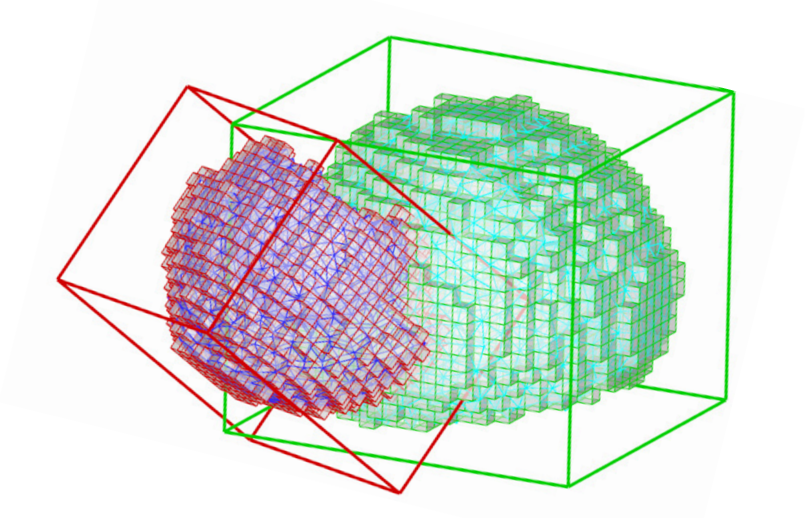
Robustness and Accuracy of Donor Search Algorithms on Partitioned Unstructured Grids



Beatrice Roget and Jay Sitaraman
University of Wyoming

10TH SYMPOSIUM ON
OVERSET COMPOSITE GRIDS
AND SOLUTION TECHNOLOGY
SEPTEMBER 20-23, 2010
NASA AMES RESEARCH CENTER
MOFFETT FIELD, CALIFORNIA USA

Outline

- Motivation
  - Introduction to PUNDIT
  - Characterization of search robustness issues
  - Description of current search algorithm (Approximate Inverse Map)
- Alternating Digital Tree Search
- Exact Inverse Map
- Performance results for few test cases
- Conclusions

What is PUNDIT?

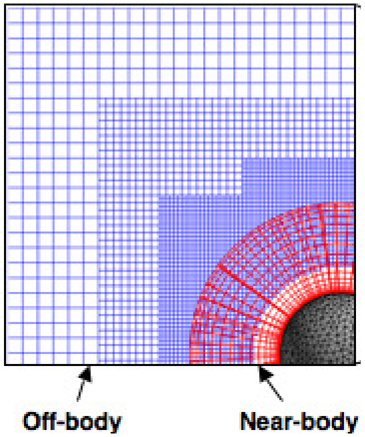
- PUNDIT stands for Parallel Unsteady Domain Information Transfer
  - Domain connectivity module in HELIOS (Developed through DoD/HIARMS/CREATE-AV program)
- PUNDIT provides fully automated domain connectivity support in parallel (distributed memory) computing systems
  - Requests only local grid and solution data as known to solvers
  - Uses solver based grid partitioning
- Salient Features of PUNDIT
  - Implicit fringe determination search strategy, i.e fringes are not explicitly specified
  - In the case of multiple overlapping grids, grids with best resolution is used for flow solution and all others are interpolated
  - More search operations than traditional explicit hole-cutting techniques since candidate receptor points can include the entire grid
  - Minimum hole-cutting using ray-tracing

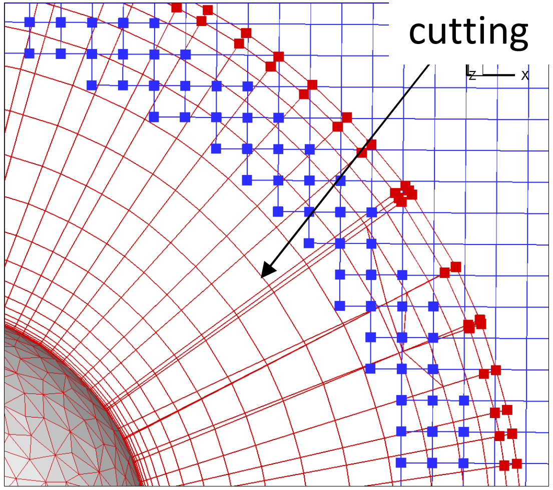
Sitaraman, J., Floros, M., Wissink, A. and Potsdam, M., “Parallel Domain Connectivity Algorithm For Unsteady Flow Computations Using Overlapping And Adaptive Grids”, *Journal of Computational Physics*, Vol. 229, Issue 12, June 2010.

Implicit Fringe Determination

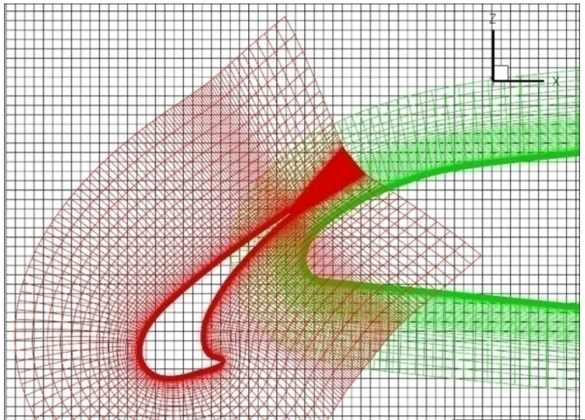
Interpolate flow variables between multiple meshes and solvers every solution iteration

Determine *fringes, donors, hole points, interpolation weights* <u>*(Domain Connectivity)*</u>

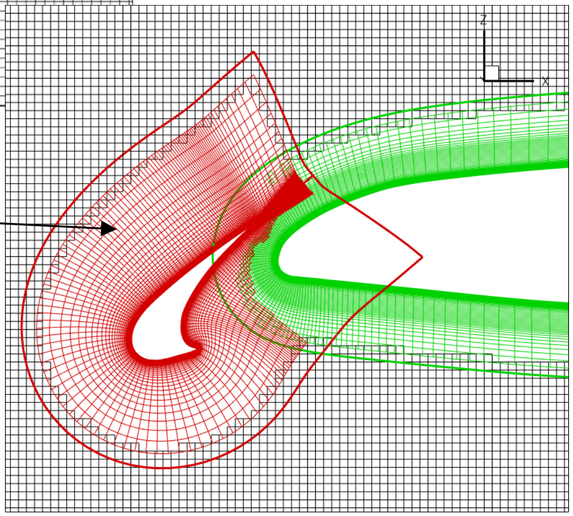




Explicit hole cutting



- Automation
- Optimal connectivity
- Interpolation accuracy
- More expensive

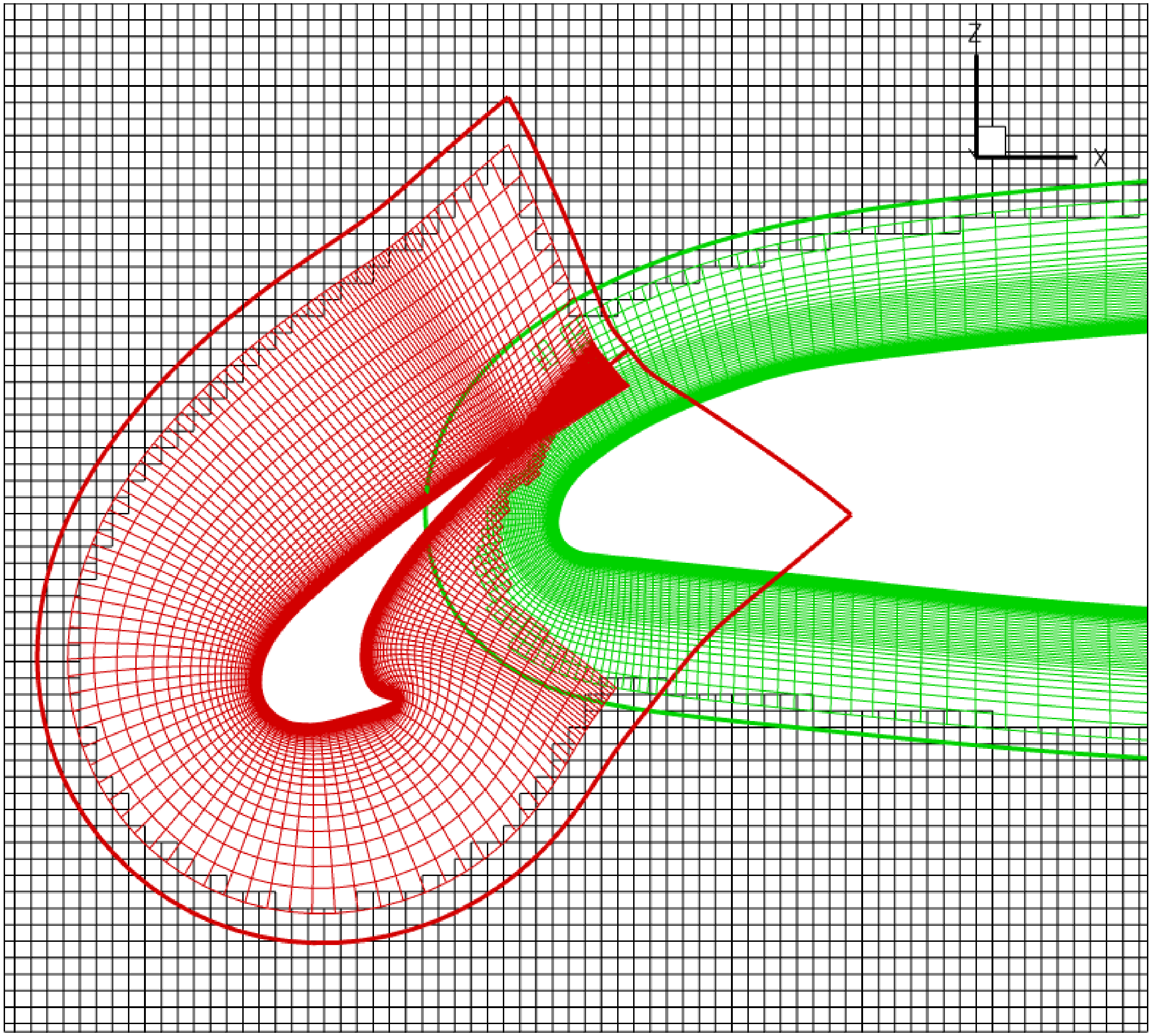


Implicit hole cutting

PEGASUS (Rogers 2003)

NAVAIR (Lee 2004)

Implicit-hole cutting

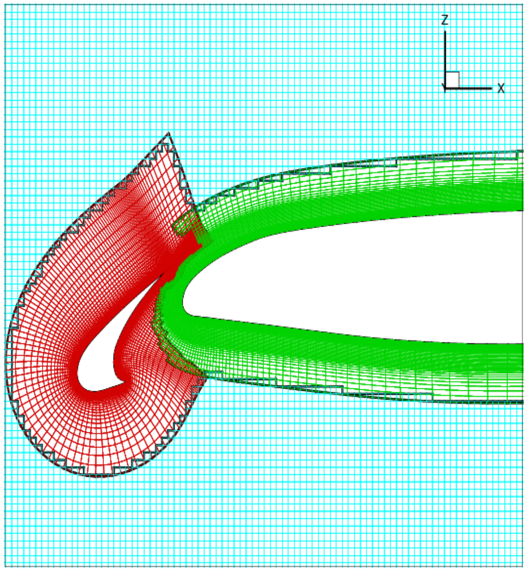


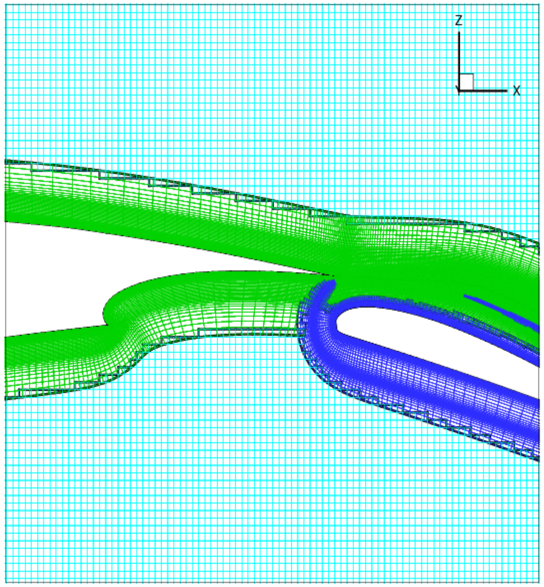
Solver points:

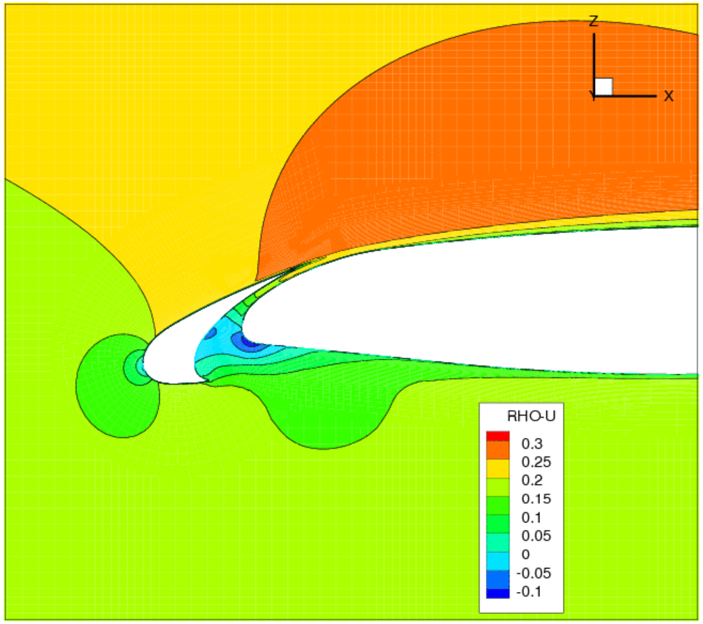
Grid nodes were flow variables are being solved

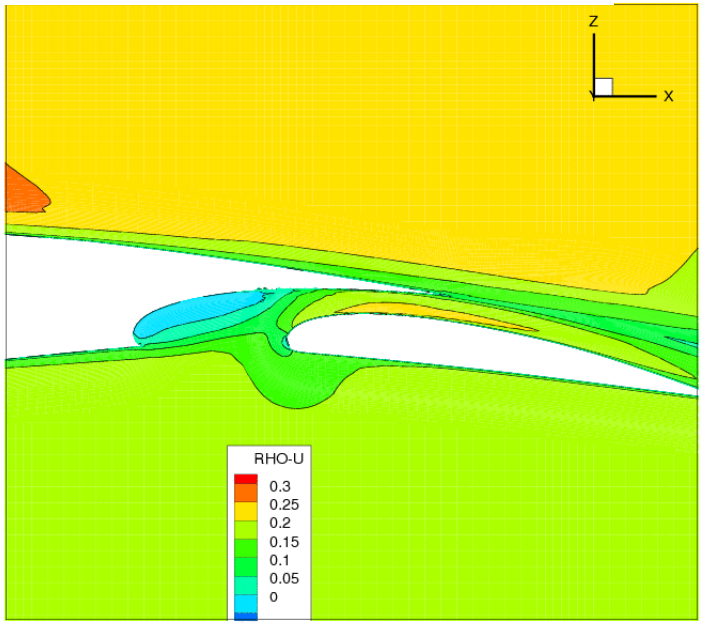
Fringes and solver points are mutually exclusive to maintain donor quality

AGARD A2 slotted airfoil solution



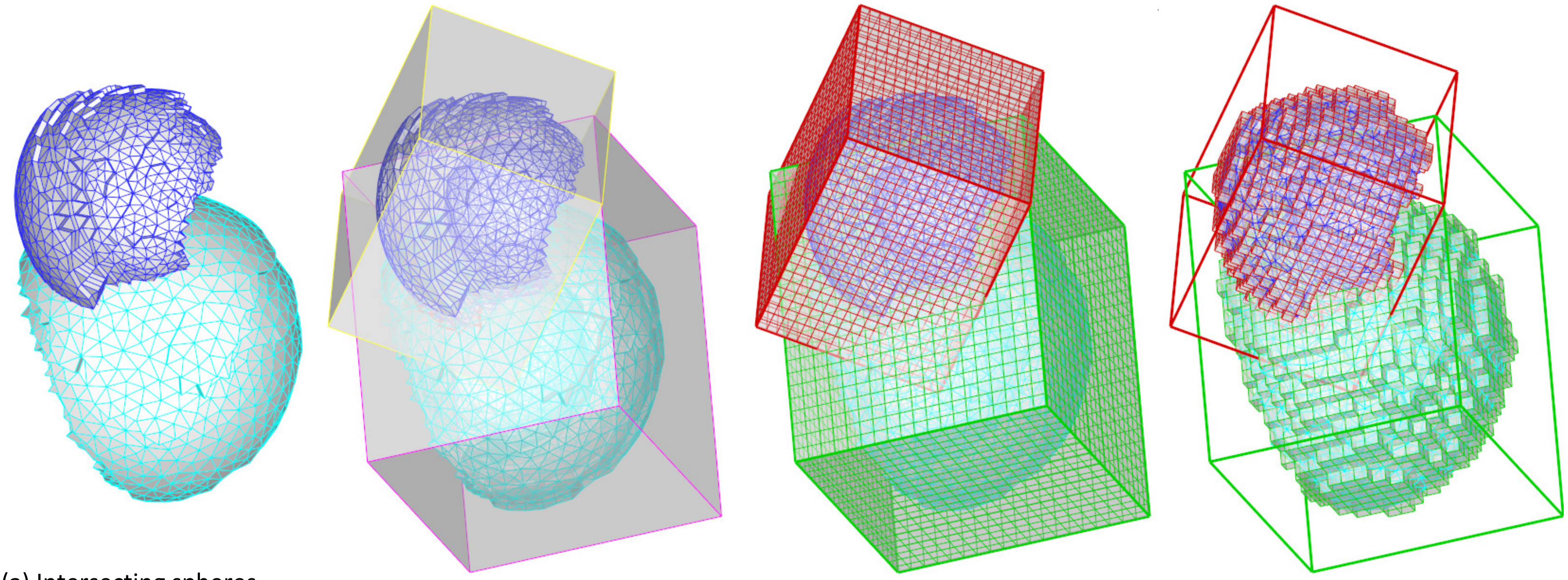






Original Search Algorithm

- Meta-data structure
  - Approximate Inverse Map (aIM) for efficient search



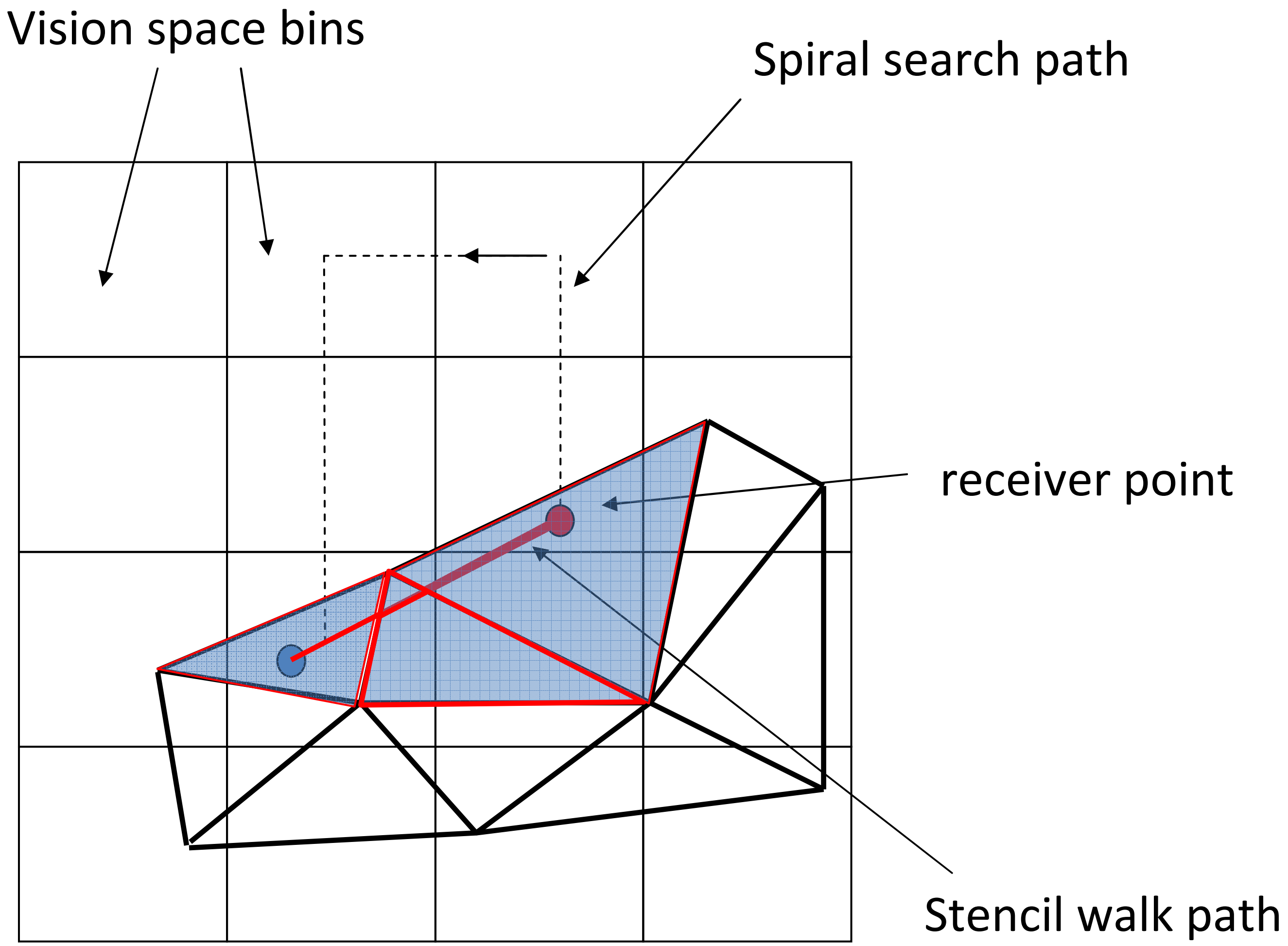
(a) Intersecting spheres (partitioned grids shown)

(b) Oriented bounding boxes created using inertial bisection

(c) Inverse map is created by dividing the bounding boxes in to smaller sub-blocks and re-ordering the cells based on **"cell-center"** containment

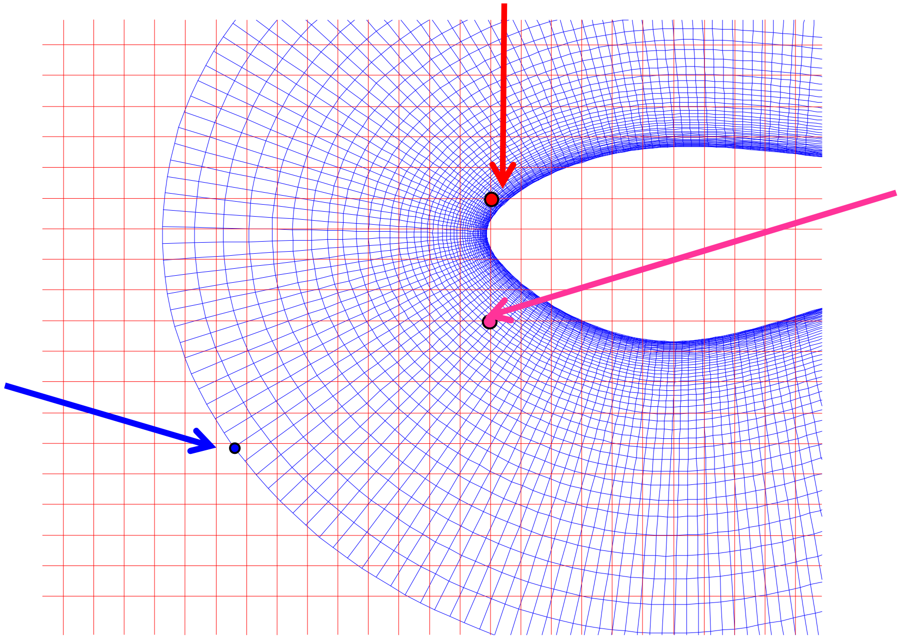
(d) Only sub-blocks that contain mesh cells are shown .

Donor search using stencil walk



Search Issues: Orphans and Incomplete Fringes

- **Orphan point**
  - **Field point** that immediately borders a hole-point (both near-body and off-body)
  - **Compulsory receptor** that did not get a donor (only in near-body meshes)
- **Incomplete Fringe point**
  - **Field point** that contain a hole-point in its discretization stencil (both near-body and off-body)



Orphan point (borders hole point)

Incomplete Fringe point

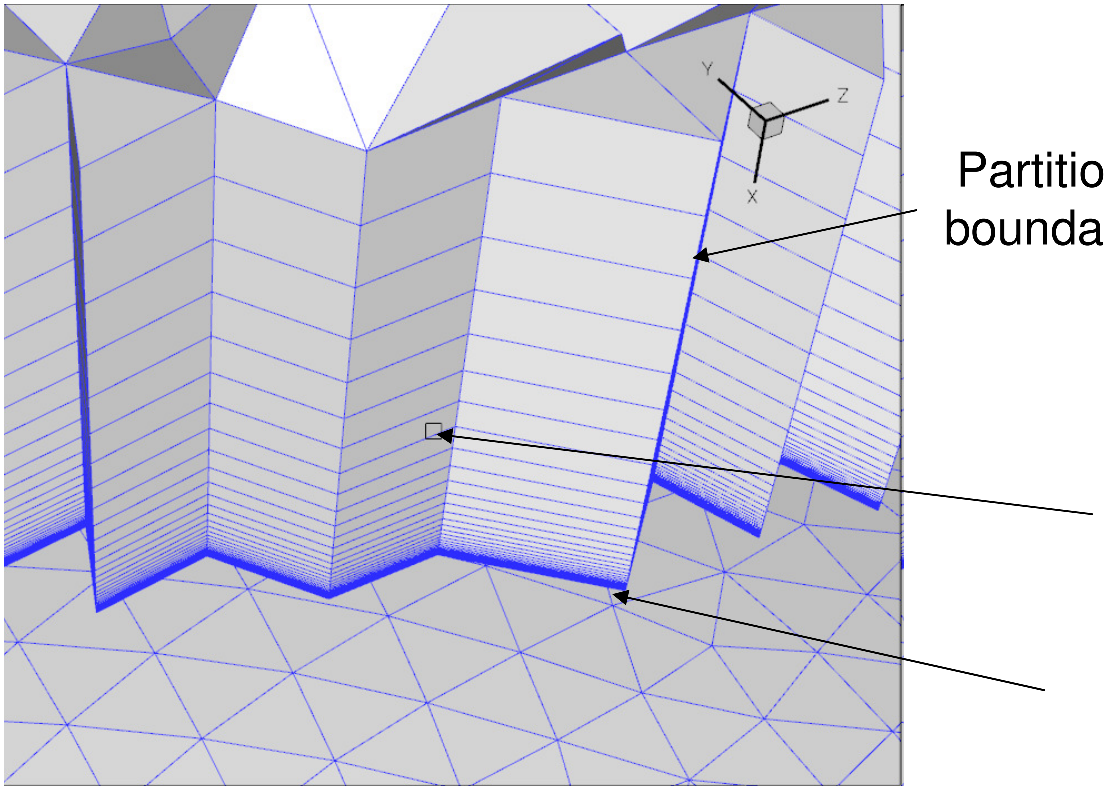
Orphan point (compulsory receptor)

Search Issues: Orphans and Incomplete Fringes

•We found most test cases have a few off-body orphans and incomplete fringes

•Further investigation showed that most of these orphans are generated because the donor-search fails at partition boundaries

•Problem may be in the extension of the stencil-walk algorithm (especially to walk back into the domain)



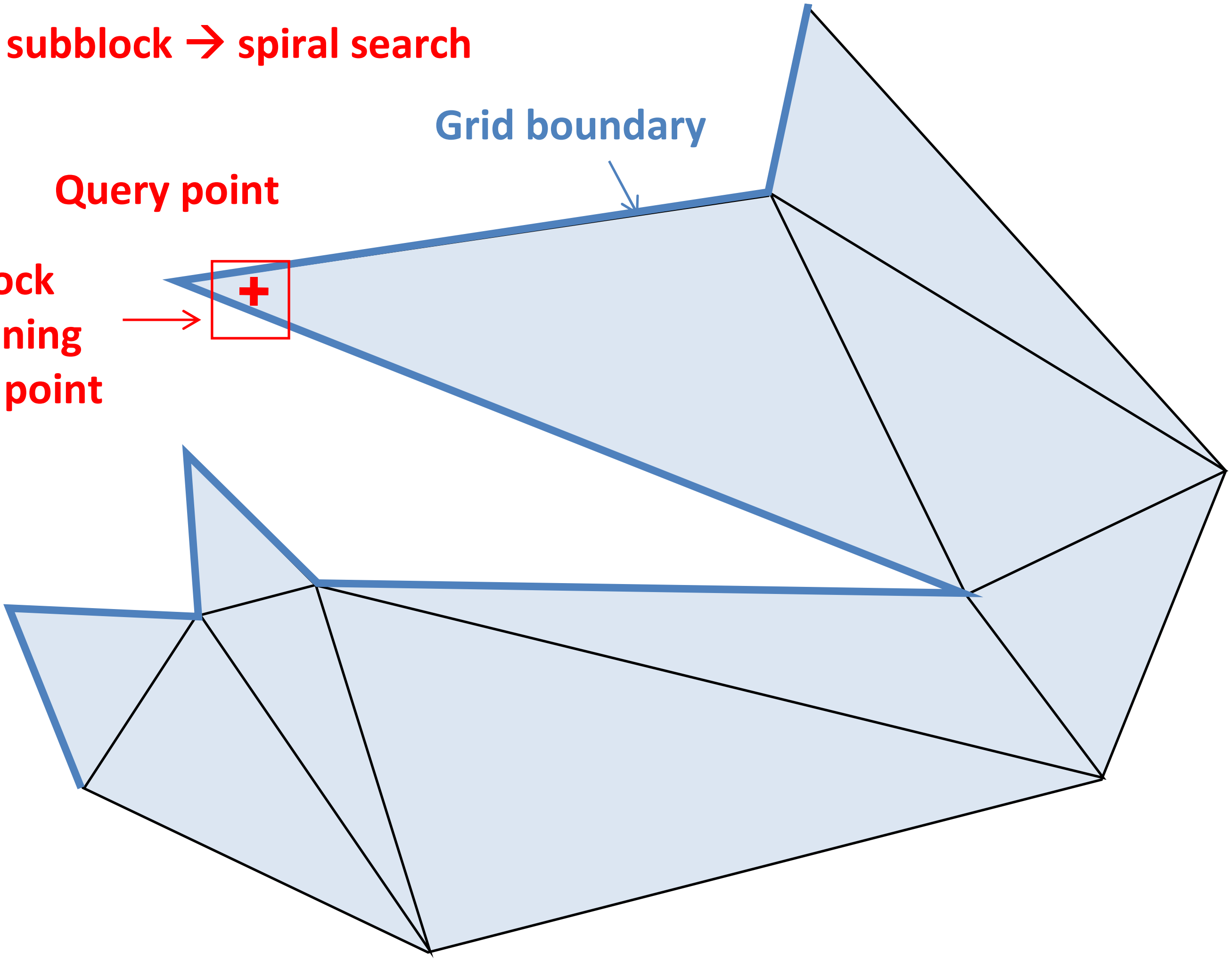
Partition boundary

Orphan point

Viscous wall boundary

Orphan point problem: example

No cell center in subblock → spiral search

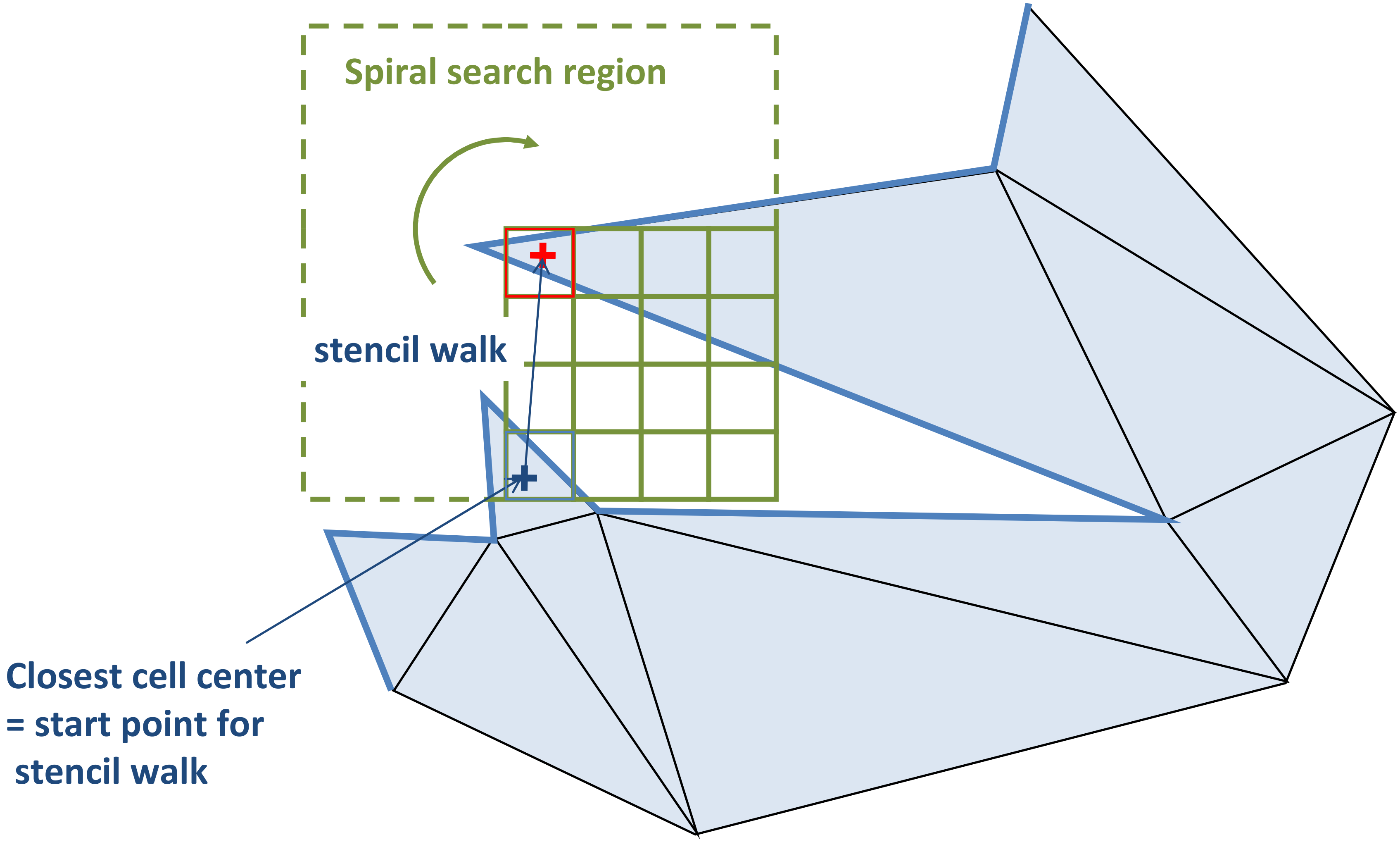


Grid boundary

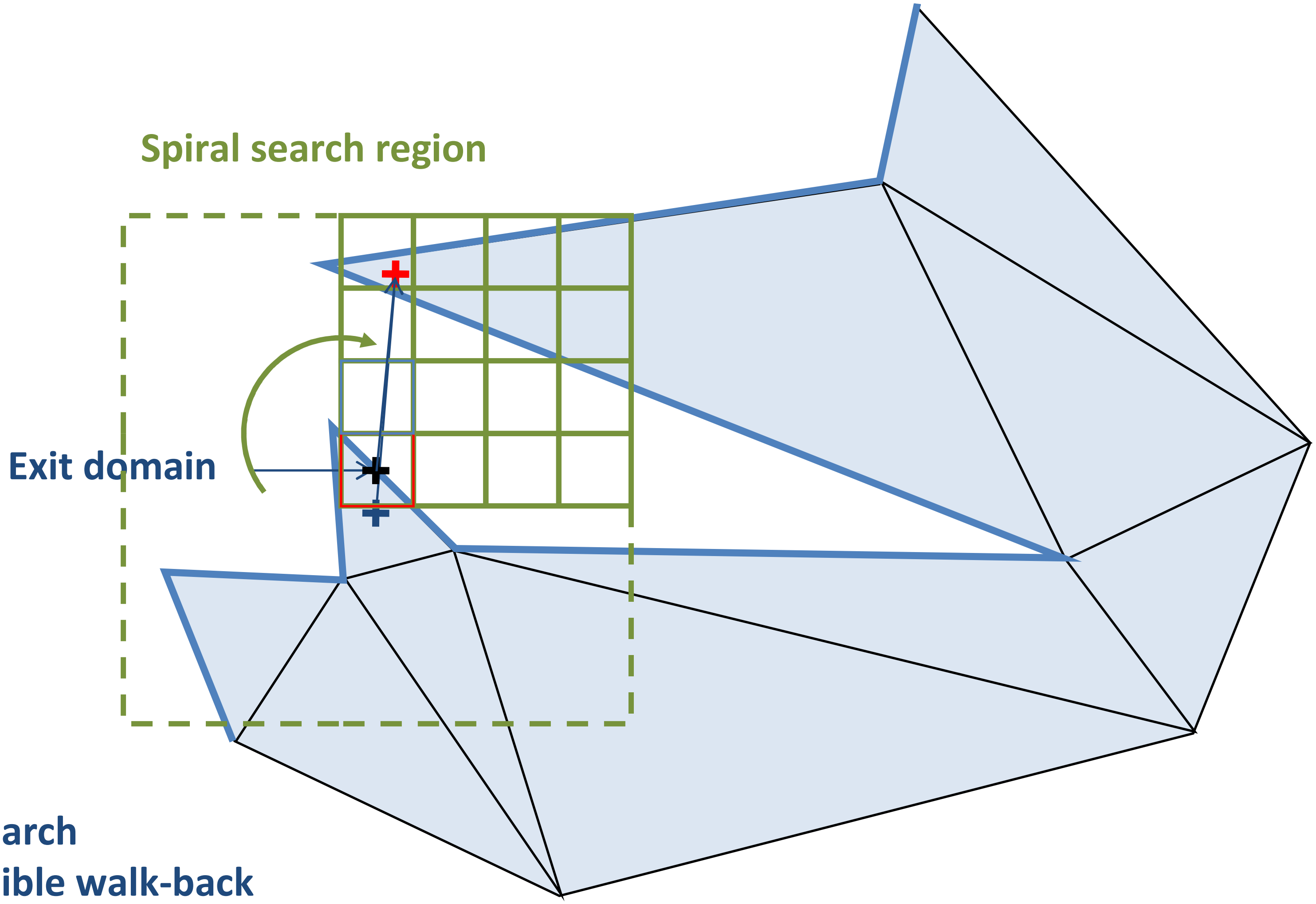
Query point

Subblock containing query point

Orphan point problem: example



Orphan point problem: example

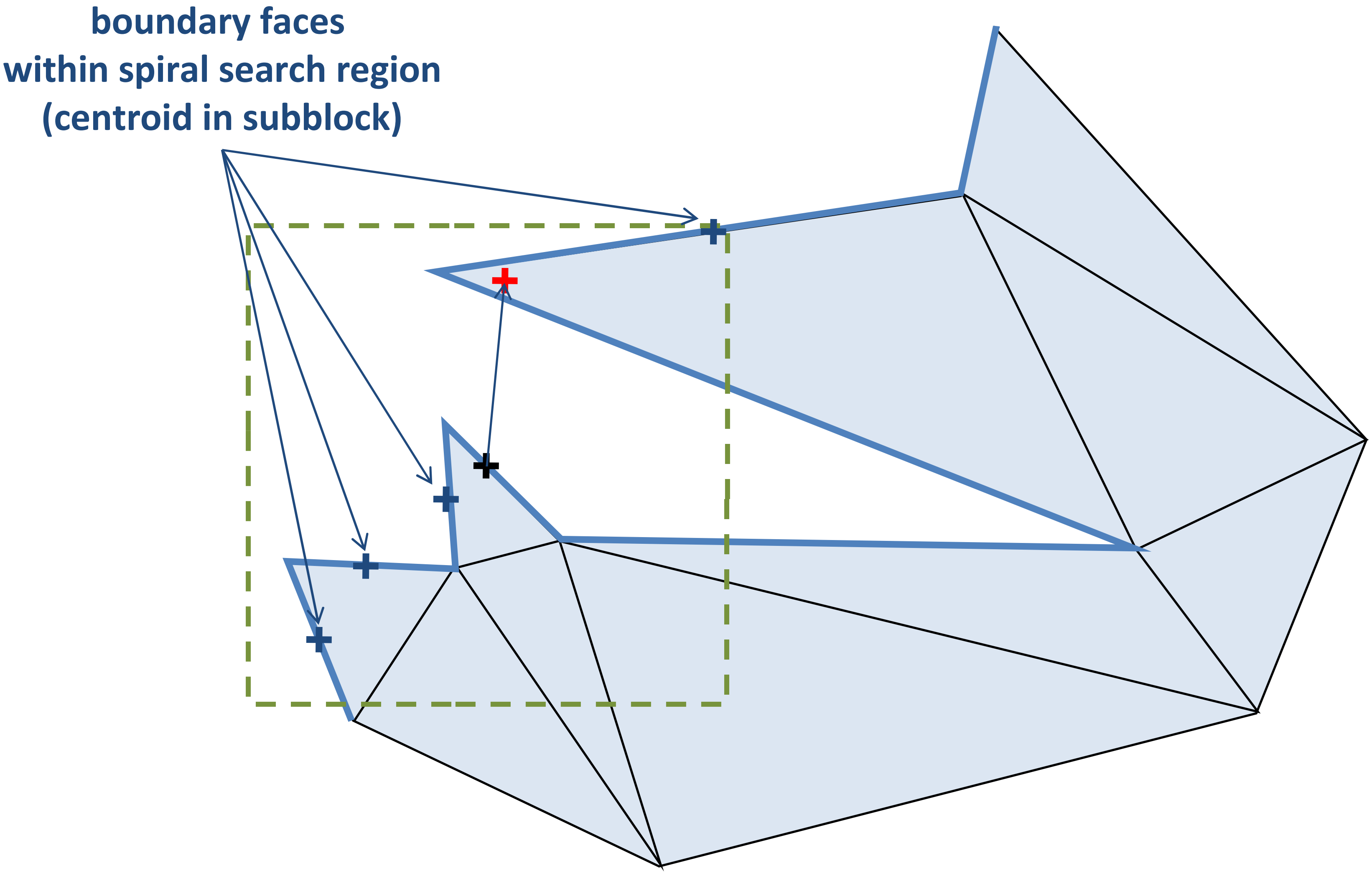


Spiral search region

Exit domain

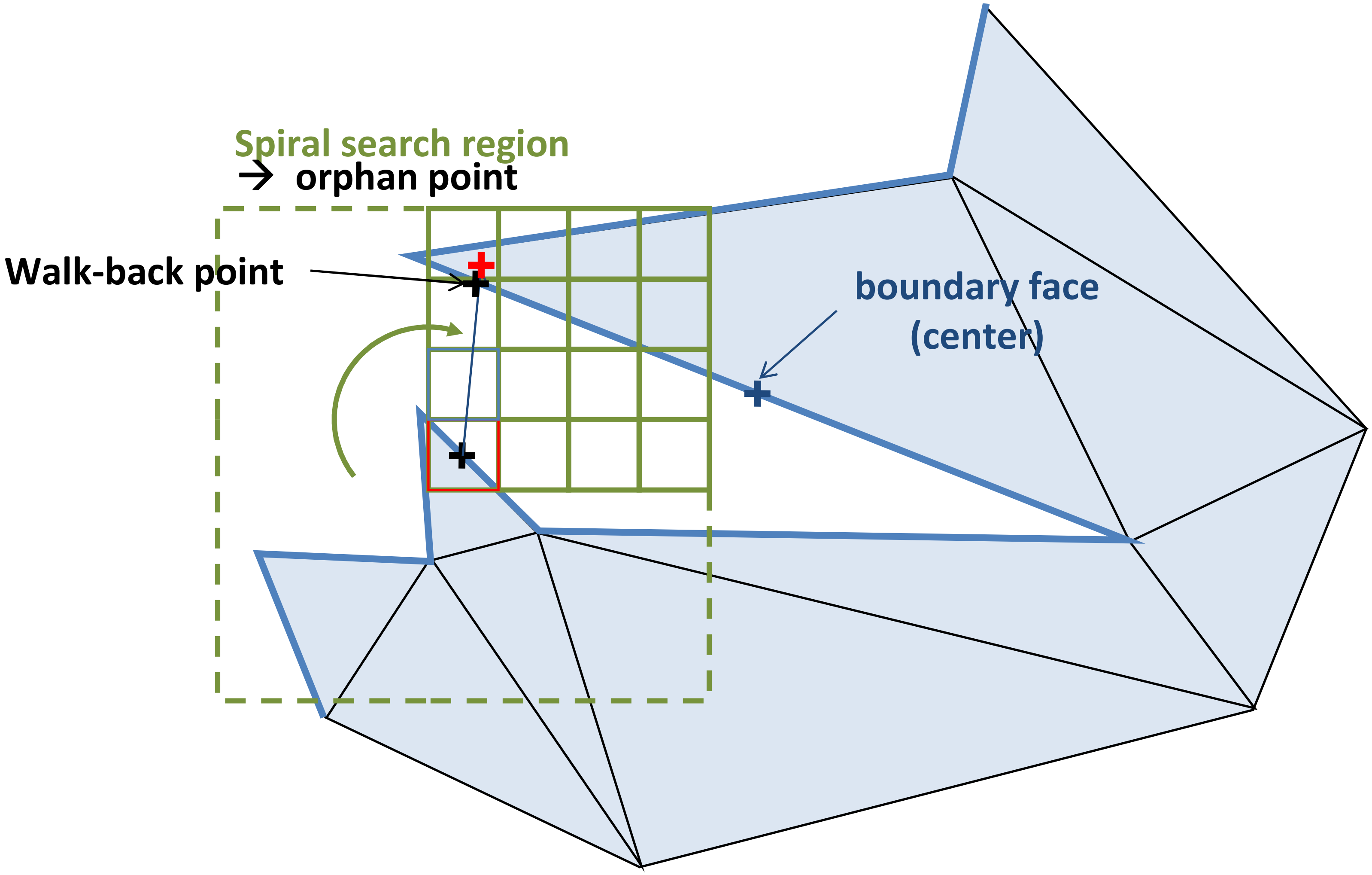
→Spiral search
for possible walk-back

Orphan point problem: example



boundary faces
within spiral search region
(centroid in subblock)

Orphan point problem: example



→ Walk-back point is missed because boundary face (center) is outside of spiral search region

Alternating Digital Tree (ADT)

- Realized that we need a more robust search algorithm

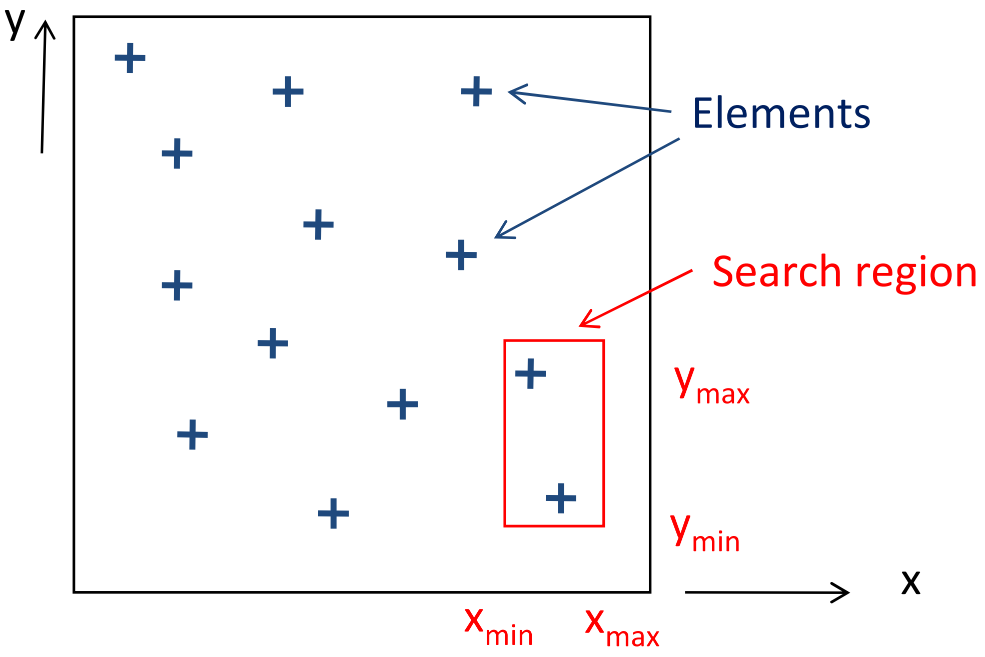
Standard ADT search problem in 2D:

Given a list of elements (in 2D, elements=2D points) : points $\left\{ \begin{matrix} x_1 & x_2 & \ldots & x_N \\ y_1 & y_2 & \ldots & y_N \end{matrix} \right\}$

determine all elements inside a search region : limits $\left\{ \begin{matrix} x_{min} & x_{max} \\ y_{min} & y_{max} \end{matrix} \right\}$

i.e. Determine all points such that $x_{min} < x_i < x_{max}$
$y_{min} < y_i < y_{max}$

Example in 2D:



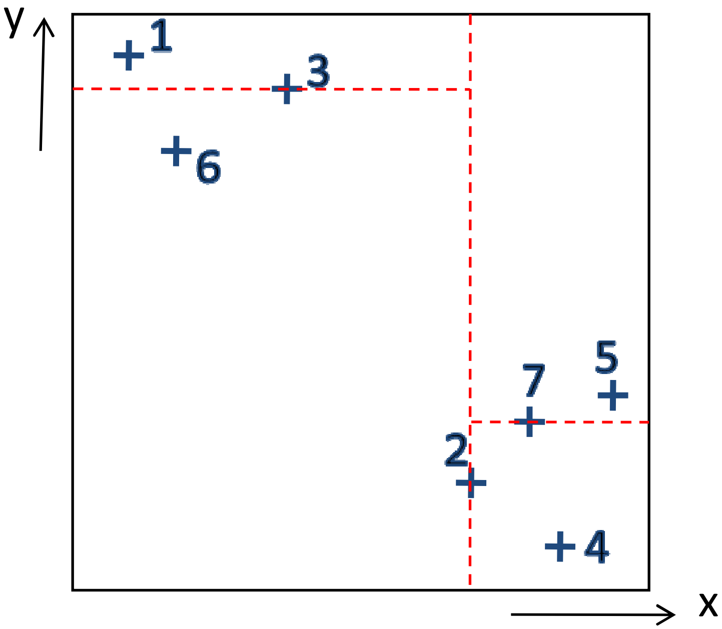
Algorithm adapted from: “An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems”
Javier Bonet, Jaime Peraire, *International Journal for Numerical Methods in Engineering,* 1991

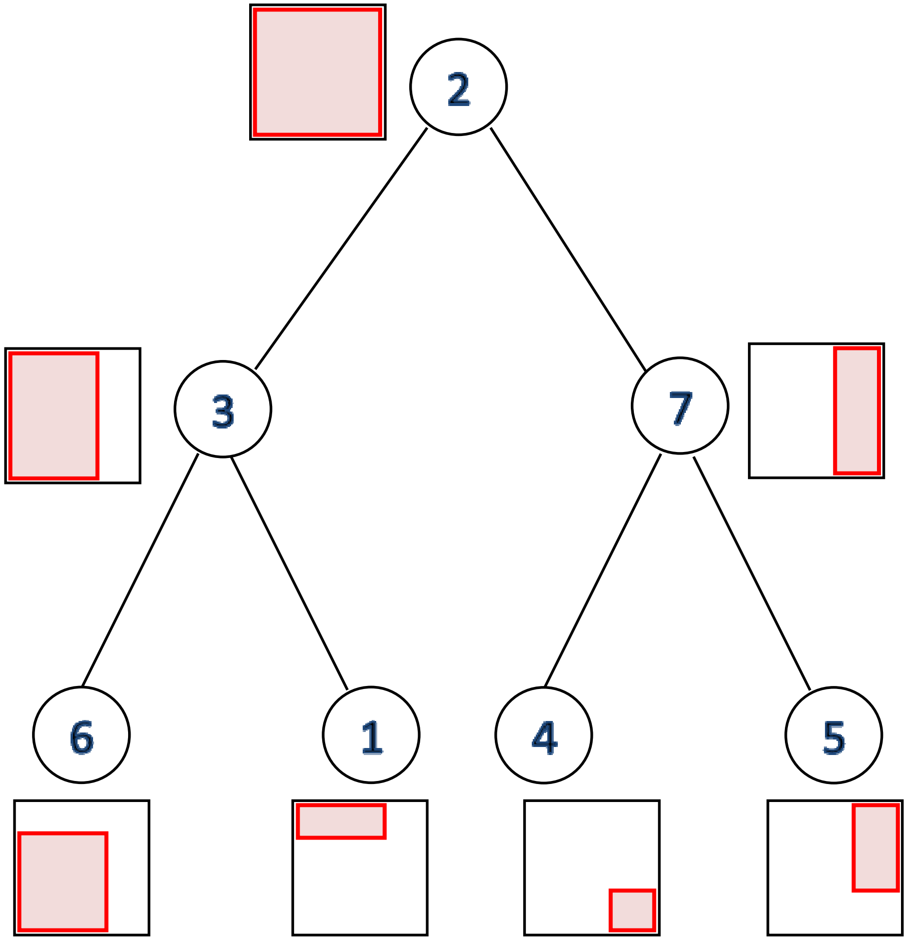
Alternating Digital Tree (ADT)

Step 1 : organize elements in a binary tree structure :

- At each tree node, divide elements into two groups according to position along a dimension (using median will result in a **balanced** tree)
- Alternate dimensions
- Each tree node is associated with an element and a region of space

Example in 2D:



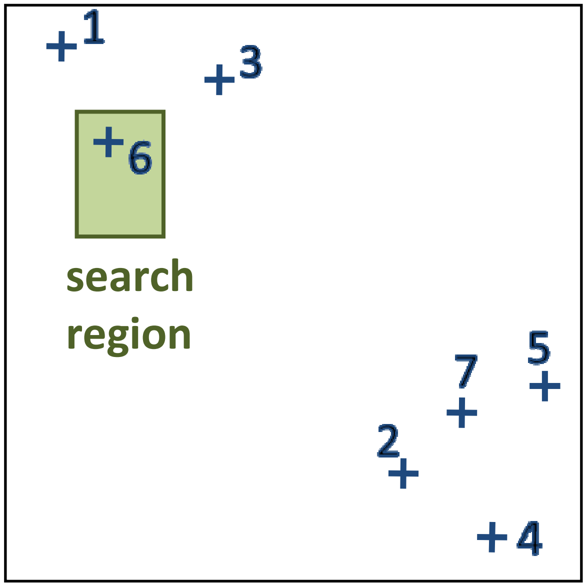


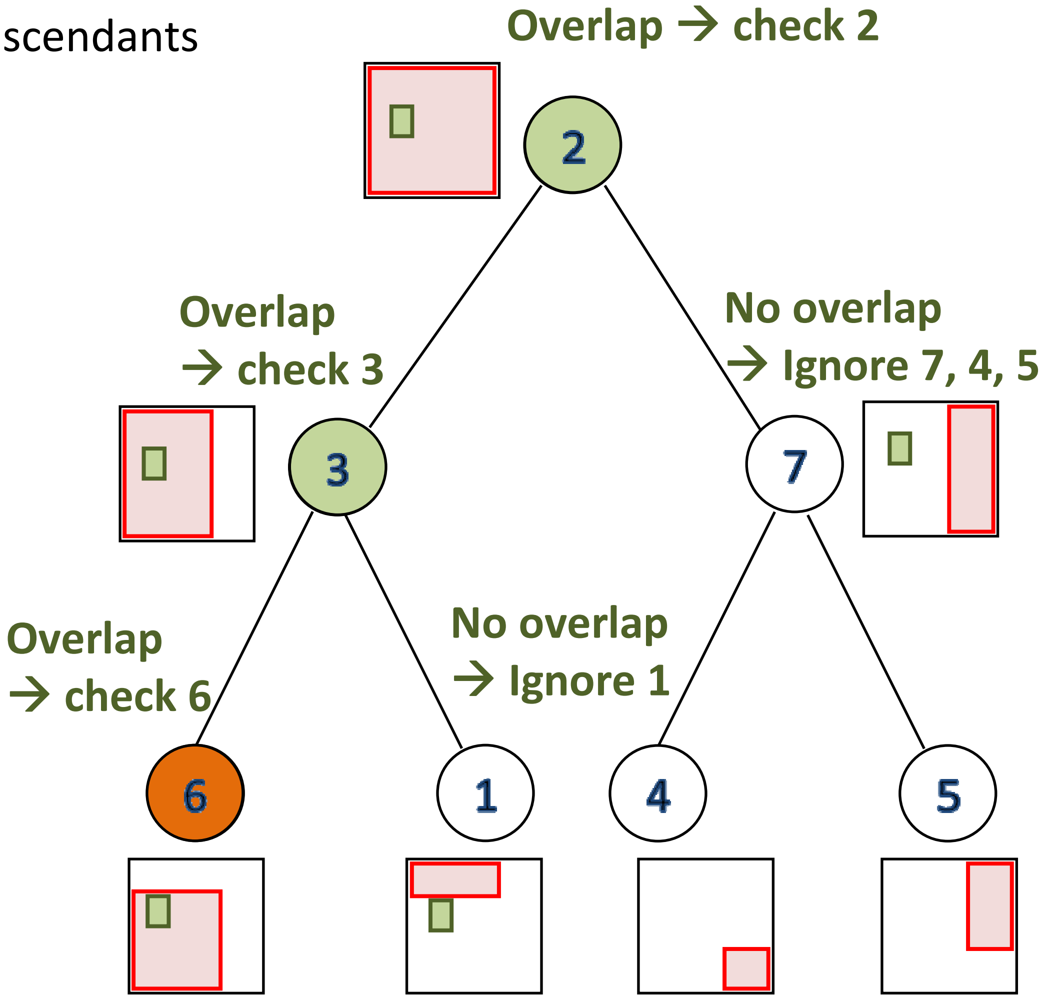
Alternating Digital Tree (ADT)

Step 2 : search the ADT for elements inside search region:

- At each tree node, check for region overlap with search region
- If overlap, check for element containment
- If no overlap, ignore all node descendants

Example in 2D:




Overlap → check 2

Overlap → check 3

No overlap → Ignore 7, 4, 5

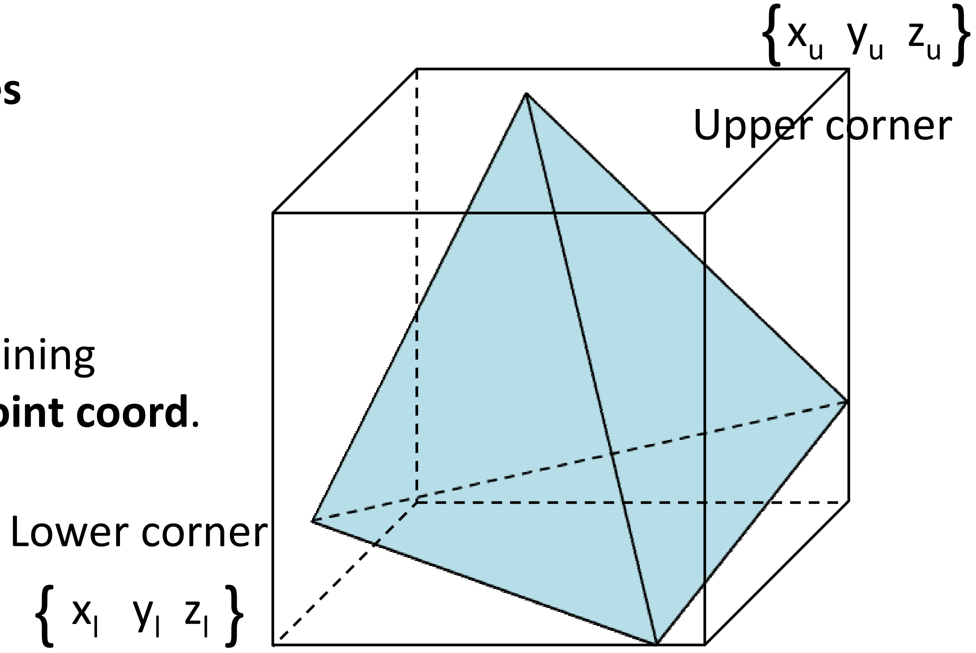
Overlap → check 6

No overlap → Ignore 1

Adapting the ADT algorithm for PUNDIT

Elements in ADT are **cell bounding boxes**
i.e., 6D elements :

$$\{ x_l \;\; y_l \;\; z_l \;\; x_u \;\; y_u \;\; z_u \}$$

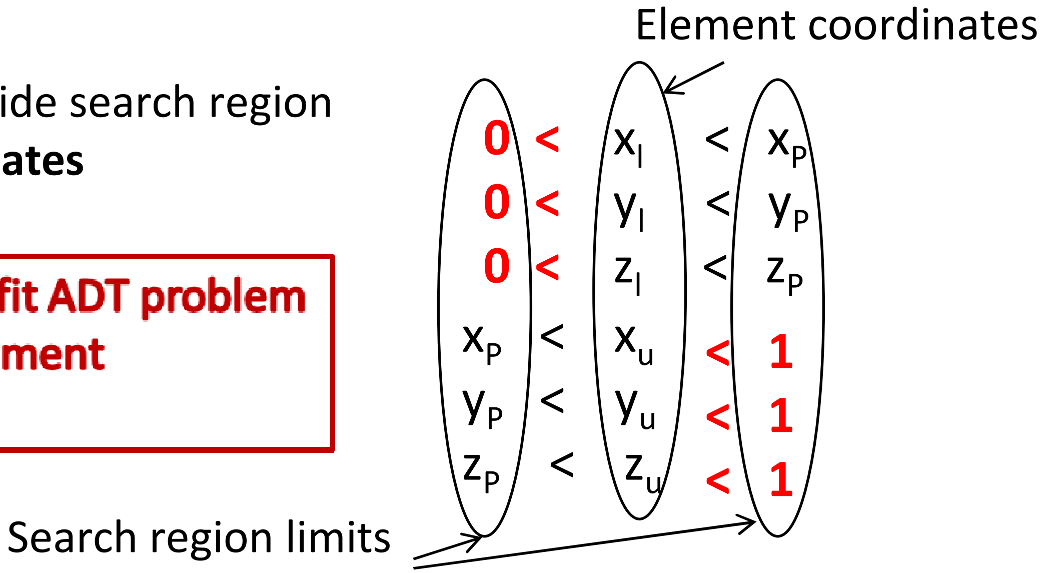


PUNDIT problem: find the element containing
Point P = $\{ x_P \;\; y_P \;\; z_P \}$ i.e. bound **search point coord**.

i.e.

$$0 < x_l < x_P < x_u < 1$$
$$0 < y_l < y_P < y_u < 1$$
$$0 < z_l < z_P < z_u < 1$$

ADT problem: find the element(s) inside search region
i.e. bound **element coordinates**



Element coordinates

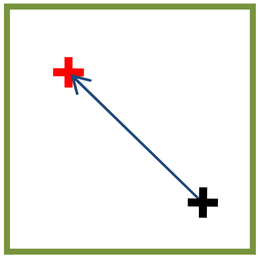
$$0 < x_l < x_P$$
$$0 < y_l < y_P$$
$$0 < z_l < z_P$$
$$x_P < x_u < 1$$
$$y_P < y_u < 1$$
$$z_P < z_u < 1$$

Search region limits

PUNDIT problem can be re-phrased to fit ADT problem by bounding the coordinates of the element (bounding box) :

Improving robustness

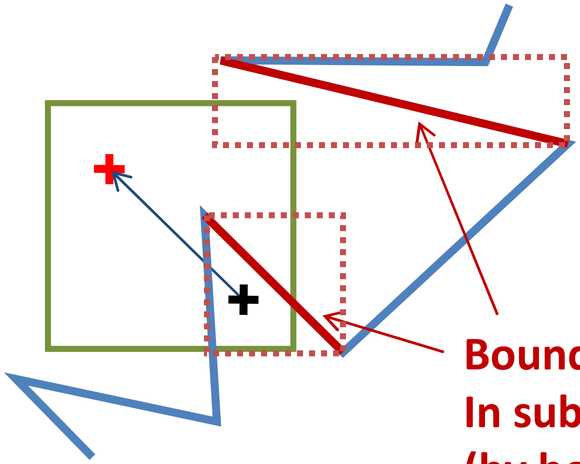
1/ Initial guess for stencil walk should always be inside target subblock :



query point

start point

2/ In case of stencil walk exiting grid boundary, check all other boundary faces Inside target subblock (**by bounding box**) :



Boundary faces
In subblock
(by bounding box)

→ **even** number of new intersections: query point is **outside** grid domain

→ **odd** number of new intersections: query point is **inside** grid domain (a donor cell can be identified)

Improving robustness

Advantages of improved method (Exact Inverse Map)

1/ removes need for time-consuming spiral search

2/ query points in subblocks with no true intersection with grid domain can be identified
immediately as field points

3/ Robust: no orphan point should be generated

Cost:
Need to identify a cell point inside each subblock to serve as stencil walk initial guess: increased preprocessing time

Cell point search

Problem = Identifying a cell point inside each **subblock without cell center**
(to serve as stencil walk initial guess)

This is done by geometric considerations:

if such a point exists, one of the following must be true:

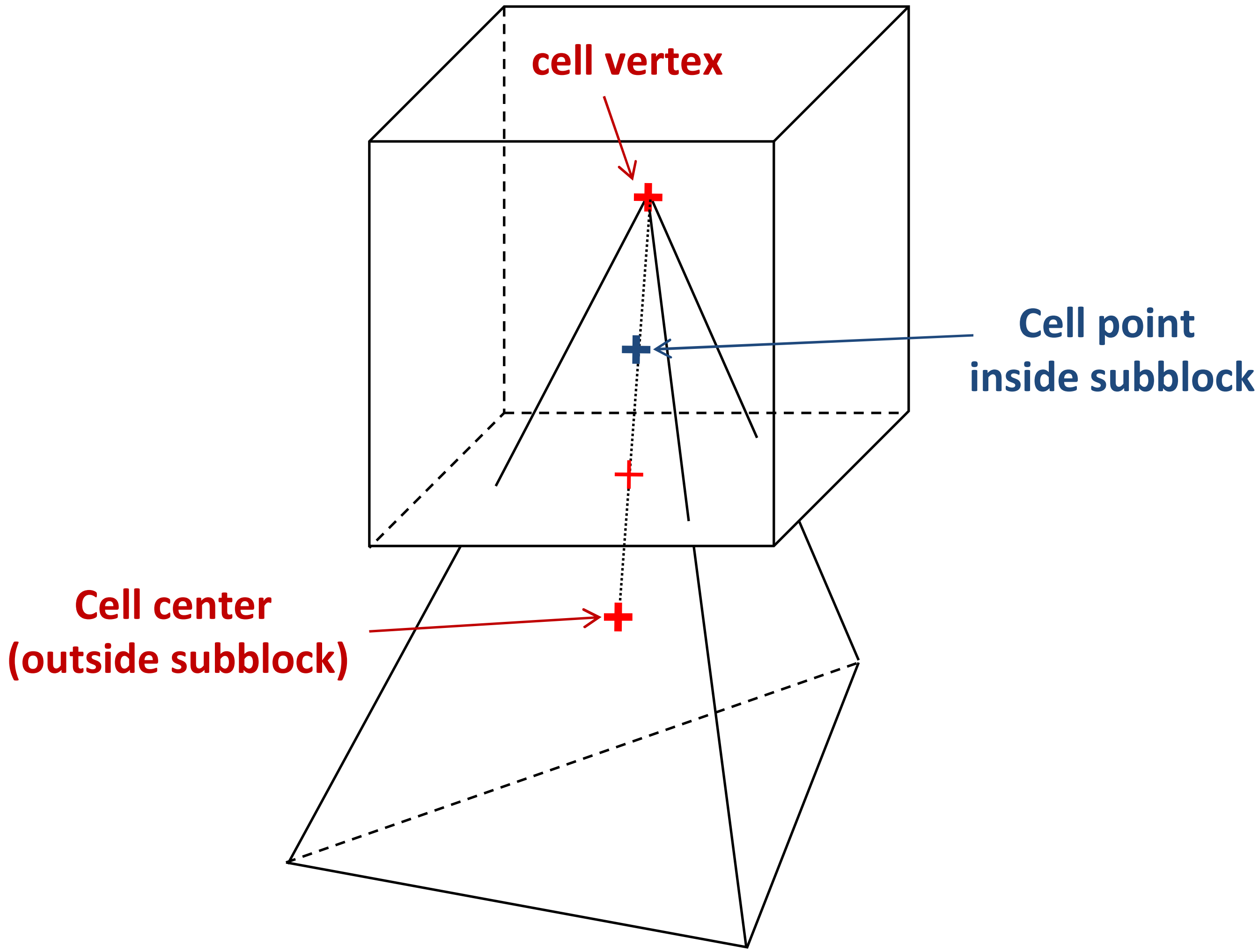
a cell vertex is inside the subblock (case 1)
a cell edge intersects a face of the subblock (case 2)
a cell face intersects an edge of the subblock (case 3)
a cell contains the subblock entirely (case 4)

Otherwise, no such point exists and the subblock has no true intersection with the grid domain (case 5)

For each subblock without cell center, cells with bounding box intersection are identified. The unique list of vertices, edges, and faces they are composed of is then extracted. Since cases 1 to 5 are increasingly time-consuming to identify, they are checked in this order.

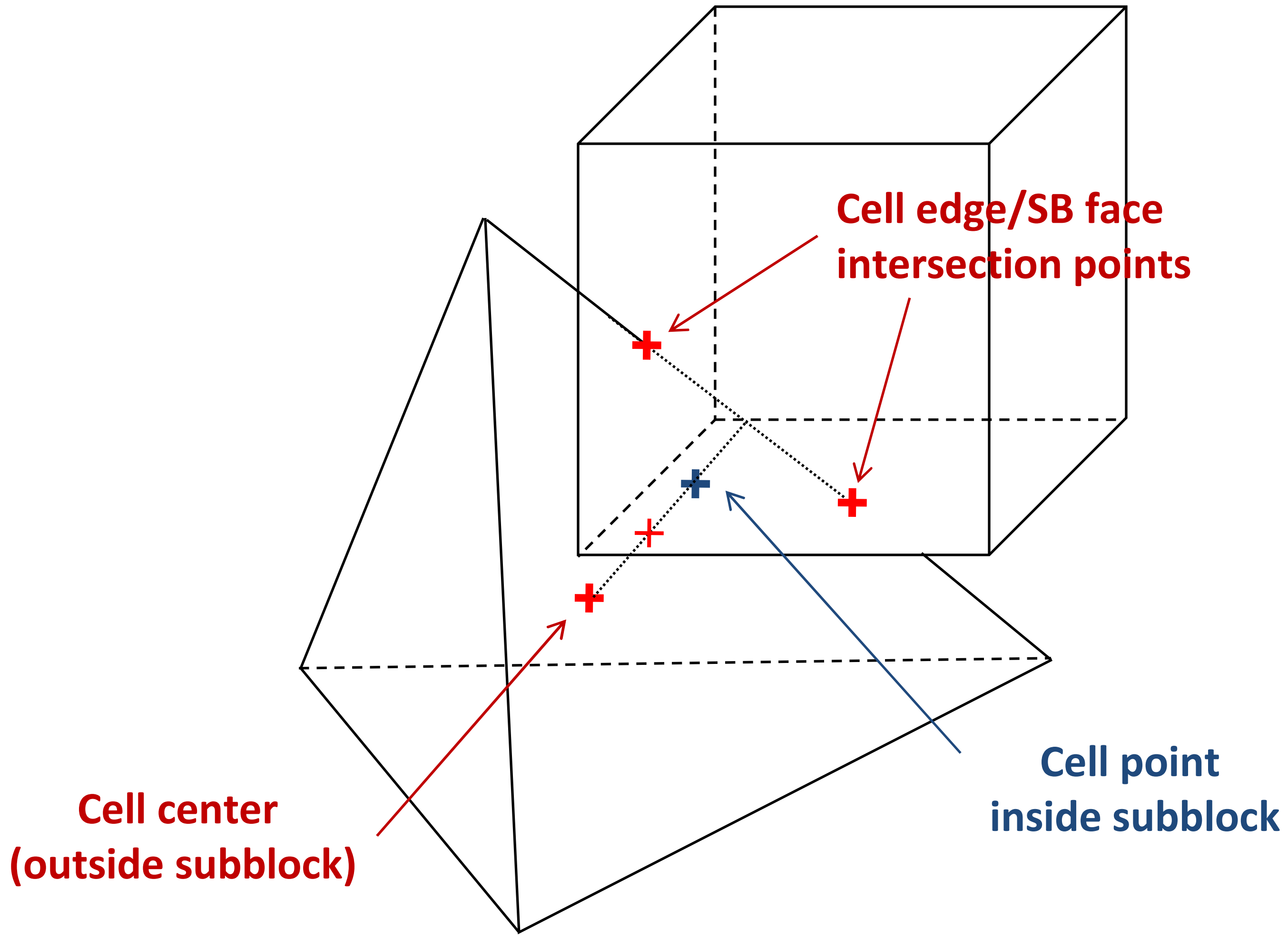
Cell point search: case 1

case 1: cell vertex inside subblock



Cell point search: case 2

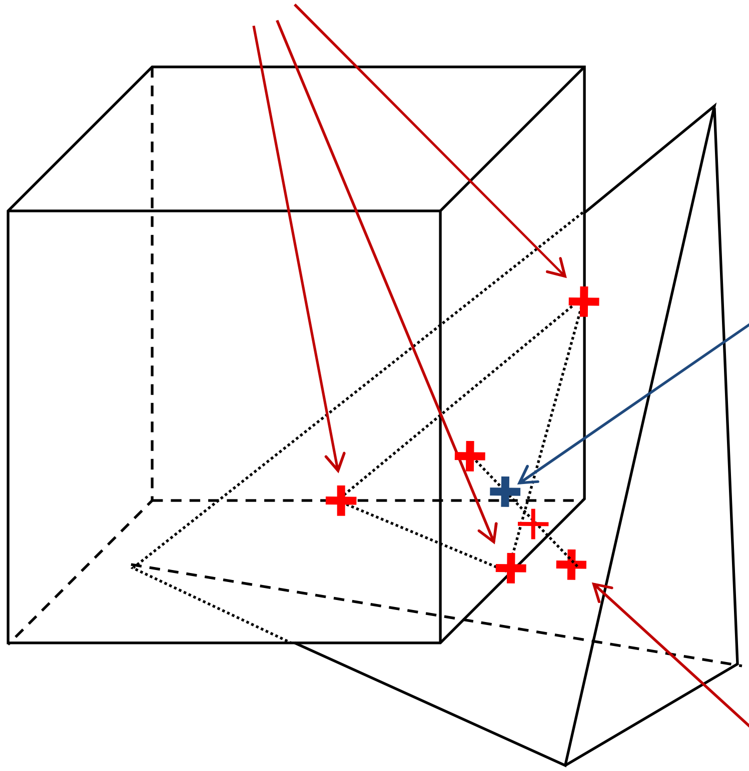
case 2: cell edge intersects subblock face



Cell point search: case 3

case 3: cell face intersects subblock edge

Cell face/ SB edge intersection points

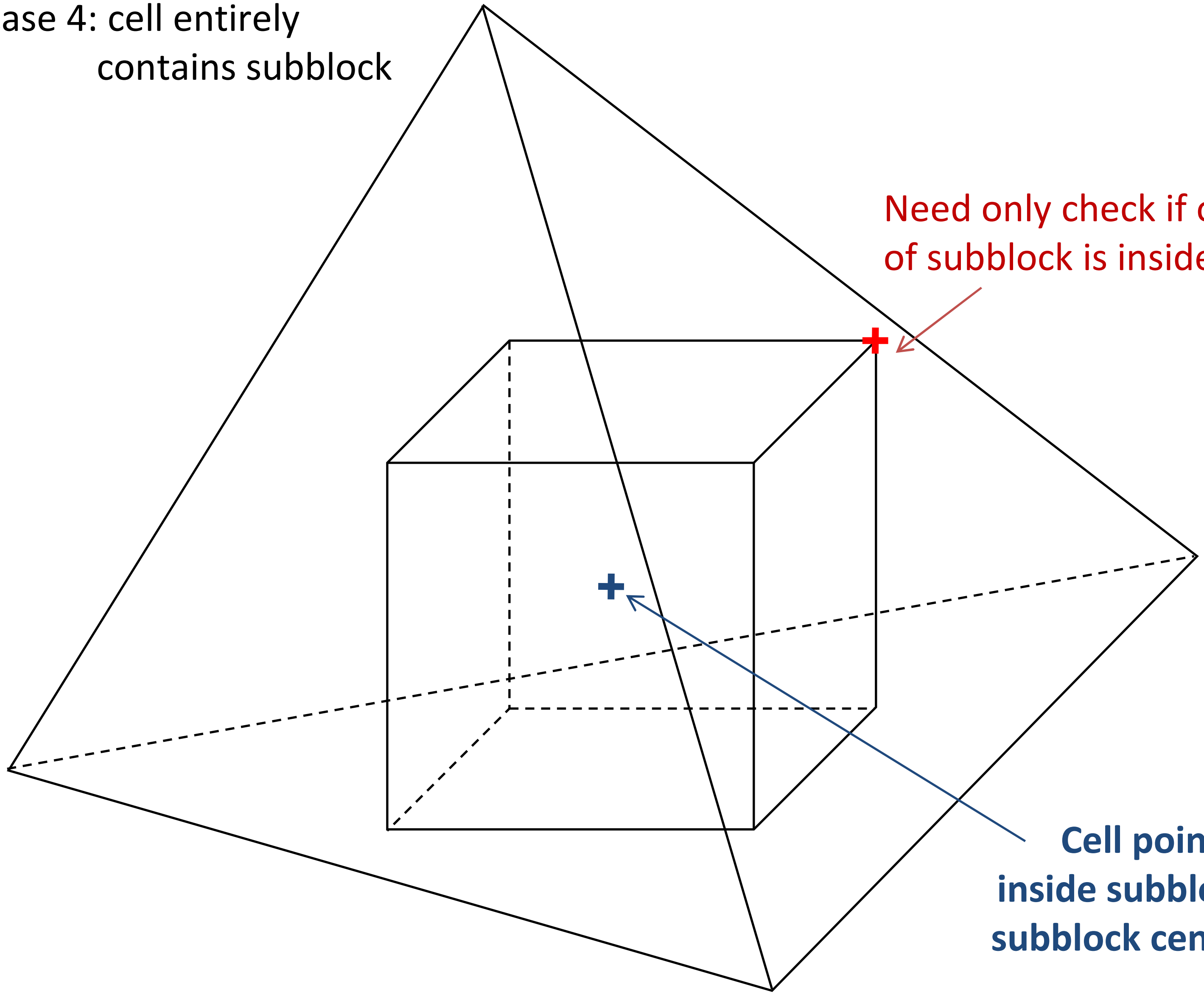


Cell point inside subblock

Cell center (outside subblock)

Cell point search: case 4

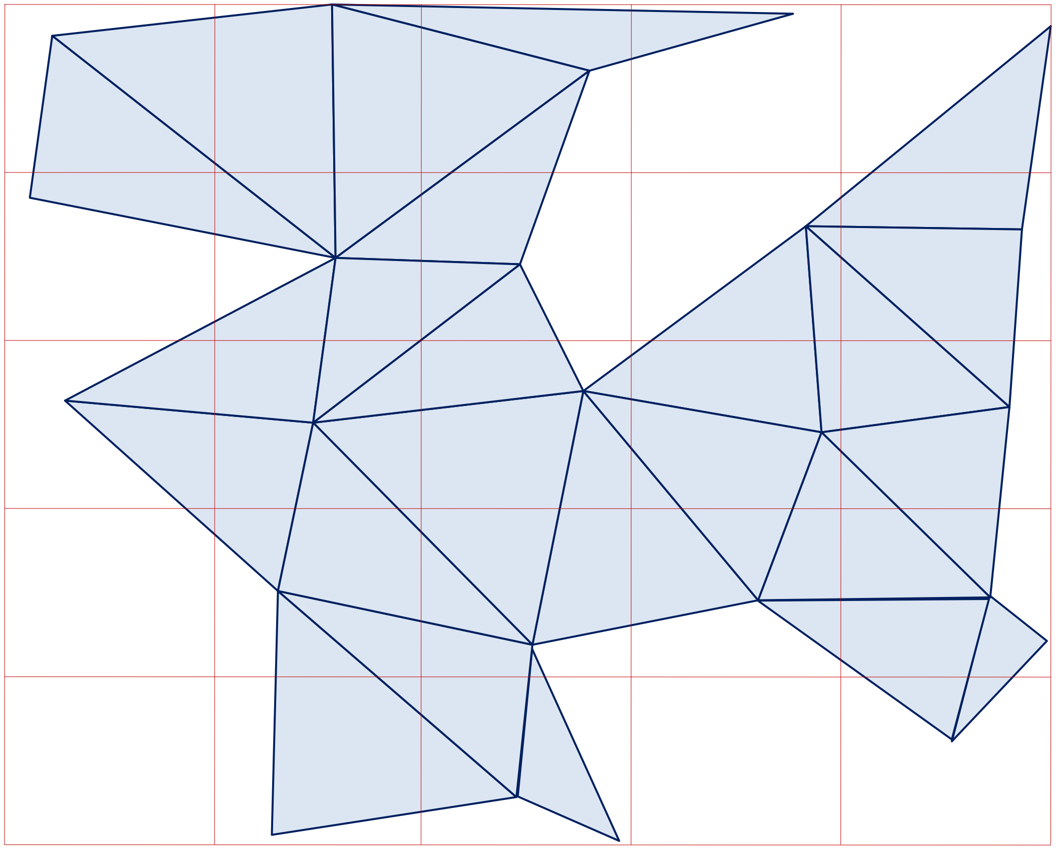
case 4: cell entirely
contains subblock



Need only check if one vertex
of subblock is inside cell

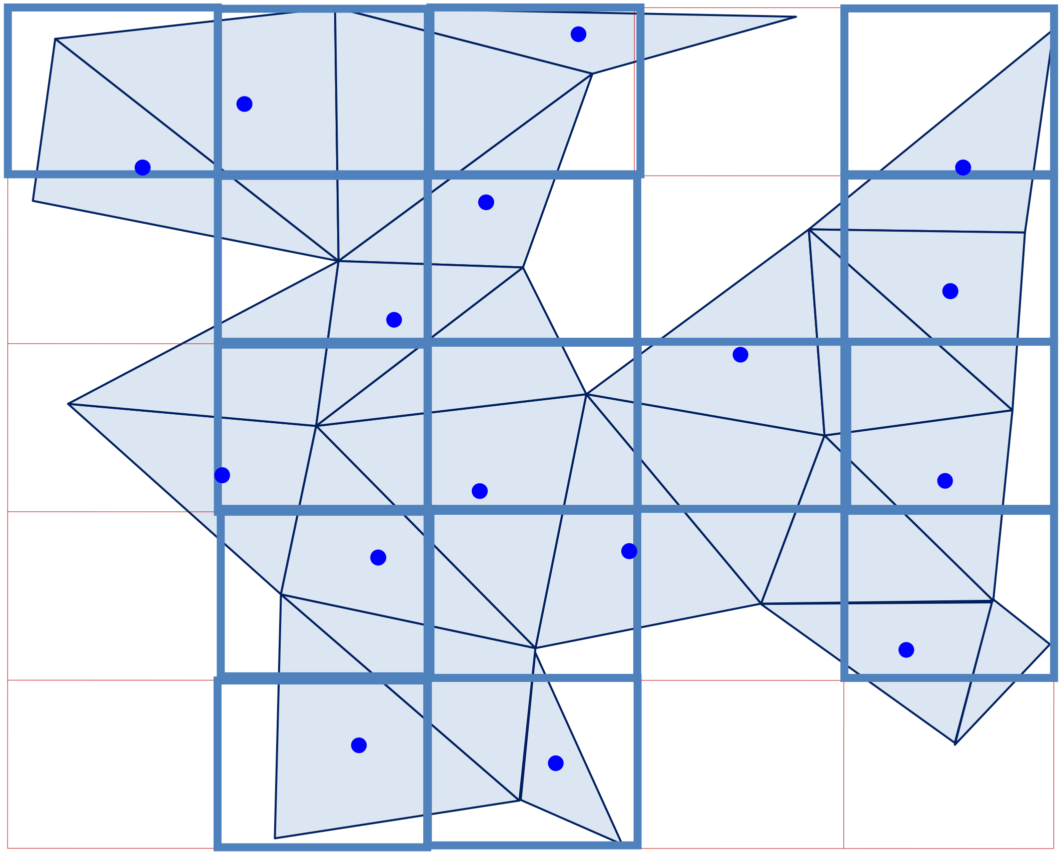
**Cell point
inside subblock =
subblock centroid**

PUNDIT problem: example in 2D



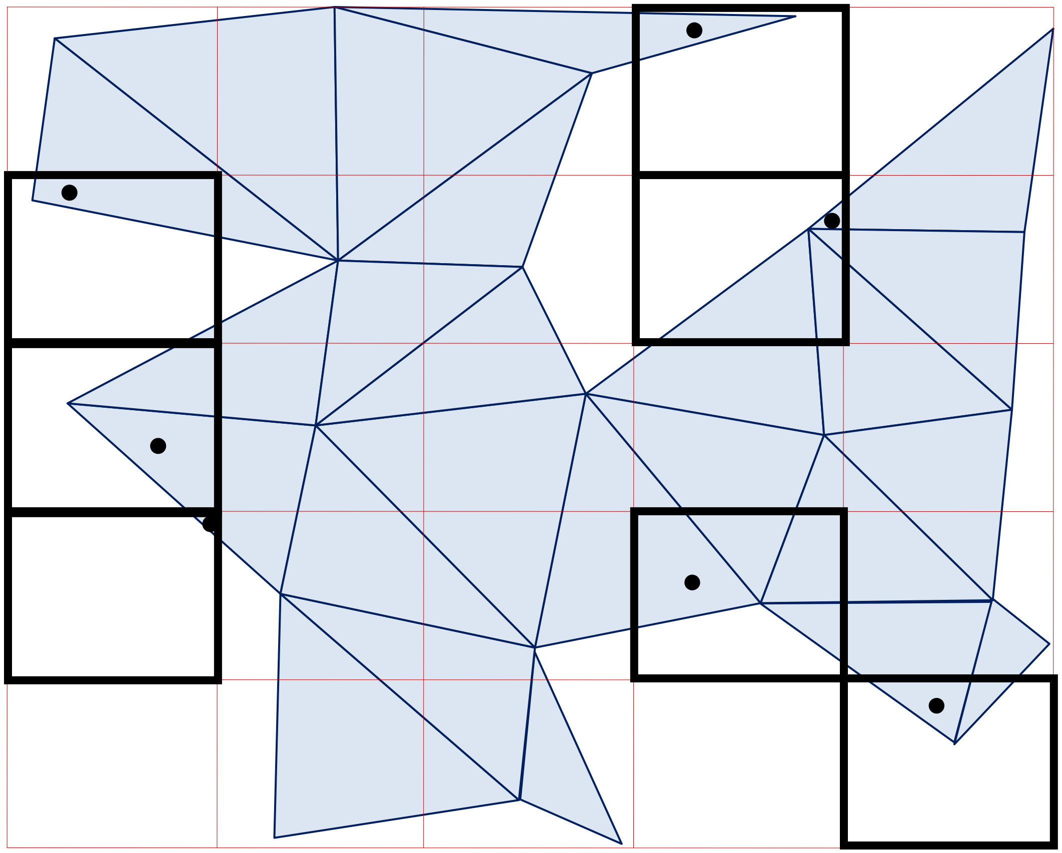
25 subblocks

PUNDIT problem: example in 2D



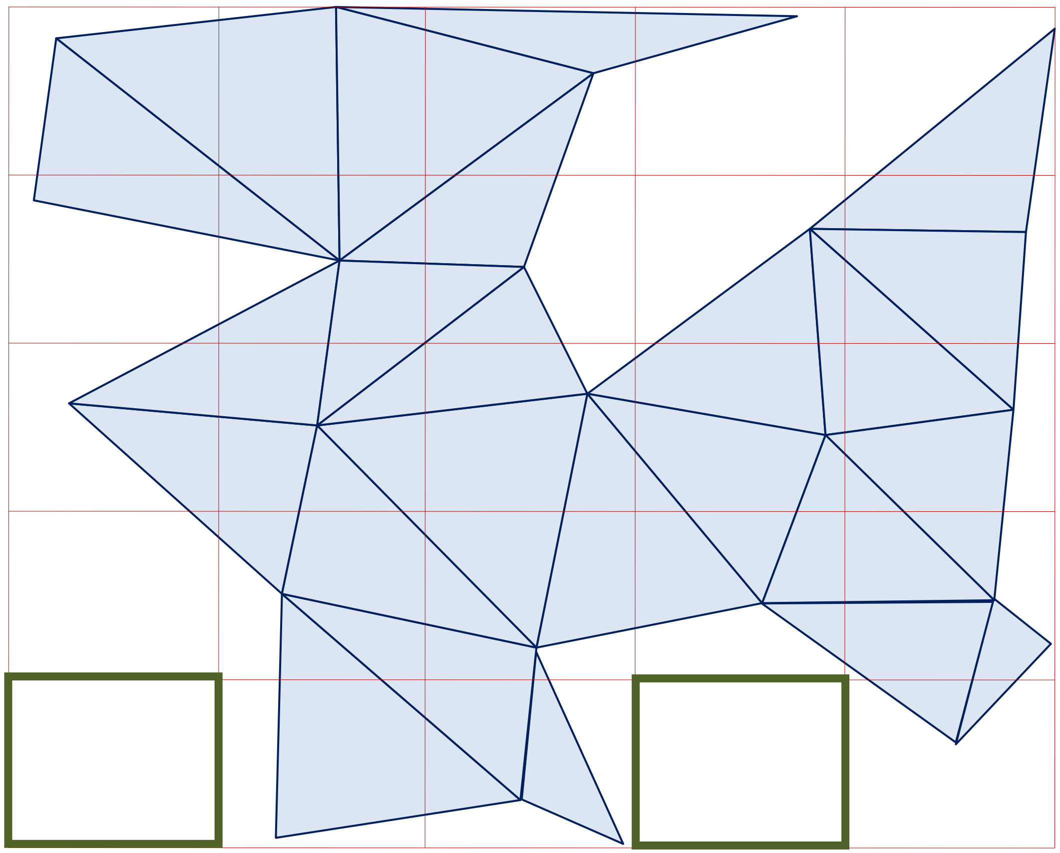
25 subblocks ------ 16 subblocks with cell center

PUNDIT problem: example in 2D



25 subblocks ------ 16 subblocks with cell center
7 subblocks with cell point

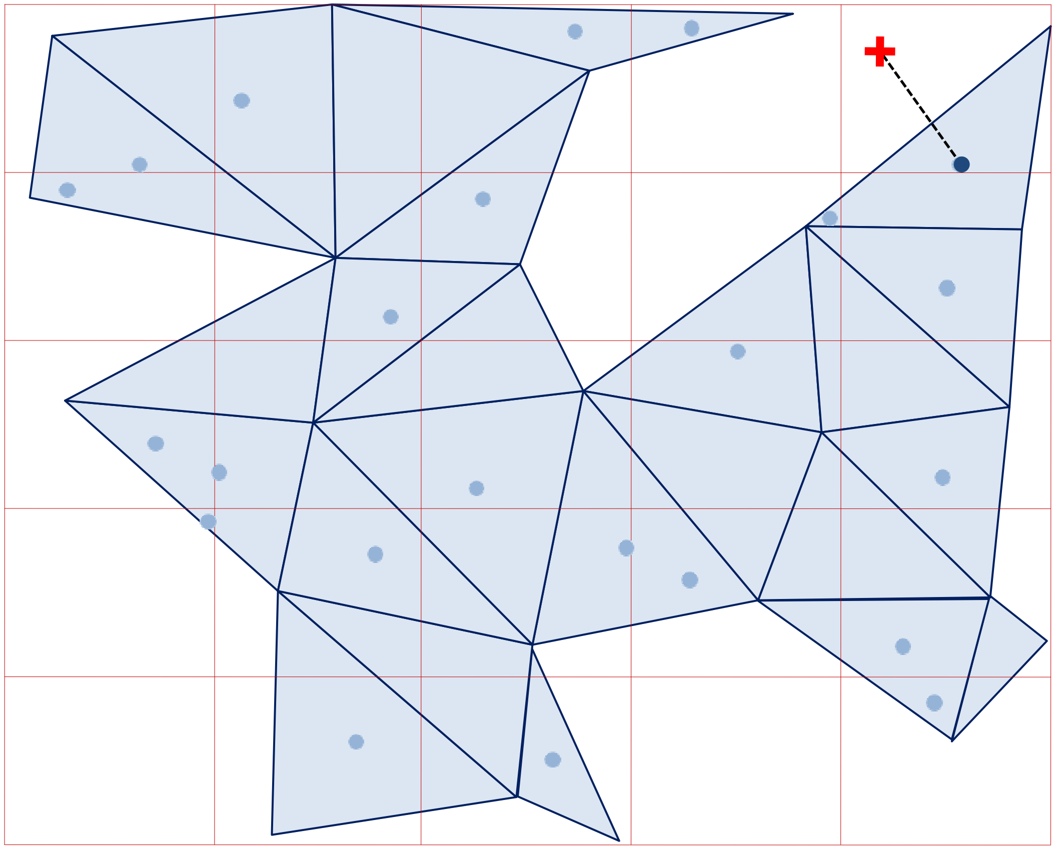
PUNDIT problem: example in 2D



25 subblocks ------ 16 subblocks with cell center
7 subblocks with cell point
2 subblocks with no grid intersection

Search example in 2D

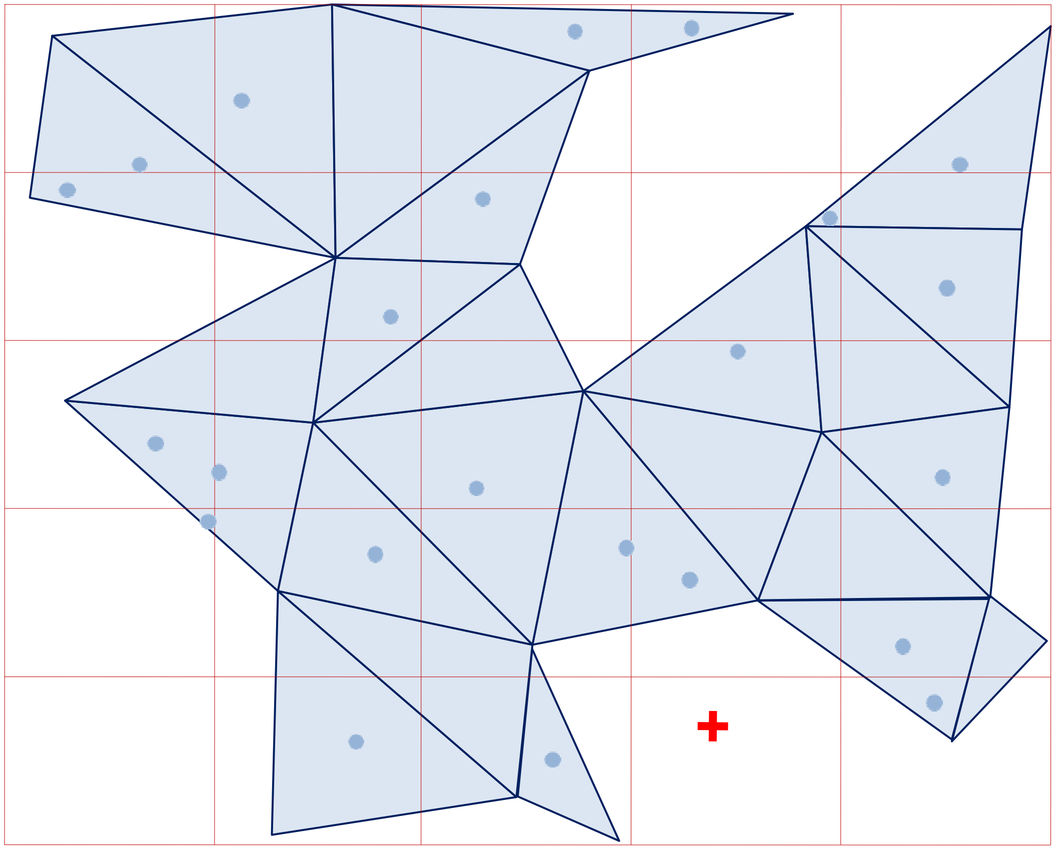
query point



Initial guess

1 boundary intersection → query point outside domain

Search example in 2D



query point outside domain

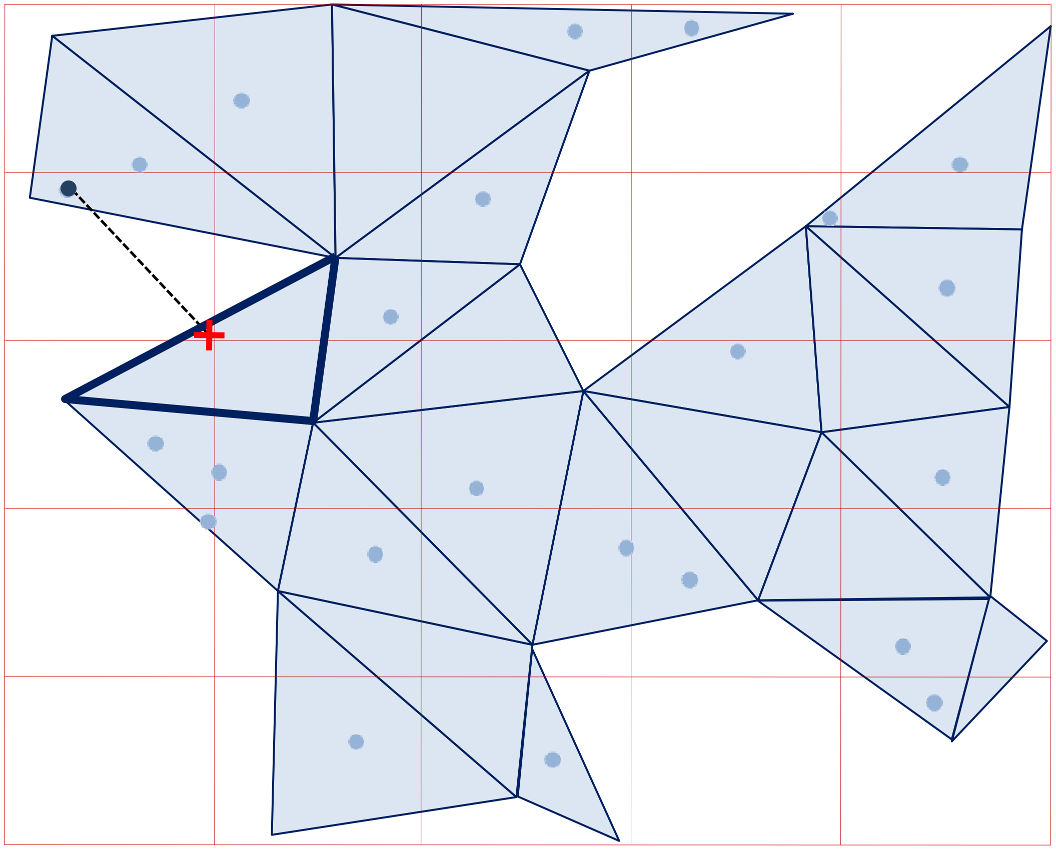
Search example in 2D

UNIVERSITY OF WYOMING

Initial guess

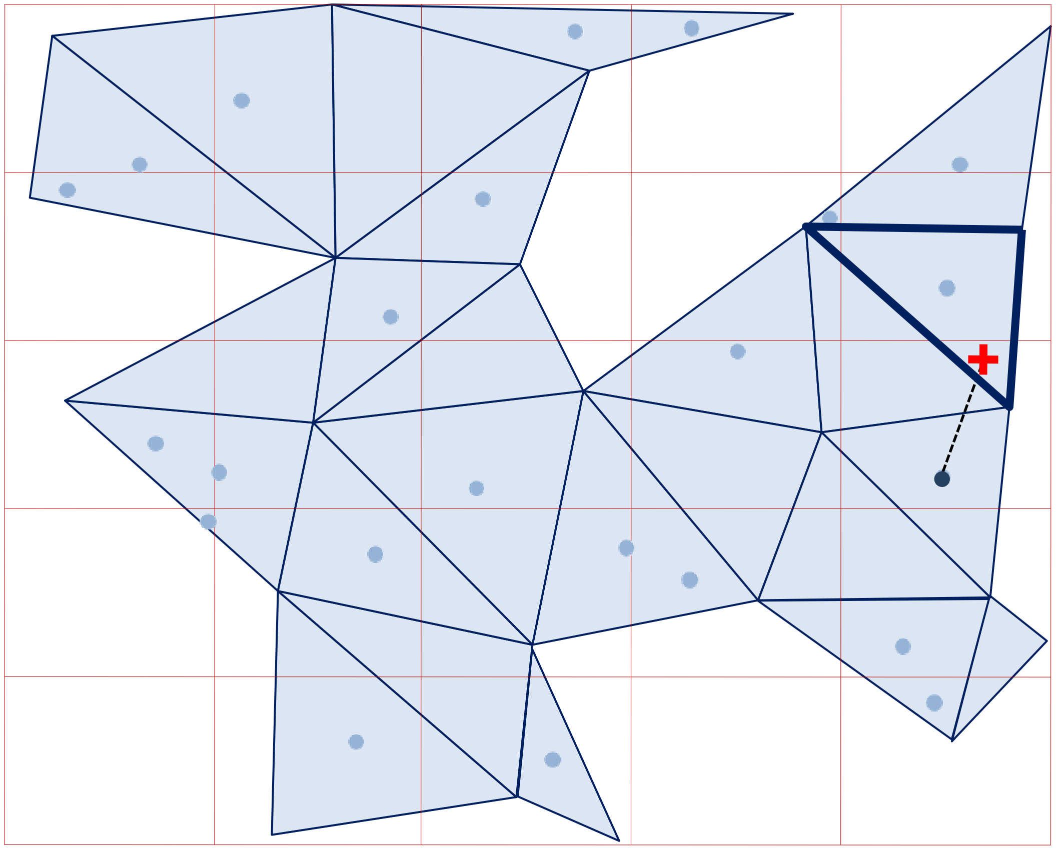
query point

donor cell



2 boundary intersections → query point inside domain

Search example in 2D

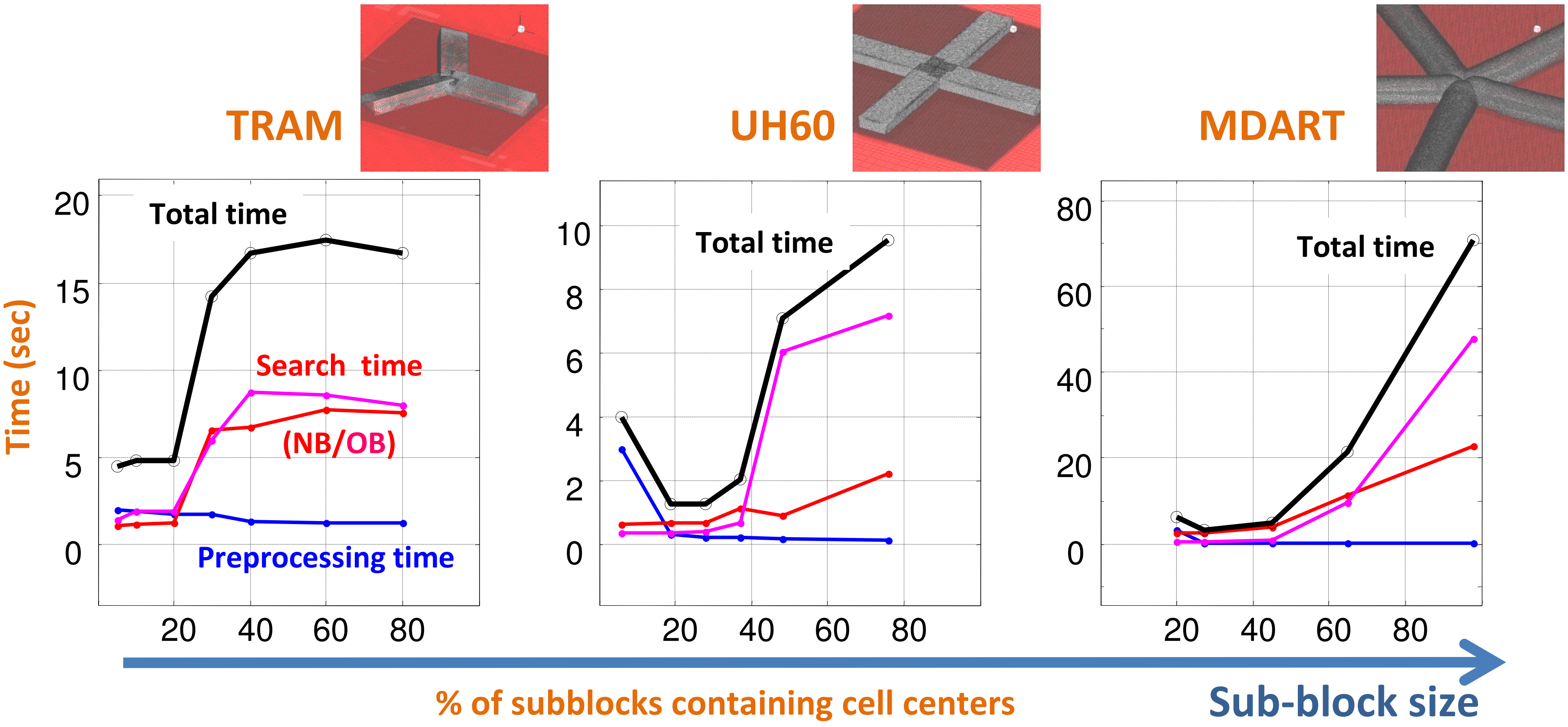


donor cell

query point

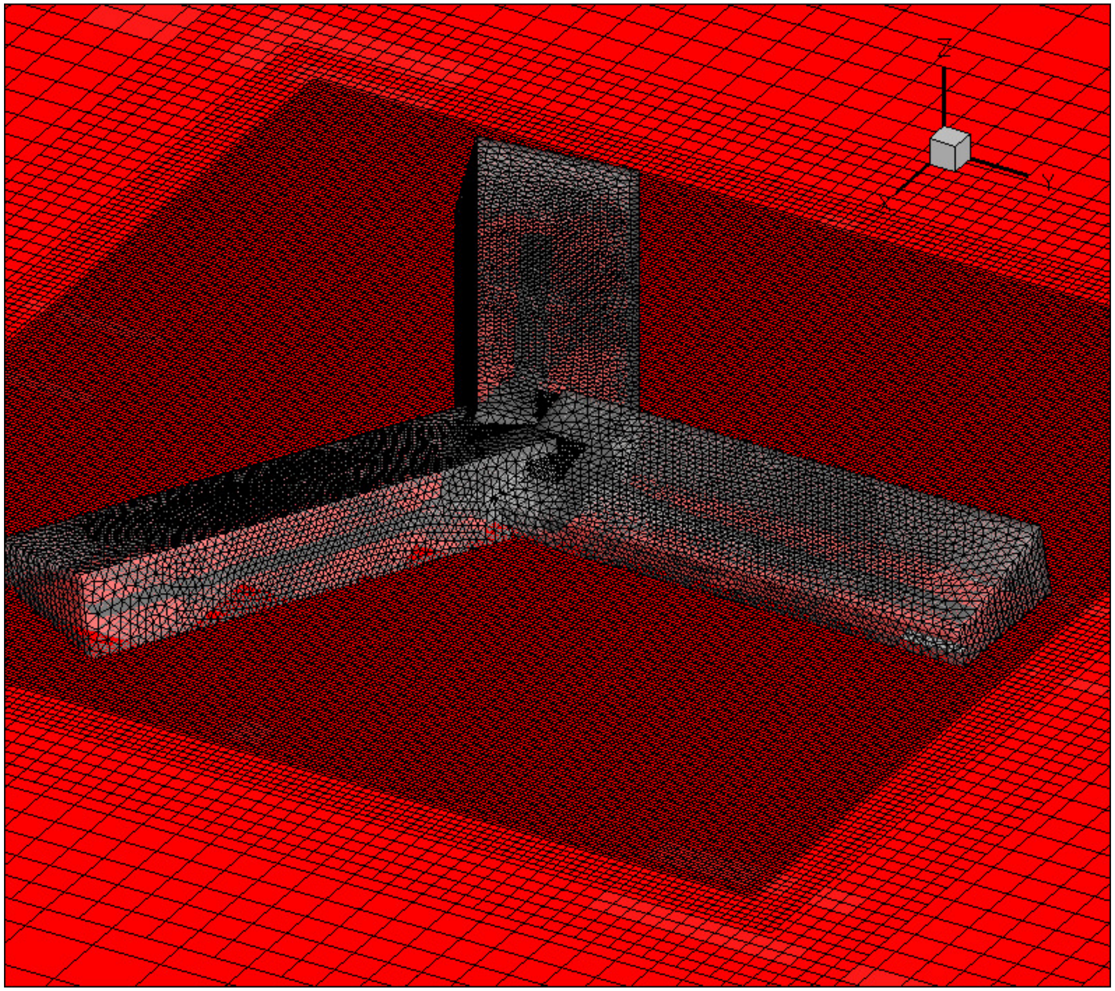
Initial guess

Optimal Subblock Size ?



- **Pre-processing time decreases** as sub-block size increases
  (less sub-blocks without cell center)
- **Search time increases** as sub-block size increases
  (more cells to search among)
- **Optimal subblock size** observed to be such that
  about 20% subblocks contain cell centers (**20% cell containment**)

Results: TRAM case



Near-Body cells	0.85 Millions
Off-Body cells	17.33 Millions
Nb of Processors	16

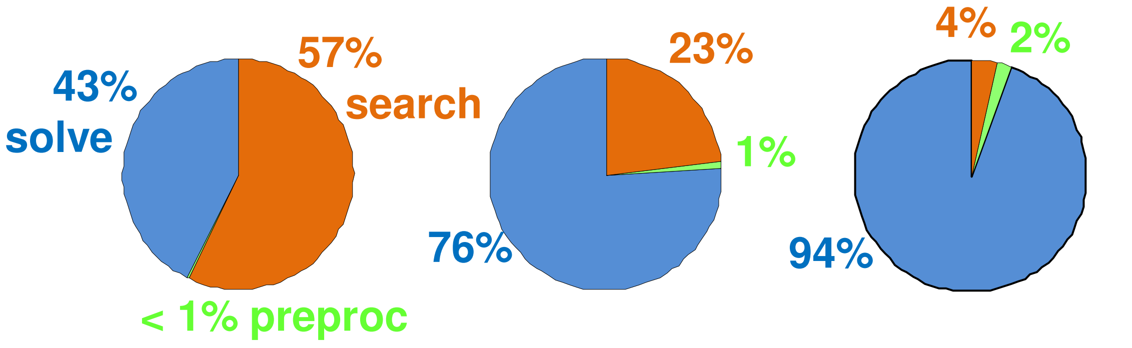
Results: TRAM case

ACCURACY

	RECEPTORS	ORPHANS
ADT method	223326	0
approx. Inverse Map method	222827 (-499)	0
exact Inverse Map method	223326 (-0)	0

SPEED

Task share
(% of total time per time step)

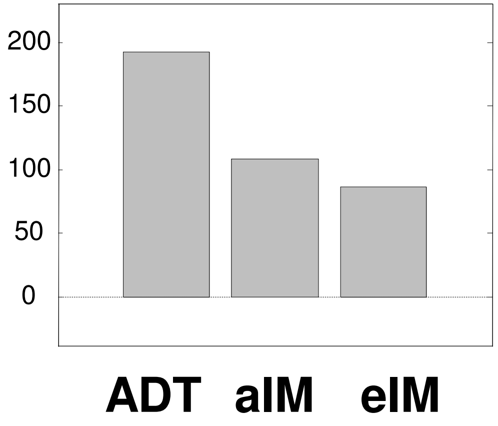


ADT

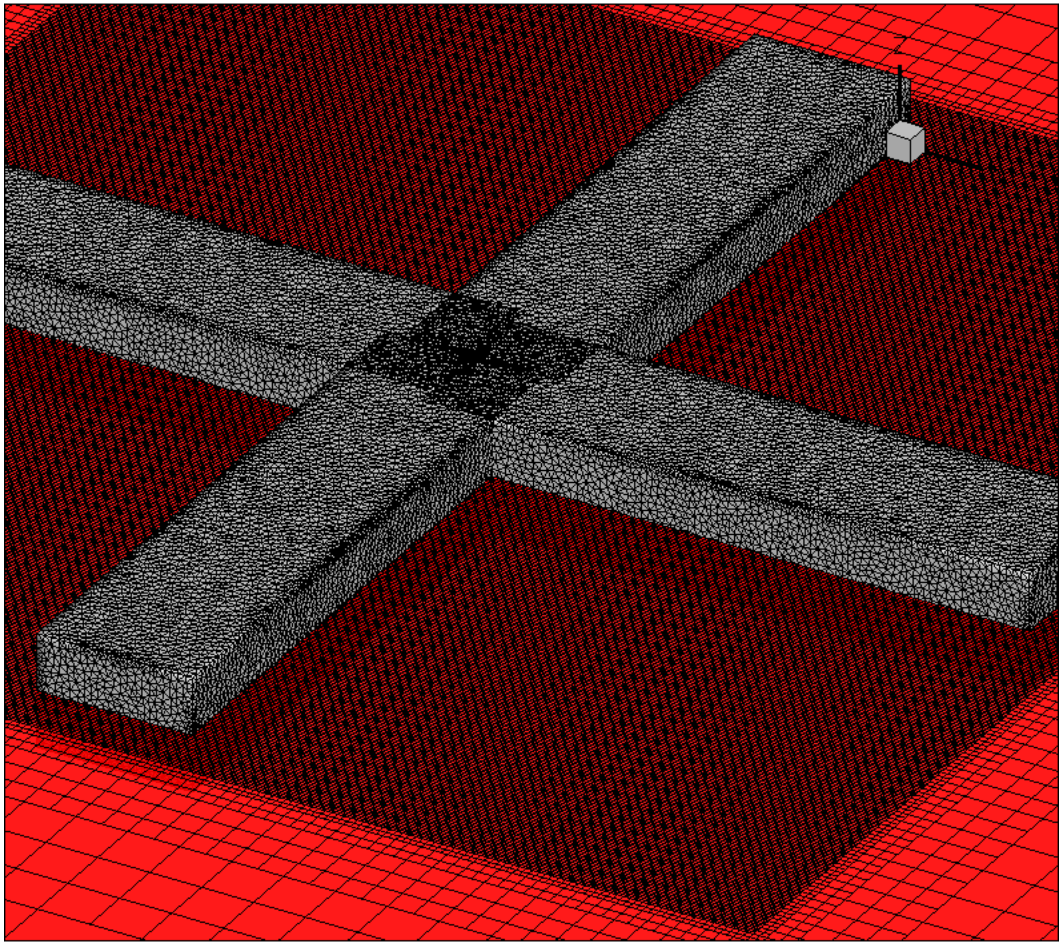
approx. Inverse Map

exact Inverse Map

Total time
per time step (sec)



Results: UH60 case



Near-Body cells	4.57 Millions
Off-Body cells	7.3 Millions
Nb of Processors	128

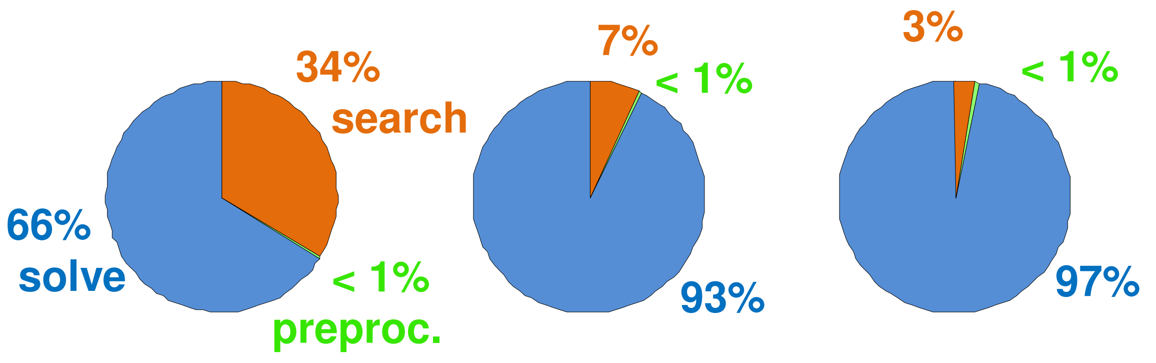
Results: UH60 case

ACCURACY

	RECEPTORS	ORPHANS
ADT method	81223	0
approx. Inverse Map method	80998 (-225)	50
exact Inverse Map method	81203 (-20)	0

SPEED

Task share
(% of total time per time step)

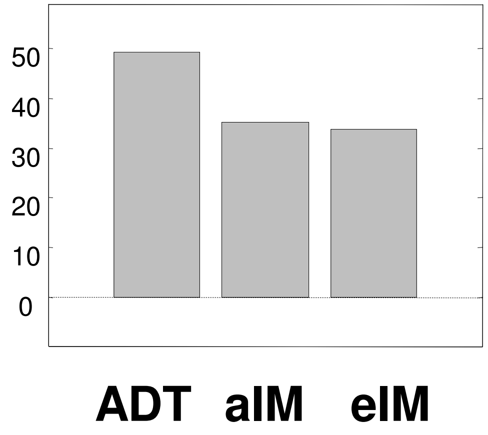


ADT

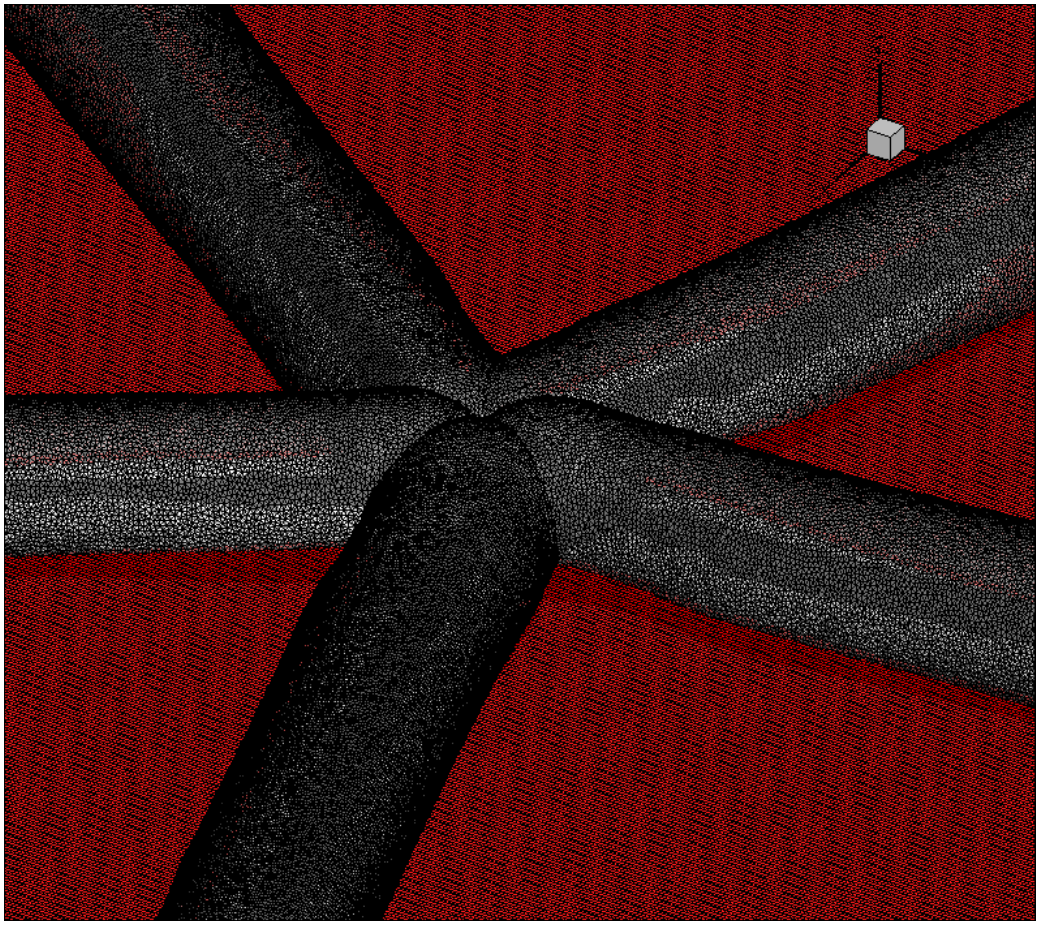
approx.
Inverse Map

exact
Inverse Map

Total time
per time step (sec)



Results: MDART case



Near-Body cells	15.57 Millions
Off-Body cells	65.29 Millions
Nb of Processors	240

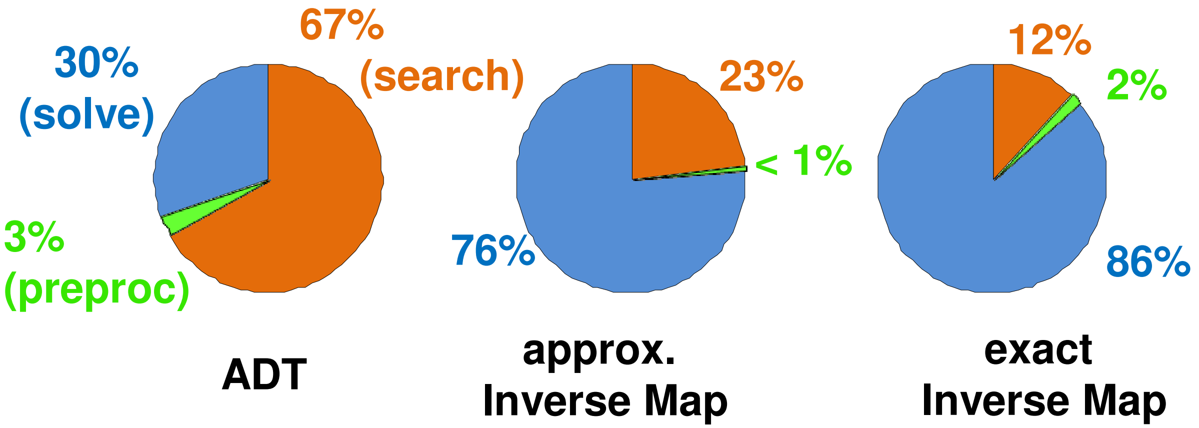
Results: MDART case

ACCURACY

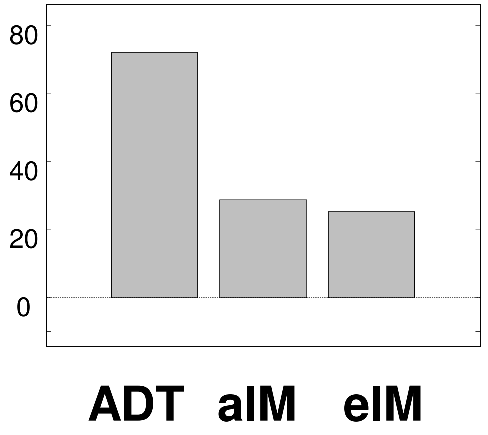
	RECEPTORS	ORPHANS
ADT method	1370834	0
approx. Inverse Map method	1369041 (-1793)	350
exact Inverse Map method	1370784 (-50)	10

SPEED

Task share (% of total time per time step)



Total time per time step (sec)



Conclusions

Explored two methods for donor searches (ADT and eIM)

- ADT search
  - most robust and accurate for donor search (no orphans or incomplete fringes in any case)
  - 10-20 times slower than both aIM and eIM methods
  - 2-3 orders of magnitude faster than brute force
  - Gold standard to verify accuracy of donor search
- Exact Inverse Maps (eIM)
  - Accuracy comparable to ADT
  - 2-5 times faster than approximate inverse maps
  - Optimal sub-block size found to correspond to 20% cell center containment

This work was supported by grant funding through the Army Research Office with Dr. Roger Strawn as the Technical Monitor