

Adaptive Refinement in GlennICE

William Wright

HX5, LLC

AIAA Aviation 2021, Aug 2-5

Co-Authors

David Rigby

Eric Galloway

Christopher Porter

This material is a work of the U.S. Government and is not subject to copyright protection in the United States

Published by the American Institute of Aeronautics and Astronautics, Inc., with permission.

Background – Lagrangian vs Eulerian

Lagrangian

More accurate

Does not assume continuum

Easier to do splashing/breakup

Eulerian

Generally faster

Easier to obtain off-body water concentration

Easier to do two-phase coupling

Why is Lagrangian More Accurate?

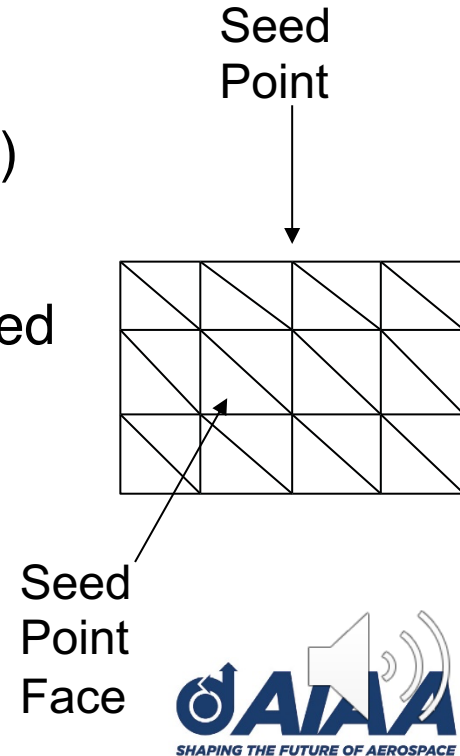
- Eulerian schemes use the same grid for air flow and particle flow
 - The grid that captures air velocity gradients is not optimized to capture particle velocity gradients
- Lagrangian schemes in essence create a “grid” that is optimized for each particle size
- However, it can take a large number of trajectories to achieve this effect

Optimizing a Lagrangian Process for Icing

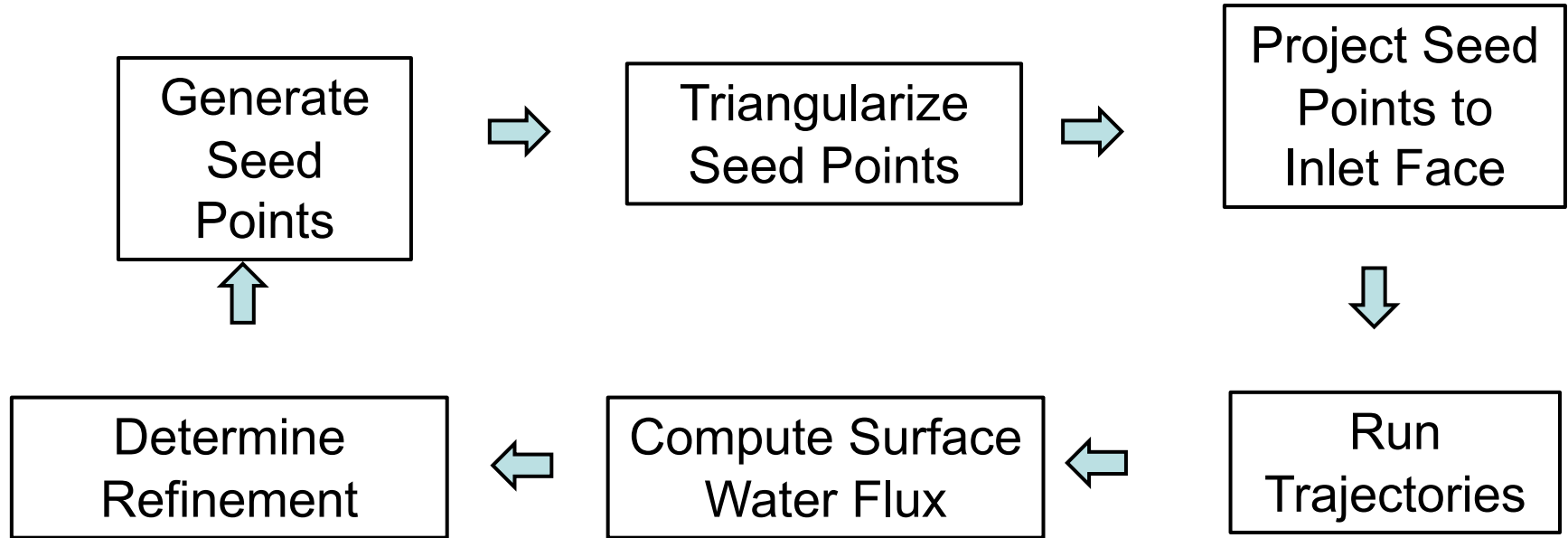
- Determine best (speed/accuracy) integration scheme
 - Dormand-Prince algorithm used in MatLab, Python
- Optimize for multi-processors
- Minimize number of trajectories
 - How best to determine release point of trajectories that will hit the surface?

GlennICE Adaptive Refinement Process

- Select release point to refine
 - Minimum wall distance criteria
 - Hit surface criteria (trajectory hit the icing surface)
- Refine the face
 - Refine faces containing the points that are selected
 - Refine the faces that neighbor those faces
 - Can expand this to the neighbors of neighbor faces

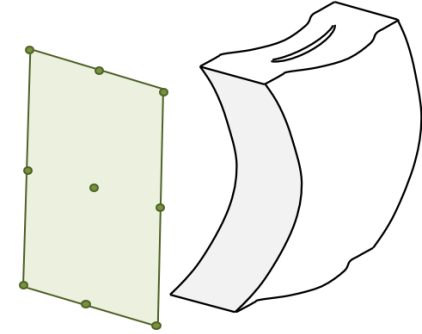


Adaptive Refinement Flow Chart

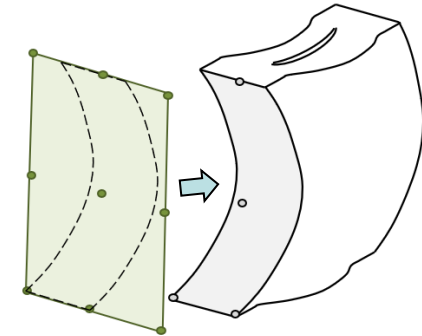


Seed Points and Release Points

- Seed points originate on a two-dimensional plane.
- Release points are the projection of seed points onto the *Inlet* surface of the CFD solution.
- Theoretically not all seed points will become release points, however v1.1.0 requires this.



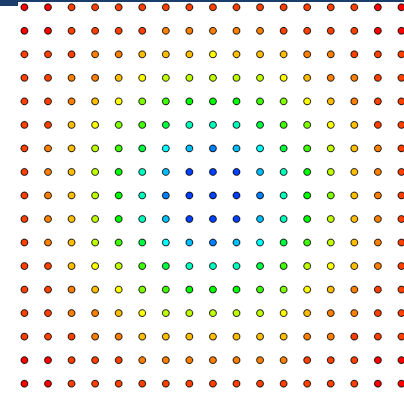
Seed points (green dots) residing in the seed plane.



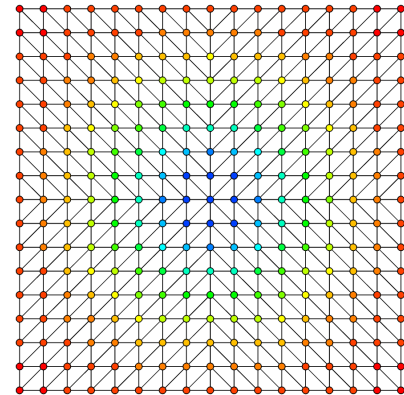
The projection of seed points (green dots) onto the Inlet surface creating release points (grey dots).

Delaunay Triangulation

- There are a few uses of converting the point cloud into a mesh. The primary reason is to eliminate the need for neighbor searching.
- Delaunay triangulation is being leveraged here to accomplish the conversion of a point cloud to a mesh.



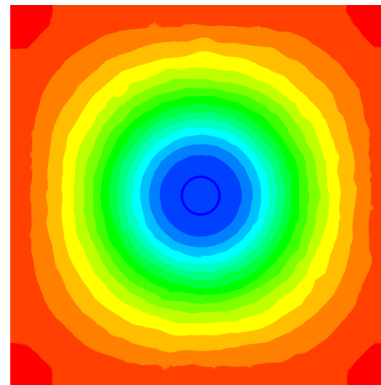
Point cloud prior to Delaunay triangulation.



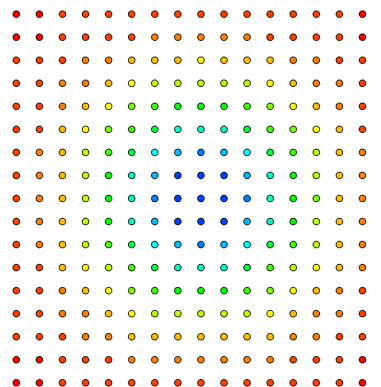
Mesh of the point cloud generated by Delaunay triangulation

Adaptive Refinement Problem

- The minimum wall distance of each trajectory can be stored at the seed point that led to that trajectory.
- The seed points are co-planar, and therefore the computational problem being solved is now two-dimensional.
- The problem is a minimum search on a discretized two-dimensional data set.
- The sink behavior of the problem is leveraged in the minimum search.



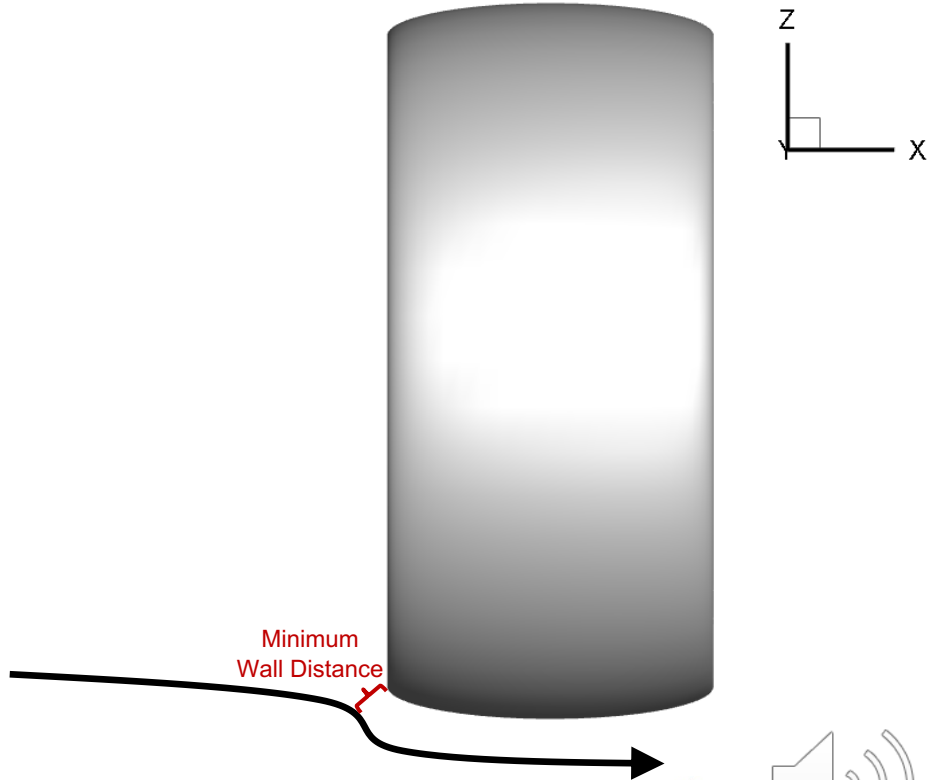
The two-dimensional minimum wall distance solution.



Data available at a given iteration.

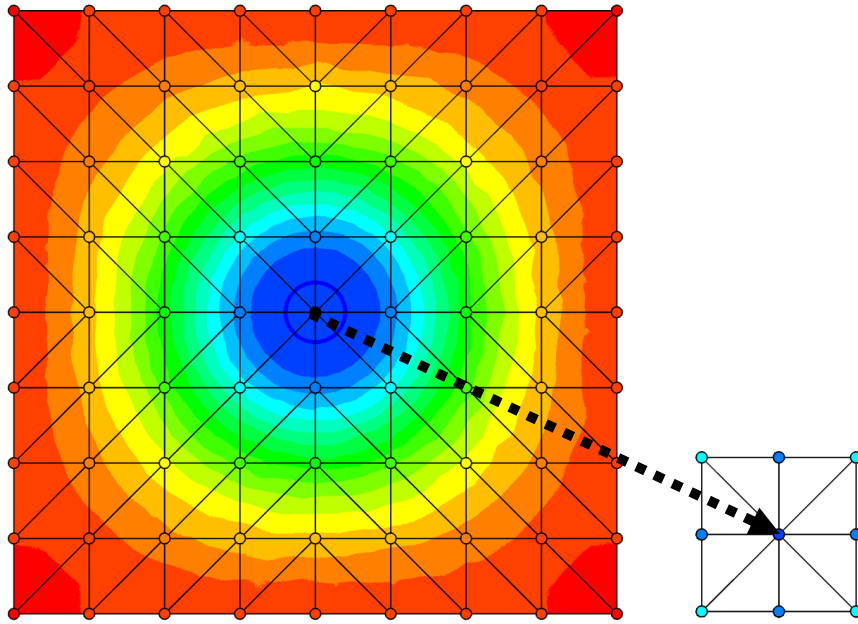
Minimum Wall Distance

- The minimum wall distance is the closest a trajectory comes to a surface of interest.
- Currently use the wall distance from the CFD turbulence model, so the surfaces of interest are all viscous surfaces in the simulation.
- There is a desire to compute wall distances from a specific (icing) surface directly.



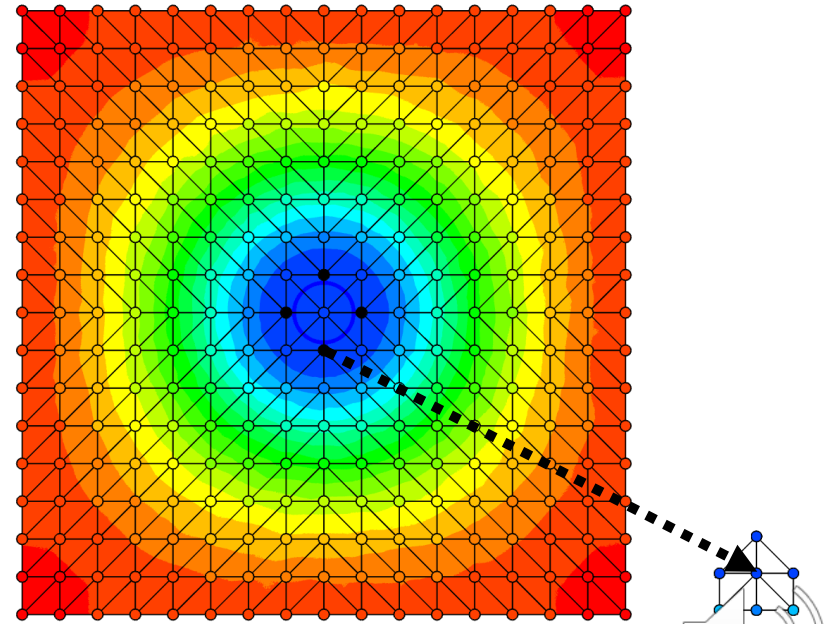
Local Minimum Search

- Defined as a node having a minimum wall distance less than its neighbors, but not equal to zero.



Refinement Level N

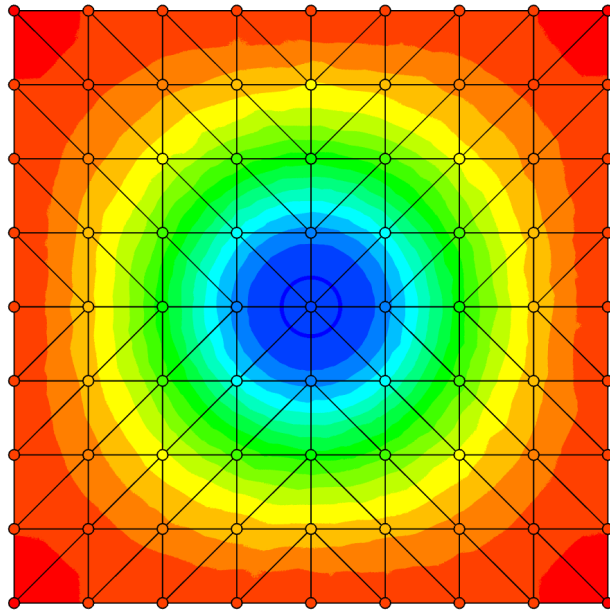
11



Refinement Level N+1

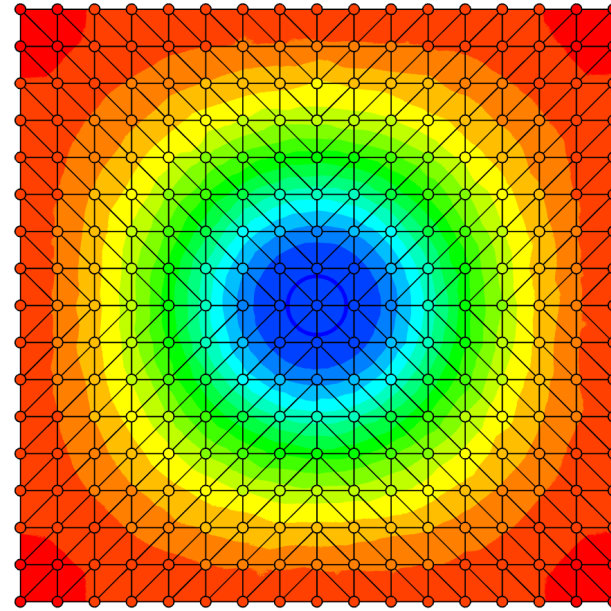
Global Minimum Search

- Defined as a node having a wall distance of exactly zero.
 - i.e. the trajectory impinges.



Refinement Level N

12



Refinement Level N+1

Robustness/Efficiency Controls

```
&adaptive_refinement  
  max_face_count      = huge  
  local_min_neighbors = 5  
  global_min_neighbors = 5  
  cease_neighbor_refinement = .false.  
/
```

max_face_count = huge

Stops the adaptive refinement algorithm from refining around trajectories that hit a face that has already been impinged by the number of trajectories specified here. The default value of huge effectively uniformly refines the region where trajectories impinge.

local_min_neighbors = 5

Increases the robustness of the adaptive refinement algorithm, at the expense of computational cost, by adding conservatism to the local minimum search and refine portion of the adaptive refinement algorithm.

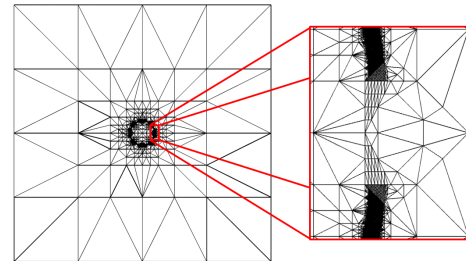
global_min_neighbors = 5

Increases the robustness of the adaptive refinement algorithm, at the expense of computational cost, by adding conservatism to the global minimum search and refine portion of the adaptive refinement algorithm.

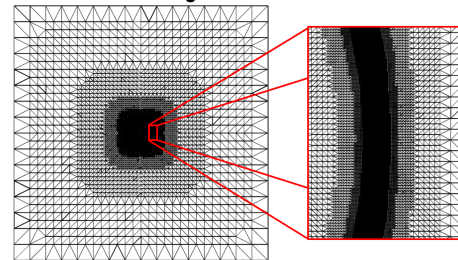
cease_neighbor_refinement = .false.

Shuts off the conservative neighbor refinement when no local minimums are found.

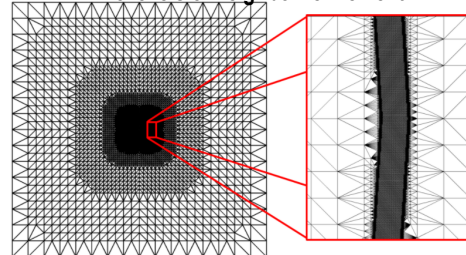
13



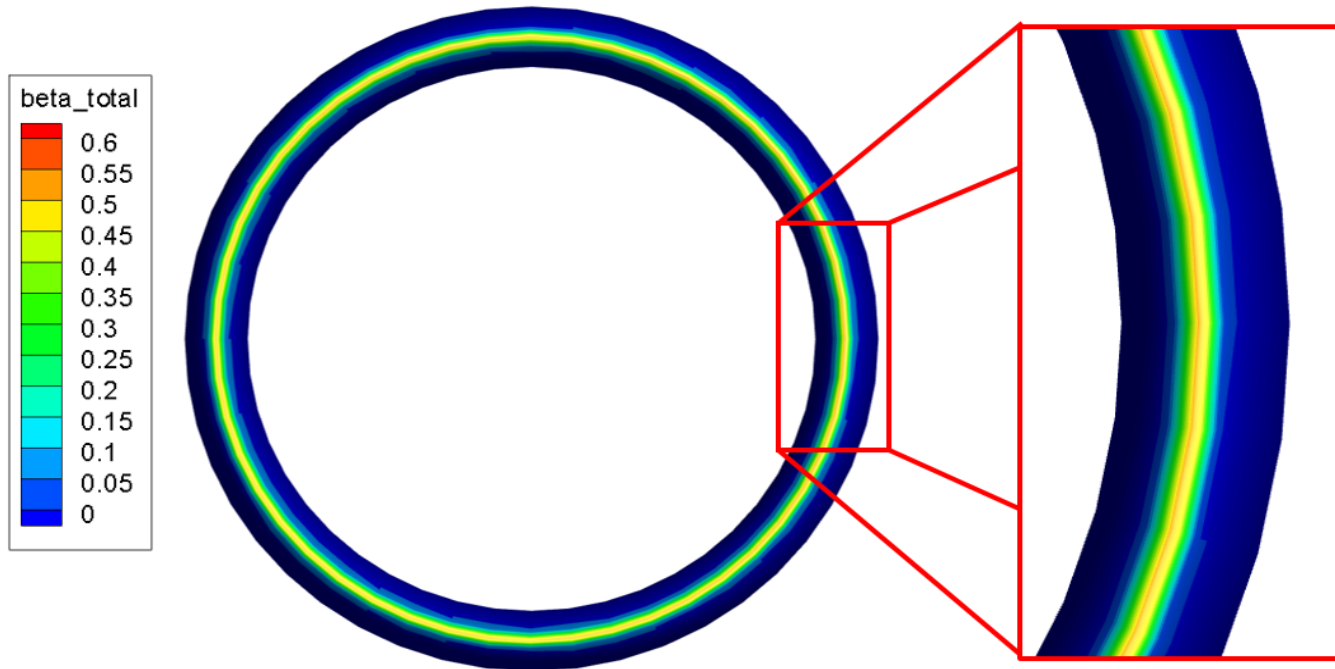
No neighbor refinement.



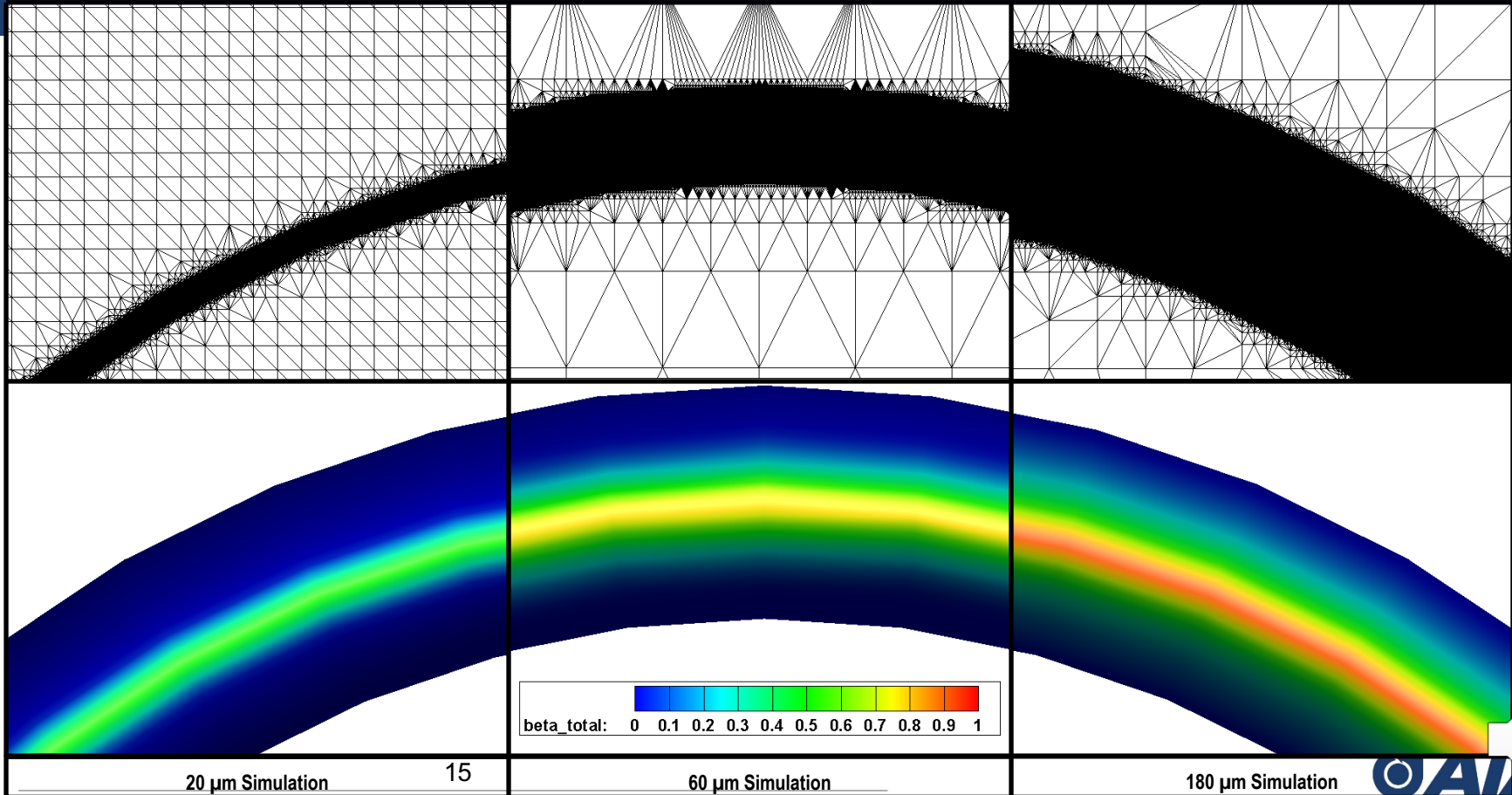
Five levels of neighbor refinement.



Annulus Collection Efficiency



Results



Conclusions

- Adaptive refinement allows GlennICE to run more trajectories that hit or are near to the surface
 - Trade off between robustness (finds the geometry) and speed (number of trajectories)
- Routine can be optimized further
 - Refining only edges rather than faces
 - Refining based on collection efficiency gradients
- Still need to implement convergence criteria

Acknowledgements

- The authors would like to acknowledge the support of the Advanced Air Transport Technology project for support for this research effort.