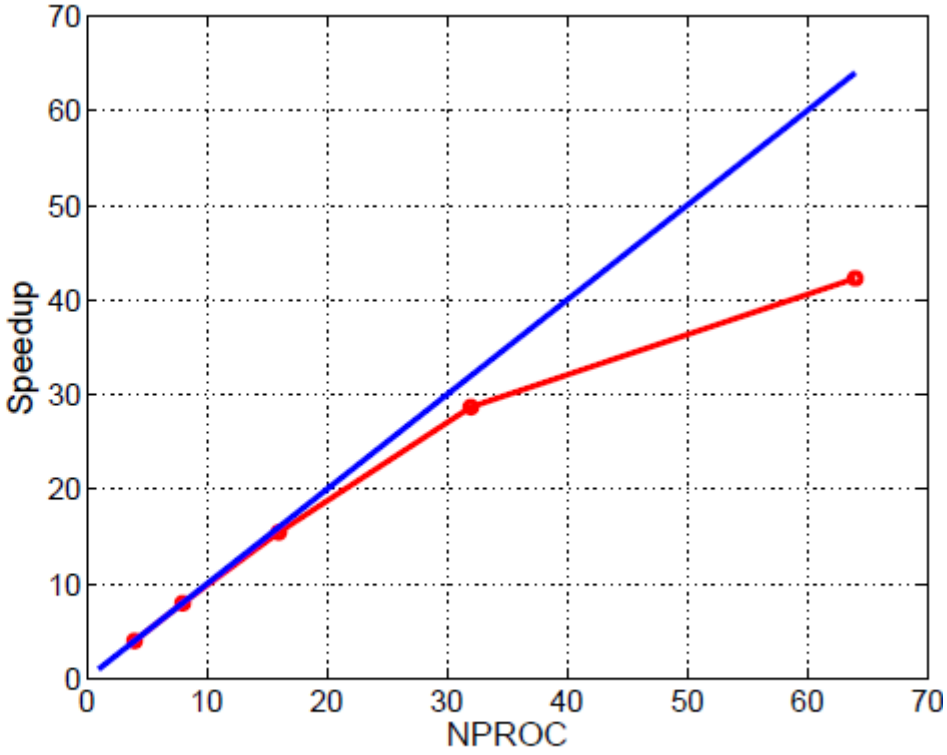
(a) strand len = 0.5c

2.5 million strand cells
11 million off-body grid nodes

NACA0012
scalability of OSCAR



(a) Execution time in milli seconds



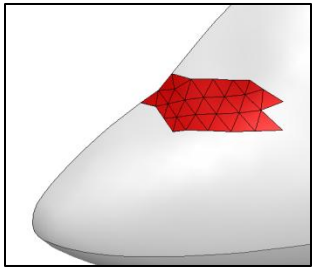
(b) Ideal vs observed speedup

Scaling degrades on 64 cores because of communication, note that the time required is only 20 milliseconds and the cluster used had only gigE network fabric which had almost 8-10ms of latency

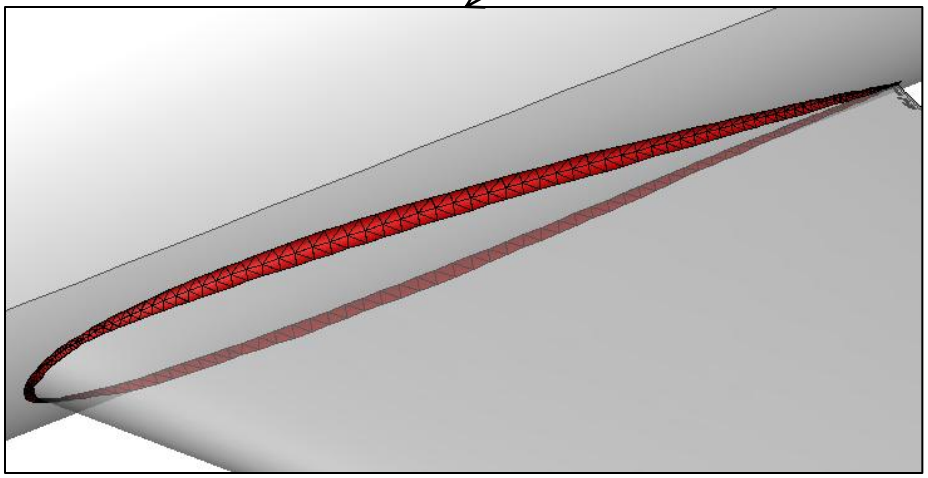
Strand Overset grid assembly issues for real geometry

- Bad – severely warped/negative volumes in concave regions
- Self intersections and how to clear them without low quality donation
- Should off-body get really close where strand intersections happen?

Strand self-overlap and invalid cells



DLR-F6 : surface elements with invalid strand cells (may require changed clipping)



Need method to identify and isolate regions of self-overlap

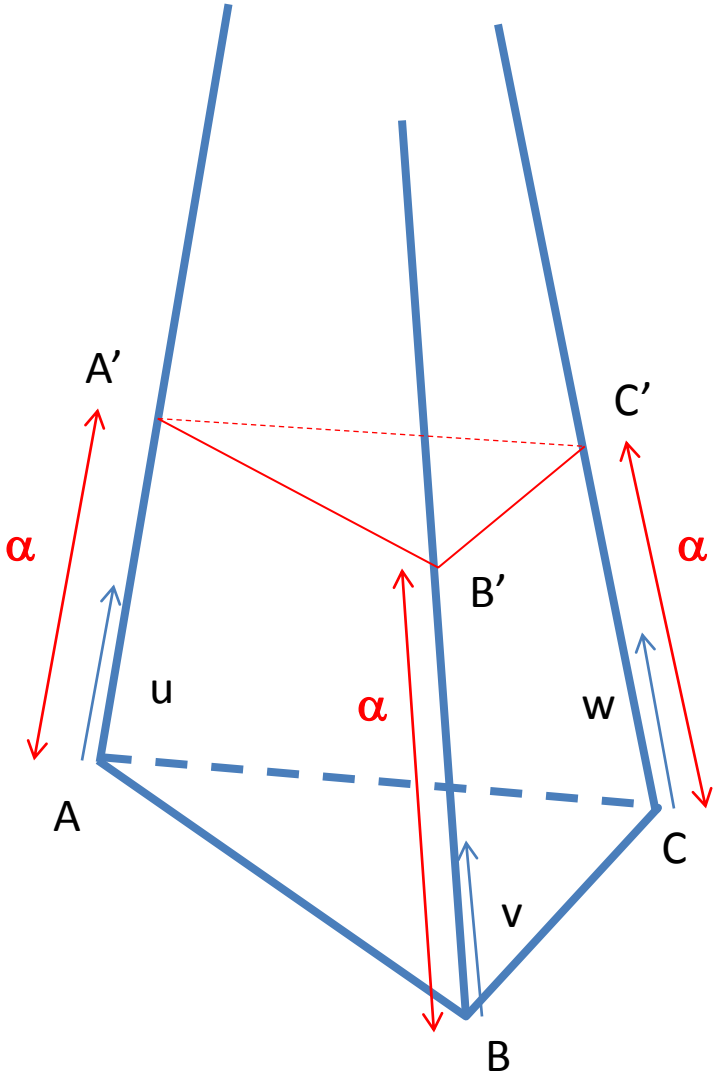
Focus on wing-fuselage junction

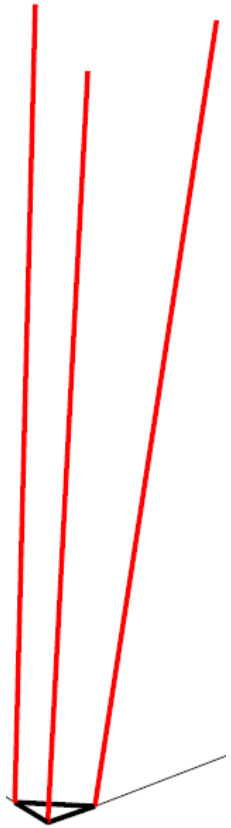
Determining clipping level for invalid cells

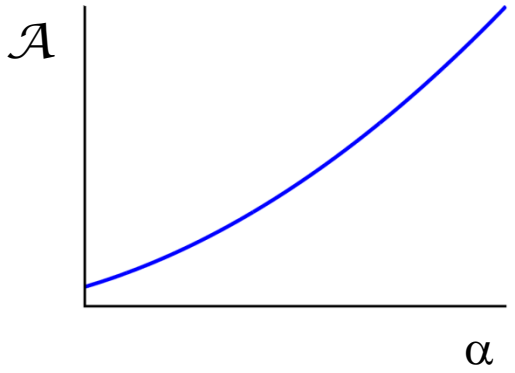
Area of triangle (A',B',C') :

$$\mathcal{A}(\alpha)^2 = \{\, p_0\ p_1\ p_2\ p_3\ p_4 \,\} \begin{Bmatrix} 1 \\ \alpha \\ \alpha^2 \\ \alpha^3 \\ \alpha^4 \end{Bmatrix}$$

find when area is minimal : $d\mathcal{A} / d\alpha = 0$:
3 values of α







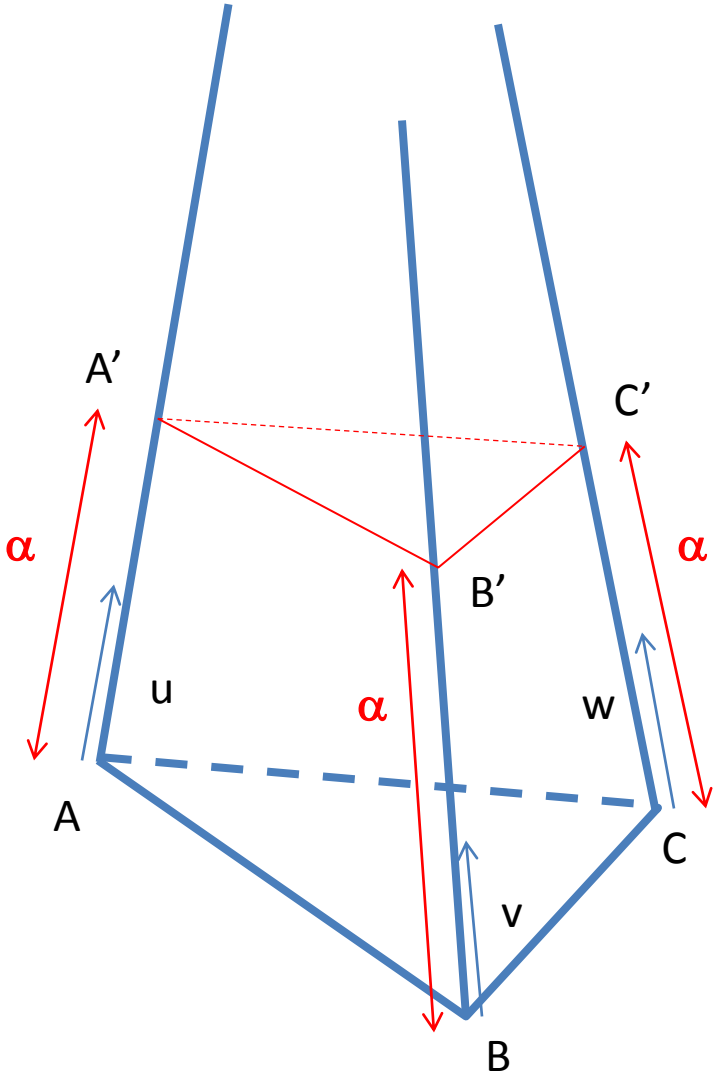
3 negative real roots:
no clipping required

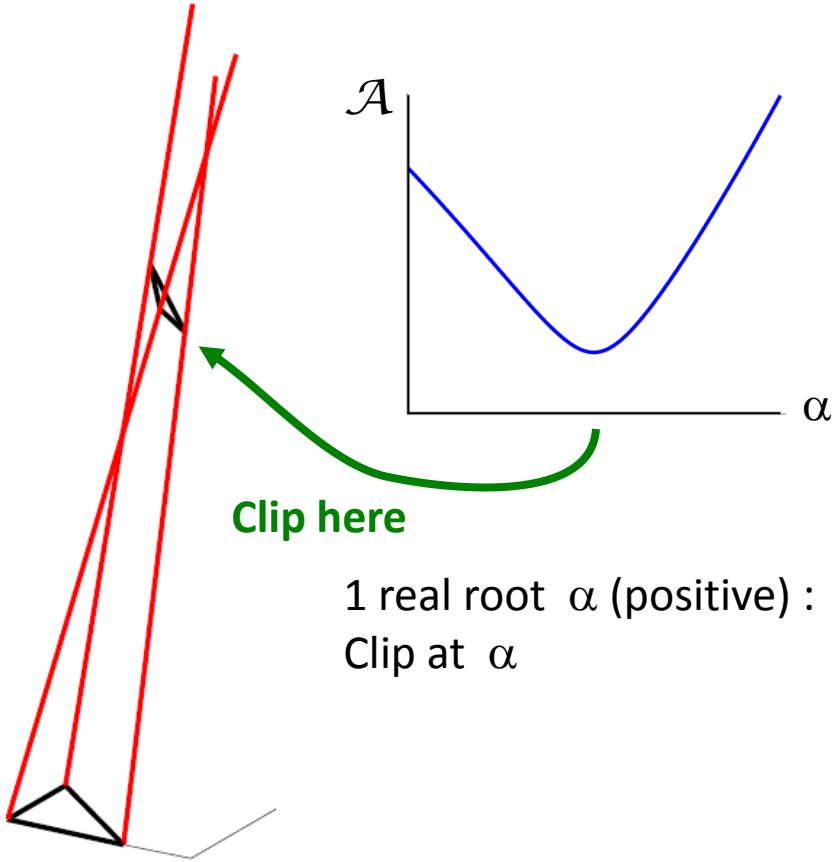
Determining clipping level for invalid cells

Area of triangle (A',B',C') :

$$\mathcal{A}(\alpha)^2 = \{\, p_0\ p_1\ p_2\ p_3\ p_4 \,\} \begin{Bmatrix} 1 \\ \alpha \\ \alpha^2 \\ \alpha^3 \\ \alpha^4 \end{Bmatrix}$$

find when area is minimal : $d\mathcal{A} / d\alpha = 0$:
3 values of α





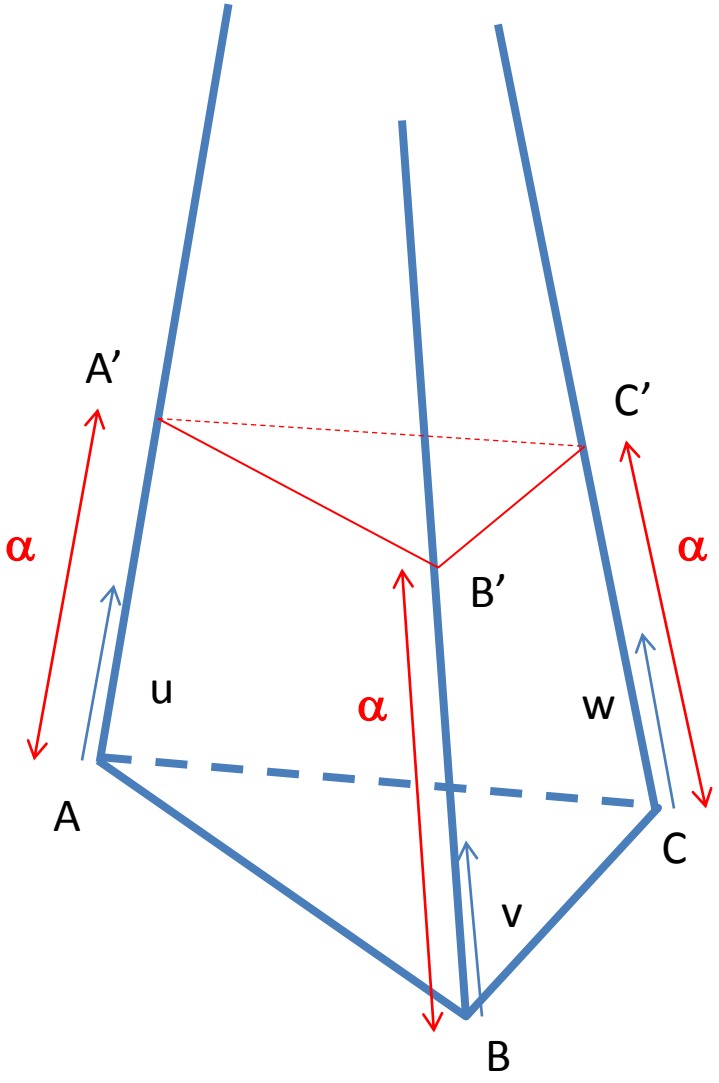
1 real root α (positive) :
Clip at α

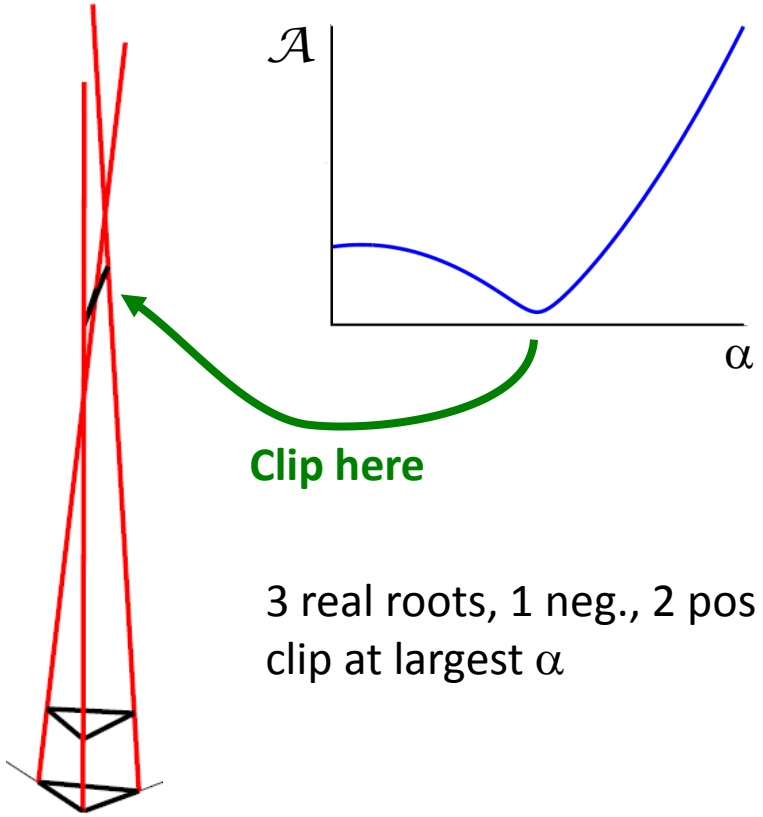
Determining clipping level for invalid cells

Area of triangle (A',B',C') :

$$\mathcal{A}(\alpha)^2 = \{\, p_0\ p_1\ p_2\ p_3\ p_4 \,\} \begin{Bmatrix} 1 \\ \alpha \\ \alpha^2 \\ \alpha^3 \\ \alpha^4 \end{Bmatrix}$$

find when area is minimal : $d\mathcal{A} / d\alpha = 0$:
3 values of α



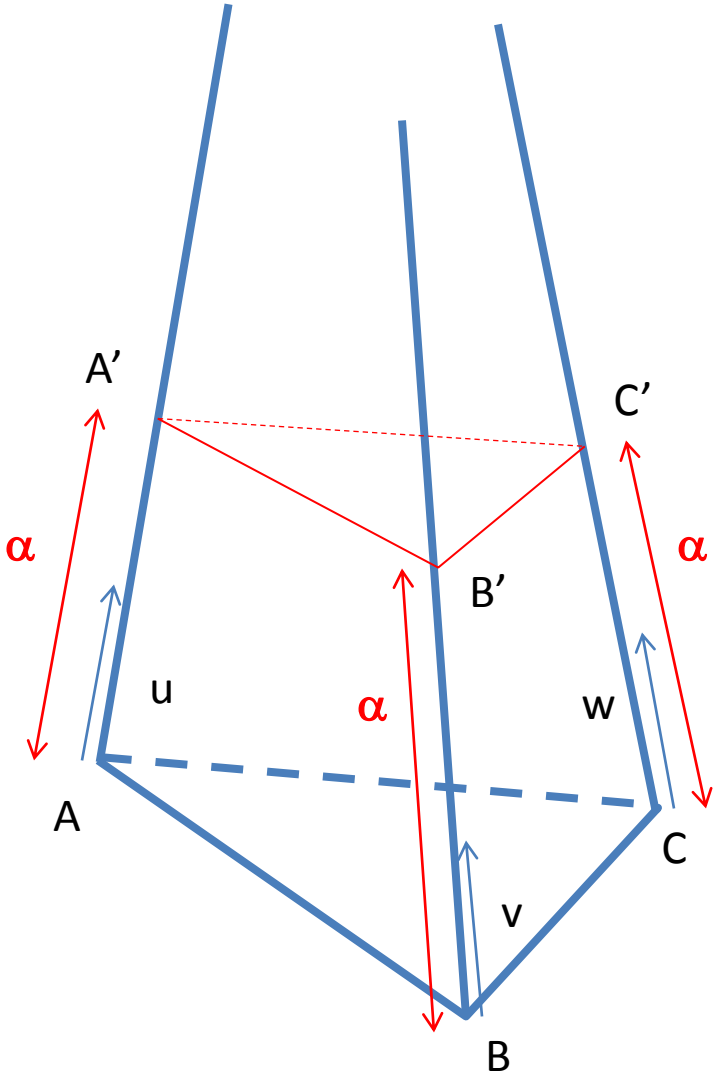


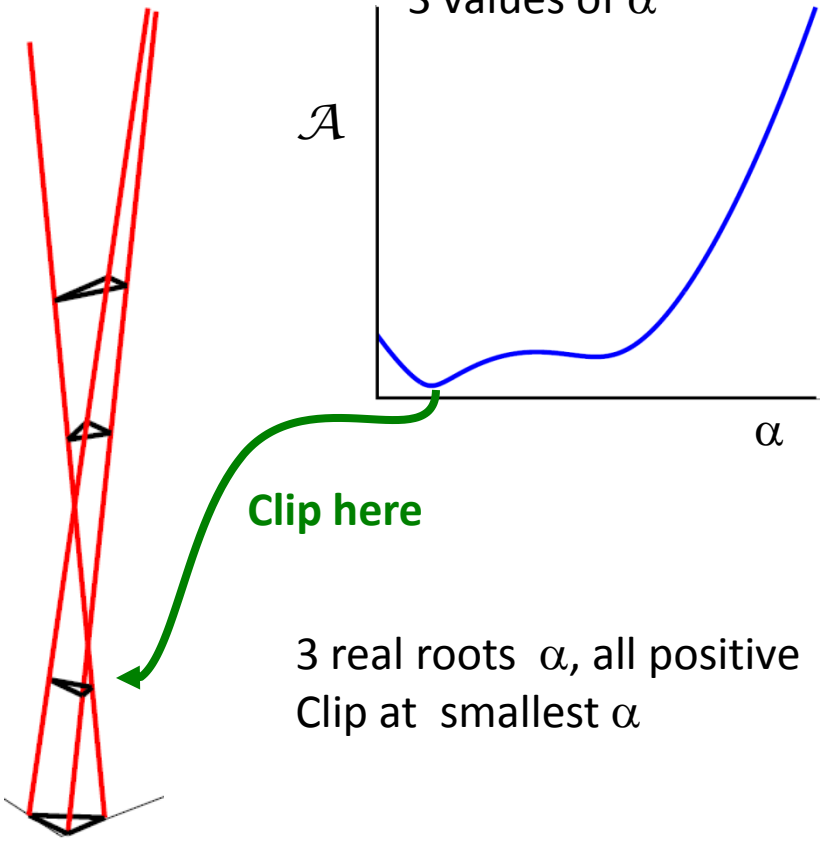
3 real roots, 1 neg., 2 pos :
clip at largest α

Determining clipping level for invalid cells

Area of triangle (A',B',C') : $\mathcal{A}(\alpha)^2 = \{ p_0\ p_1\ p_2\ p_3\ p_4 \}$ $\begin{Bmatrix} 1 \\ \alpha \\ \alpha^2 \\ \alpha^3 \\ \alpha^4 \end{Bmatrix}$

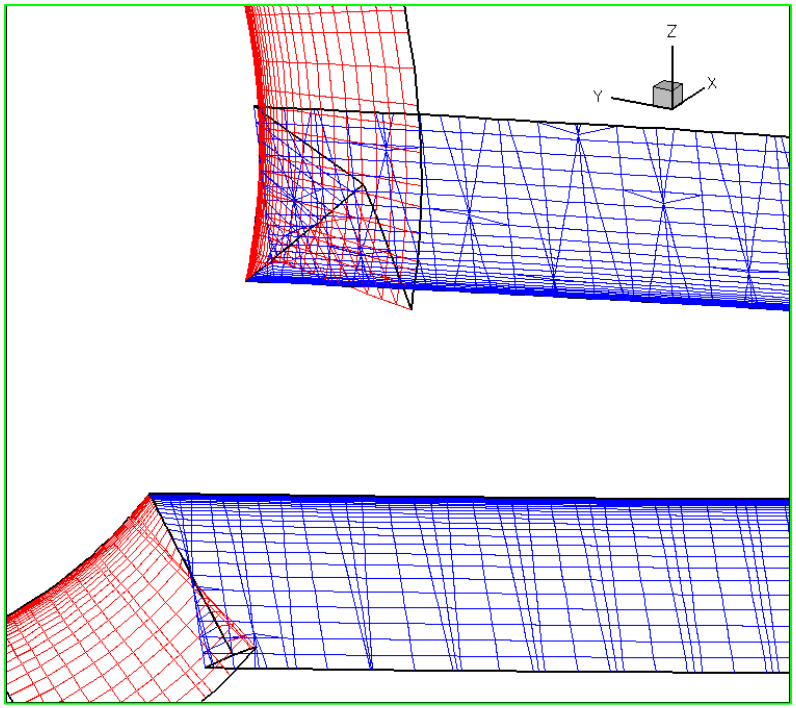
find when area is minimal : $d\mathcal{A} / d\alpha = 0$:



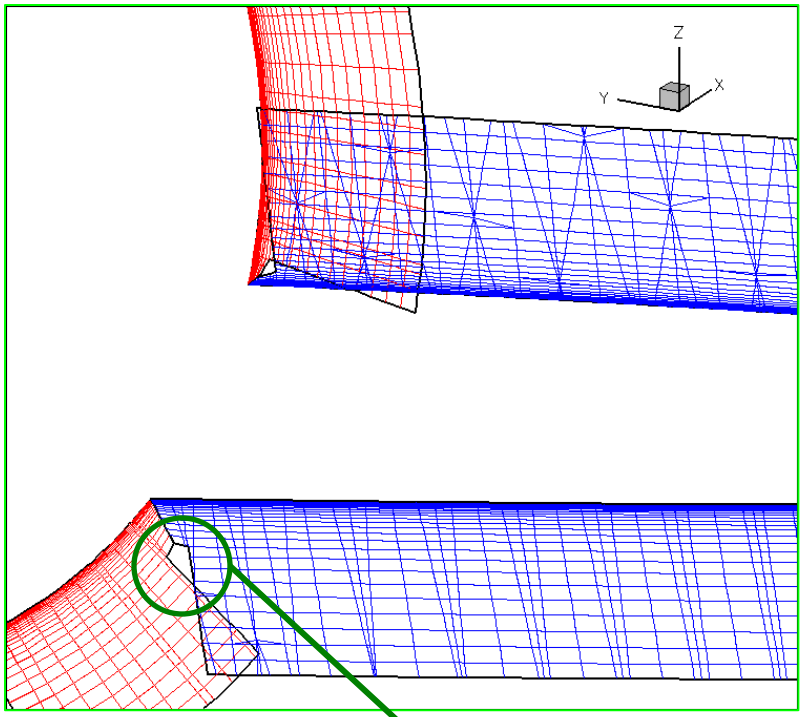


3 real roots α, all positive
Clip at smallest α

Removing invalid cells



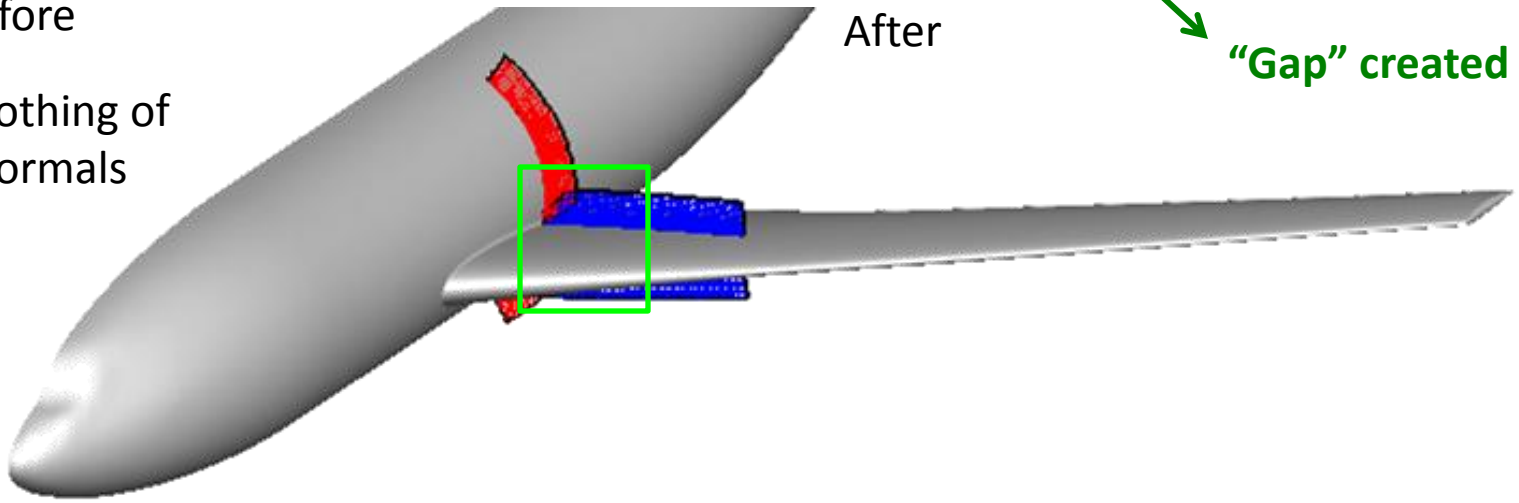
Before



After

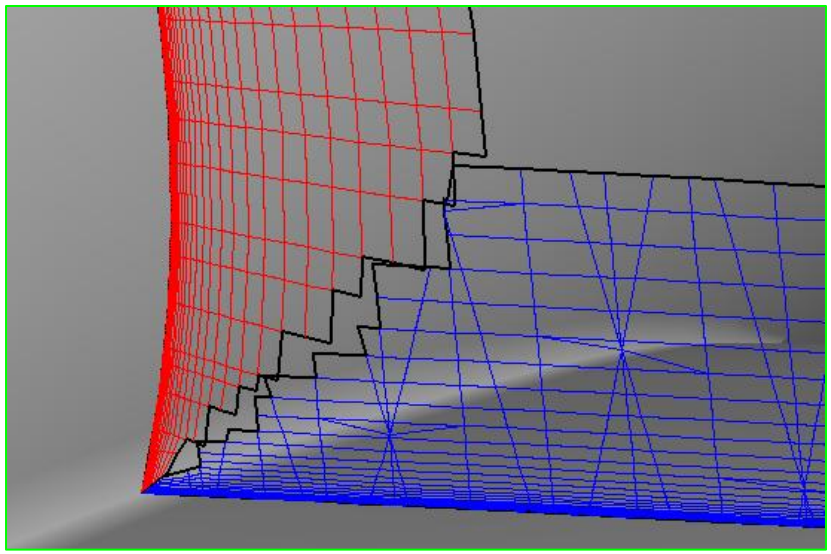
“Gap” created

(no smoothing of strand normals used)

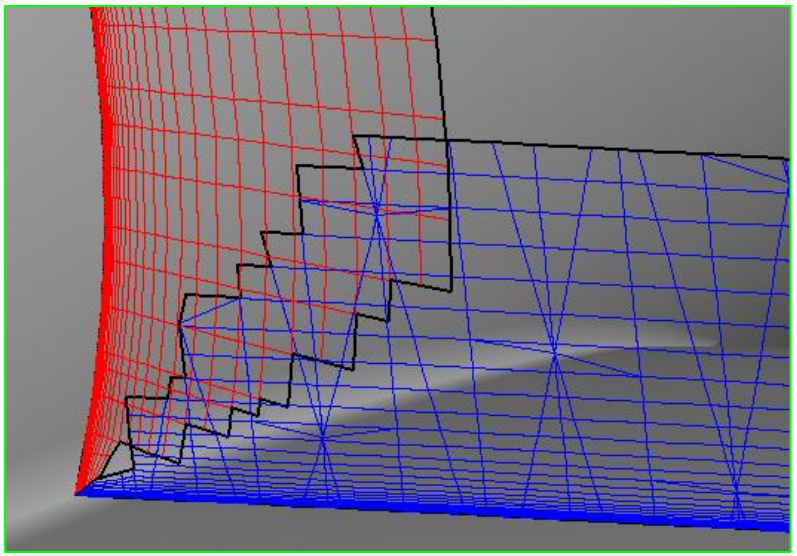


Identifying receptor cells in region of self-overlap

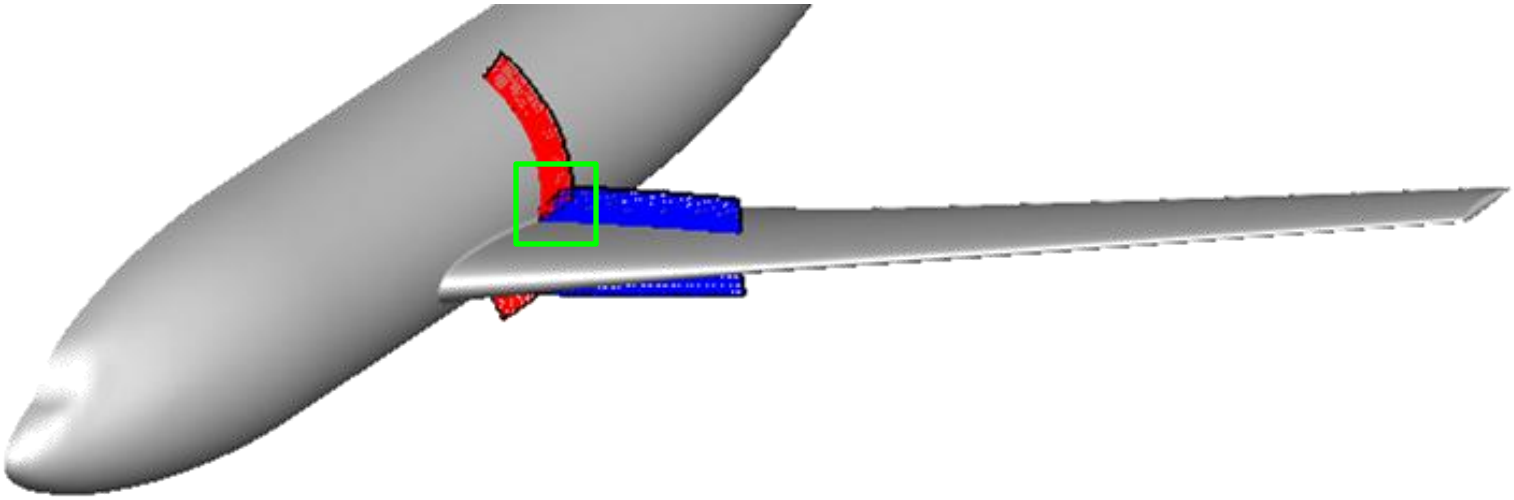
Cells with better donors become receptor cells:



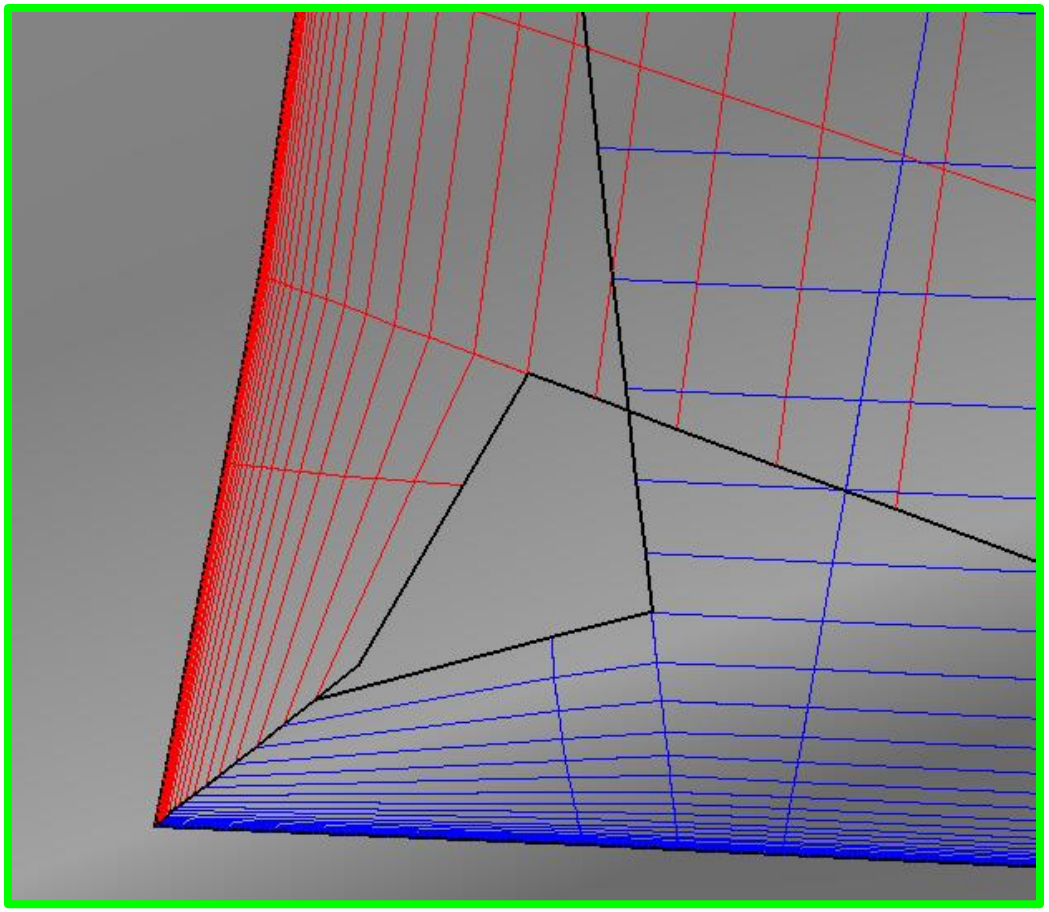
No overlap (donor from Cartesian mesh)

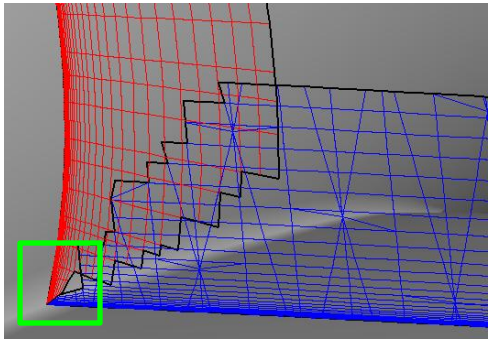


With overlap (donor from strand mesh)



Donor cells from self strand mesh: “gap” problem





When overlap donors are found in the strand mesh :
small volume with no donor cells exists, require either:

- background Cartesian mesh (patch)
- modification of the strand mesh structure

Concluding observations

- Strand/Cartesian meshing approach is a viable overset grid solution, provides an opportunity to streamline future CFD solutions
  - - hands-off, CAD to flow solution
  - Better scalability
- Demonstrated improvements
  - Search efficiency for sphere problem
  - Full solution for an overset wing problem using Strand/Cart architecture (details in next talk)
    - Fast domain connectivity (2% or time step time)

Future Work

- Application of strand/Cartesian technology for more realistic test cases
- Improve asymptotic order of search by further algorithm optimizations
  - Improved rasterization
  - Improvements to line-walk search
- Robust approach to clear self intersections and provide means of solving the “gap problem”
  - Telescoping Cartesian grids to cover the gap
  - Multiple strands to prevent gap
  - Have few strands have two normals associated with them
  - Solver modification to accommodate bad volume cells as fringes
  - Nearest-neighbor type interpolation at face-centroids