A Revised Tangle-Free Algorithm for Two-Dimensional Mesh Motion Problems

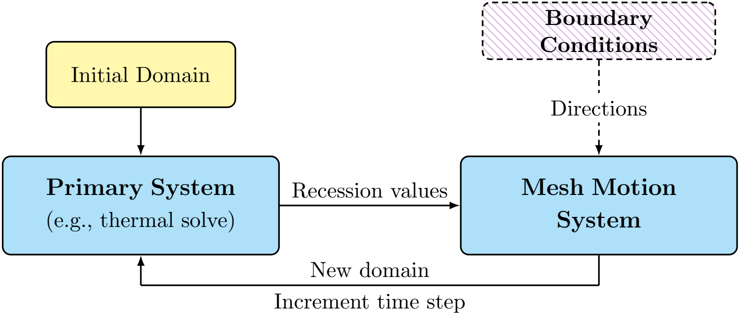
Justin Droba
Adam Amar

Lyndon B. Johnson Space Center

USNCCM 2017
July 19, 2017

Overview of Mesh Motion

- Most numerics are posed on fixed computational grids
- Mesh motion for domains whose boundary evolves in time
  - Examples: stress/strain, deformation, ablation for re-entry
- Mesh motion is an auxiliary process to a primary system



- Can also be coupled with main system dynamics

Some Specifics

- Two general classes of mesh motion algorithms:
    1. Algebraic methods powered by interpolation
    2. Dynamical systems powered by PDEs

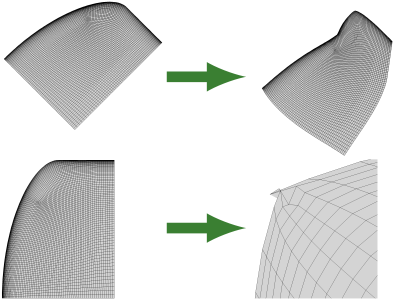
$$\begin{aligned} \mathcal{L}\mathbf{u} &= \mathbf{f} && \mathbf{x} \in \Omega(t) \\ \mathbf{u} &= \Delta s(\mathbf{x}, t)\,\mathbf{d}(\mathbf{x}, t) && \mathbf{x} \in \partial\Omega(t) \end{aligned}$$

- Non-boundary node locations determined by PDE solution
- Nature of $\mathcal{L}$ determines precisely how they move
    - Common choices: **Linear elasticity**, biharmonic
- Boundary conditions largely dictate performance of scheme
- Assume: primary system (thermal solve for us) supplies recession values Δs at specified points of faces on $\partial\Omega$
- Directions $\mathbf{d}$ are what we need to determine!

Tangle-free Boundary Conditions

Introduced in **AIAA 2016-3386**, tangle-free boundary conditions are algorithms for **d** that

1. Enable sliding along arbitrary surfaces
2. Prevent mesh tangling



Implementation as in paper needs some improvement:

- Boundary condition enforced at nodes via penalty method
  - ✗ Poorly conditioned as mesh gets finer and larger
  - ✗ Linear solvers don't converge
- Rudimentary treatment of adjacent receders
  - ✗ Requires unconventional and funky grids

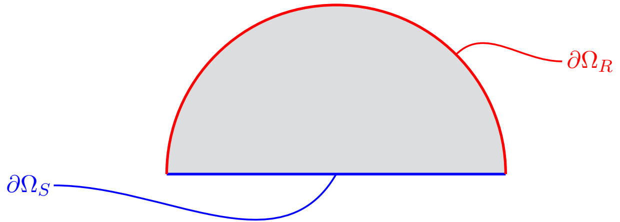
Boundary Apportionment

Decompose the boundary $\partial\Omega$ into three distinct subsets:

$\partial\Omega_R$:	Receding	Motion of Δs along $\mathbf{d}$
$\partial\Omega_S$:	Sliding	No motion in normal direction
$\partial\Omega_F$:	Stationary	No motion at all

Sets **not** disjoint but can only intersect in single points in 2D.

An example of sliding and receding sets in action:



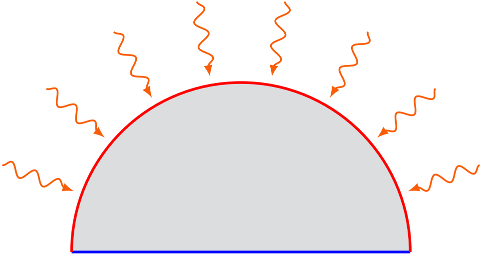
Boundary Apportionment

Decompose the boundary $\partial\Omega$ into three distinct subsets:

$\partial\Omega_R$:	Receding	Motion of Δs along $\mathbf{d}$
$\partial\Omega_S$:	Sliding	No motion in normal direction
$\partial\Omega_F$:	Stationary	No motion at all

Sets **not** disjoint but can only intersect in single points in 2D.

An example of sliding and receding sets in action:



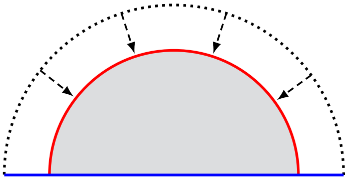
Boundary Apportionment

Decompose the boundary $\partial\Omega$ into three distinct subsets:

$\partial\Omega_R$:	Receding	Motion of Δs along $\mathbf{d}$
$\partial\Omega_S$:	Sliding	No motion in normal direction
$\partial\Omega_F$:	Stationary	No motion at all

Sets **not** disjoint but can only intersect in single points in 2D.

An example of sliding and receding sets in action:



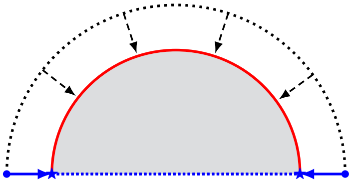
Boundary Apportionment

Decompose the boundary $\partial\Omega$ into three distinct subsets:

$\partial\Omega_R$:	Receding	Motion of Δs along $\mathbf{d}$
$\partial\Omega_S$:	Sliding	No motion in normal direction
$\partial\Omega_F$:	Stationary	No motion at all

Sets **not** disjoint but can only intersect in single points in 2D.

An example of sliding and receding sets in action:



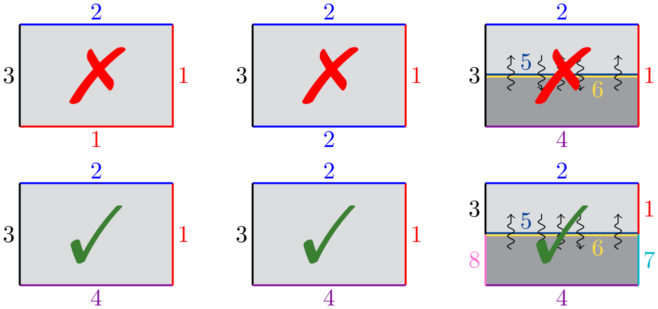
Side Set

Definition

A *side set* is a subset of $\partial\Omega$ contained entirely in one of the receding, sliding, or stationary parts of the boundary.

Tangle-free algorithms assume that side sets do not...

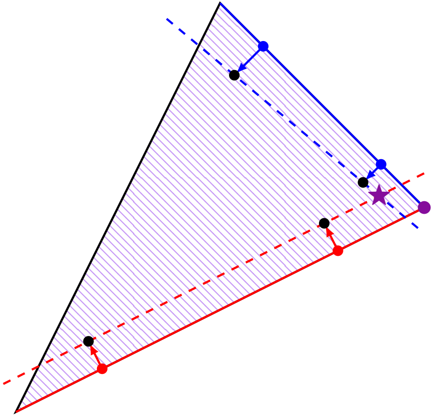
1. wrap topological corners
2. have topological disconnectivity
3. span multiple moving, contact conductance blocks



Corner Nodes

Definition

A *corner node* is a node at the junction of two side sets.

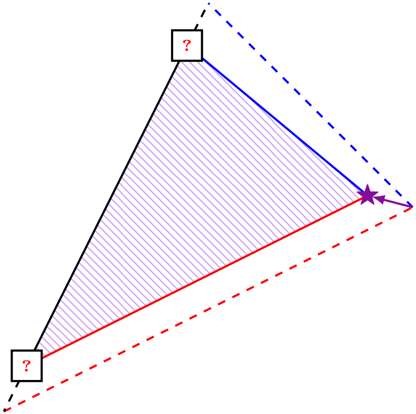


1. Identify edges and motion points (where Δs known).
2. Move points Δs along $\boldsymbol{\nu}$.
3. Compute best-fit line through new points
4. Determine intersection of these planes. Is new node.

Corner Nodes

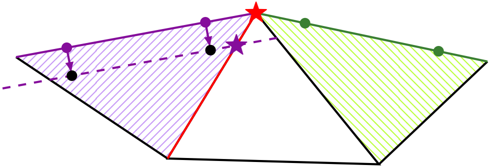
Definition

A *corner node* is a node at the junction of two side sets.



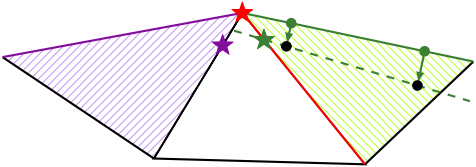
1. Identify edges and motion points (where Δs known).
2. Move points Δs along $\boldsymbol{\nu}$.
3. Compute best-fit line through new points
4. Determine intersection of these planes. Is new node.
5. Direction is unit vector from old to new node. Magnitude of actual difference is displacement.

Receding Side Sets



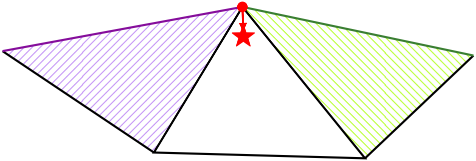
1. Identify edges and motion points.
2. Move points of first edge Δs along $\boldsymbol{\nu}$.
3. Compute best-fit line through new points.
4. Identify internal edge containing node in same element.
5. Determine intersection of best-fit and internal edge.

Receding Side Sets



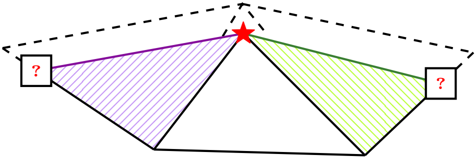
1. Identify edges and motion points.
2. Move points of first edge Δs along $\boldsymbol{\nu}$.
3. Compute best-fit line through new points.
4. Identify internal edge containing node in same element.
5. Determine intersection of best-fit and internal edge.
6. Repeat Steps 2-5 for the other element.

Receding Side Sets



1. Identify edges and motion points.
2. Move points of first edge Δs along $\boldsymbol{\nu}$.
3. Compute best-fit line through new points.
4. Identify internal edge containing node in same element.
5. Determine intersection of best-fit and internal edge.
6. Repeat Steps 2-5 for the other element.
7. Compute centroid of the two intersection points.
8. Direction is unit vector from old to new. Magnitude of actual difference is displacement.

Receding Side Sets

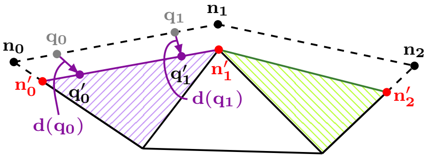


1. Identify edges and motion points.
2. Move points of first edge Δs along $\boldsymbol{\nu}$.
3. Compute best-fit line through new points.
4. Identify internal edge containing node in same element.
5. Determine intersection of best-fit and internal edge.
6. Repeat Steps 2-5 for the other element.
7. Compute centroid of the two intersection points.
8. Direction is unit vector from old to new. Magnitude of actual difference is displacement.

Change in Enforcement

- Previously: boundary cond. enforced as discrete Dirichlet.
- Better conditioned system matrix with weak enforcement:

$$a(\mathbf{u},\mathbf{v}) = \int_{\partial\Omega} \Delta s(\mathbf{x})\langle \mathbf{d}(\mathbf{x}),\mathbf{v}\rangle_{\mathbb{R}^2} d\mathbf{S}$$

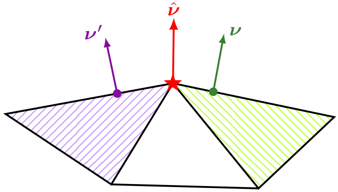


First, compute **all** new nodes $\mathbf{n'_i}$. Then for each boundary edge:

1. Identify quadrature points on original edge.
2. Parameterize edge as $\ell(t)$ and compute t_i s.t. $\mathbf{q_i} = \ell(t_i)$.
3. Parameterize new edge as $\ell'(t)$ with same t scale and find new quadrature points $\mathbf{q'_i} = \ell'(t_i)$.
4. Set $\mathbf{d}(\mathbf{q}_i) = \mathbf{q'_i} - \mathbf{q_i}$ and use to approximate integral.

Sliding Side Sets: The Old Way

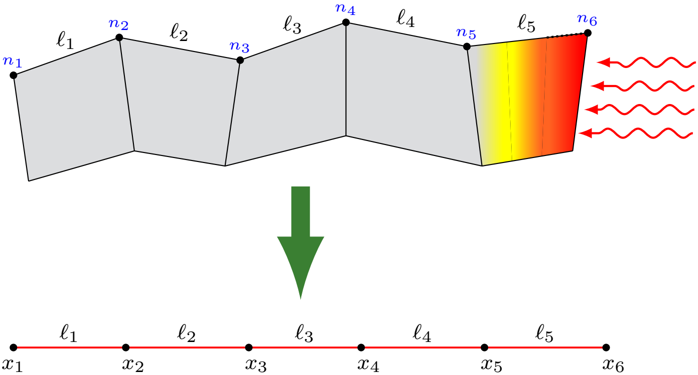
- Sliding sets: no motion allowed in normal direction.
- Normal not defined at nodes in discrete geometry
- Previously: take average of neighboring sides' normals to approximate one at node



$$\hat{\boldsymbol{\nu}} = \frac{\boldsymbol{\nu} + \boldsymbol{\nu}'}{|\boldsymbol{\nu} + \boldsymbol{\nu}'|}$$

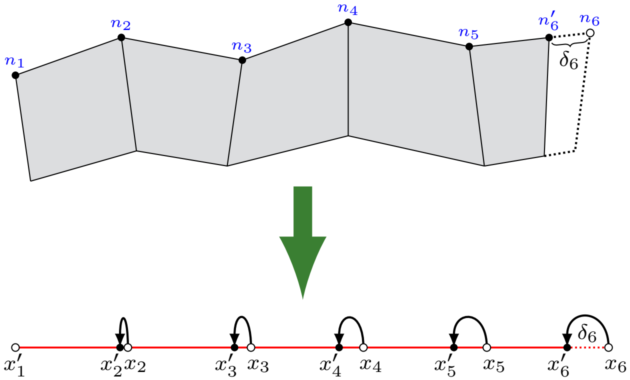
- Enforce a zero motion condition in $\mathbf{d} = \hat{\boldsymbol{\nu}}$
- Let system dynamics determine motion in direction $\hat{\boldsymbol{\nu}}^{\perp}$
- **Big Problem:** New method of enforcement requires all degrees of freedom on boundary to be known *a priori*...

Sliding Side Sets: 1D Shadow Grids



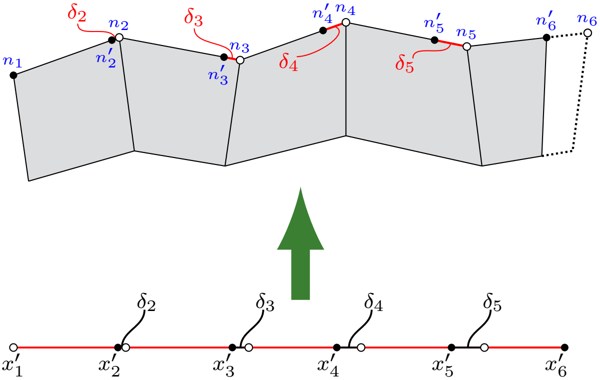
- Nodes in sliding sidesets are updated by computing *a priori* motion along a 1D shadow grid
  - Space nodes according to lengths of edges on 2D boundary
  - Use corner motion to construct boundary conditions

Sliding Side Sets: 1D Shadow Grids



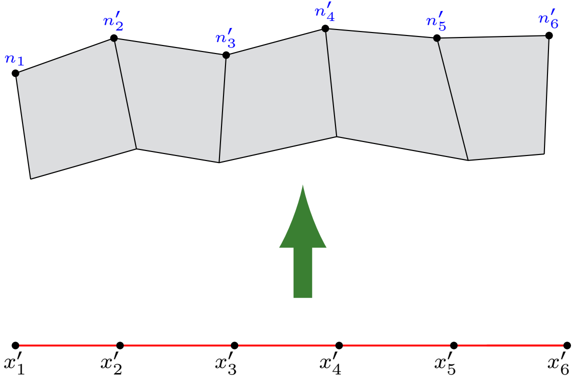
- Nodes in sliding sidesets are updated by computing *a priori* motion along a 1D shadow grid
  - Space nodes according to lengths of edges on 2D boundary
  - Use corner motion to construct boundary conditions
  - Formula for solution to 1D linear elasticity problem

Sliding Side Sets: 1D Shadow Grids



- Nodes in sliding sidesets are updated by computing *a priori* motion along a 1D shadow grid
  - Space nodes according to lengths of edges on 2D boundary
  - Use corner motion to construct boundary conditions
  - Formula for solution to 1D linear elasticity problem

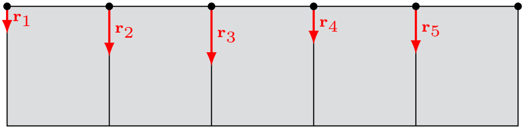
Sliding Side Sets: 1D Shadow Grids



- Nodes in sliding sidesets are updated by computing *a priori* motion along a 1D shadow grid
  - Space nodes according to lengths of edges on 2D boundary
  - Use corner motion to construct boundary conditions
  - Formula for solution to 1D linear elasticity problem

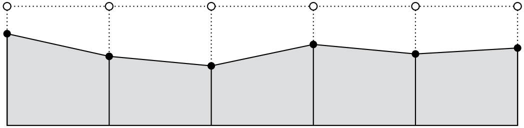
Adjacent Receders Corrections

- Motion of receding nodes is along edges



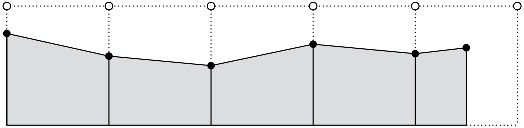
Adjacent Receders Corrections

- Motion of receding nodes is along edges



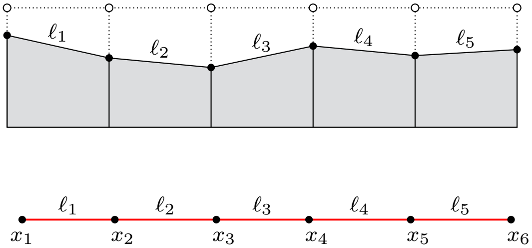
Adjacent Receders Corrections

- Motion of receding nodes is along edges
- Corner element loses mass fastest in adjacent receders case
    - Corner can leapfrog internal node and trigger a failure



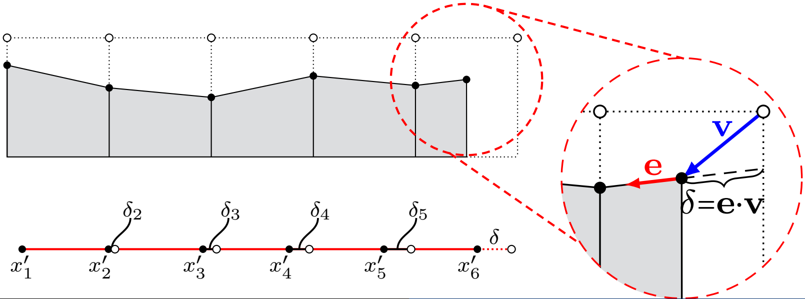
Adjacent Receders Corrections

- Motion of receding nodes is along edges
- Corner element loses mass fastest in adjacent receders case
  - Corner can leapfrog internal node and trigger a failure
- Adjacent receders require correction via 1D shadow slider
  - Construct 1D grid based on *post-recession* node locations



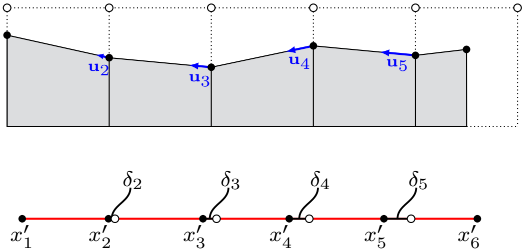
Adjacent Receders Corrections

- Motion of receding nodes is along edges
- Corner element loses mass fastest in adjacent receders case
  - Corner can leapfrog internal node and trigger a failure
- Adjacent receders require correction via 1D shadow slider
  - Construct 1D grid based on *post-recession* node locations
  - Construct b.c. from projection onto "phantom edge"
  - Use 1D grid updates as displacements along *new* edges



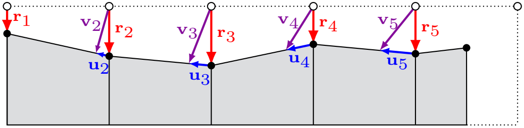
Adjacent Receders Corrections

- Motion of receding nodes is along edges
- Corner element loses mass fastest in adjacent receders case
  - Corner can leapfrog internal node and trigger a failure
- Adjacent receders require correction via 1D shadow slider
  - Construct 1D grid based on *post-recession* node locations
  - Construct b.c. from projection onto “phantom edge”
  - Use 1D grid updates as displacements along *new* edges



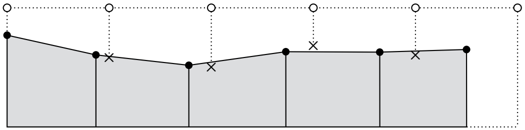
Adjacent Receders Corrections

- Motion of receding nodes is along edges
- Corner element loses mass fastest in adjacent receders case
  - Corner can leapfrog internal node and trigger a failure
- Adjacent receders require correction via 1D shadow slider
  - Construct 1D grid based on *post-recession* node locations
  - Construct b.c. from projection onto "phantom edge"
  - Use 1D grid updates as displacements along *new* edges
  - Total movement of node i is $\mathbf{v}_i = \mathbf{r}_i + \mathbf{u}_i$

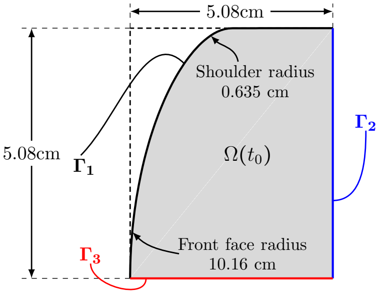


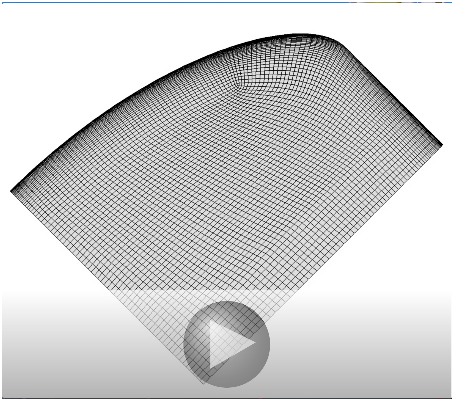
Adjacent Receders Corrections

- Motion of receding nodes is along edges
- Corner element loses mass fastest in adjacent receders case
  - Corner can leapfrog internal node and trigger a failure
- Adjacent receders require correction via 1D shadow slider
  - Construct 1D grid based on *post-recession* node locations
  - Construct b.c. from projection onto "phantom edge"
  - Use 1D grid updates as displacements along *new* edges
  - Total movement of node i is $\mathbf{v}_i = \mathbf{r}_i + \mathbf{u}_i$



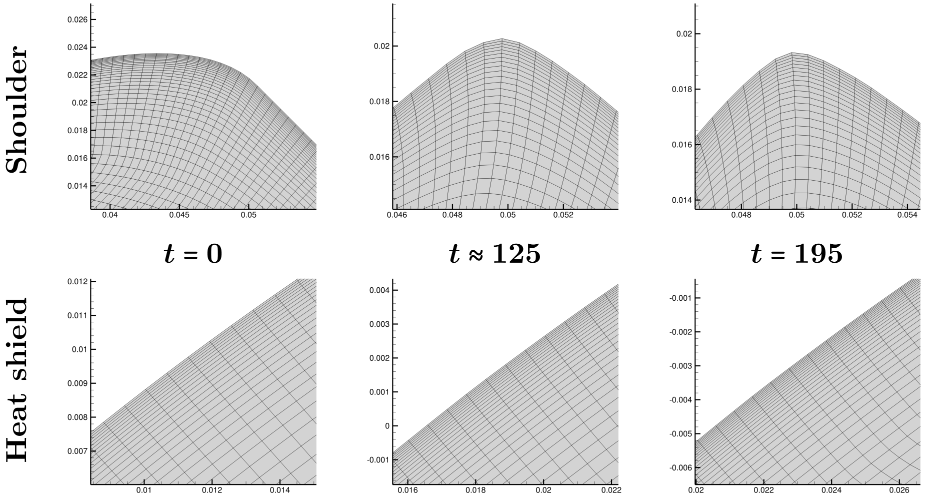
Ablating Rotating Iso-q



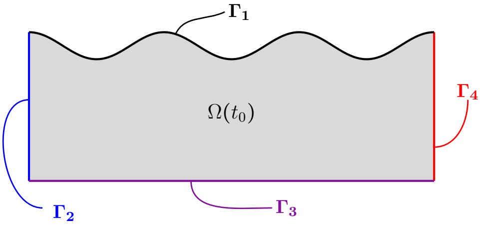


Γ_1	Receding	Aeroheating
Γ_2	Sliding	Zero heat flux
Γ_3	Sliding	Zero heat flux

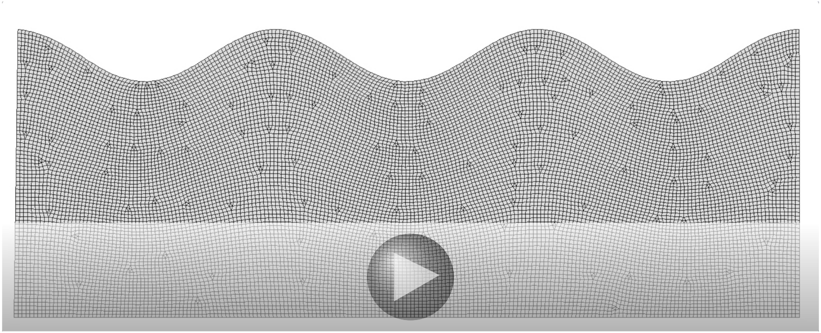
Mesh Detail



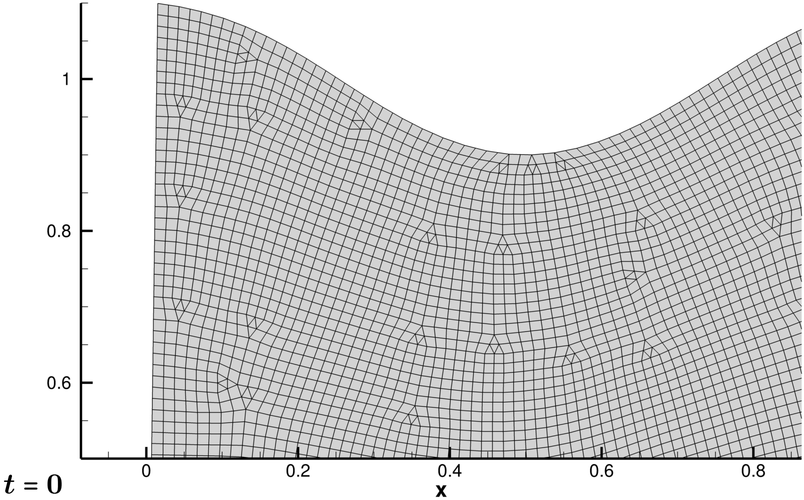
Melting Sinusoidal Surface Slider



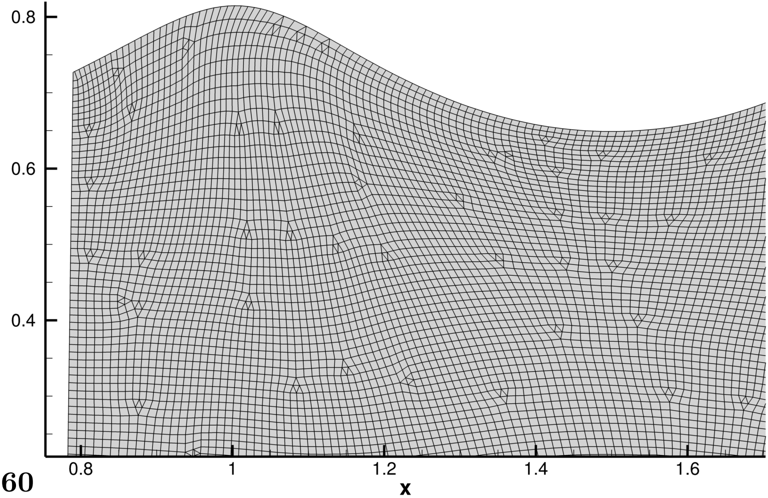
Γ_1	Receding
Γ_2	Receding
Γ_3	Sliding
Γ_4	Fixed



Mesh Detail

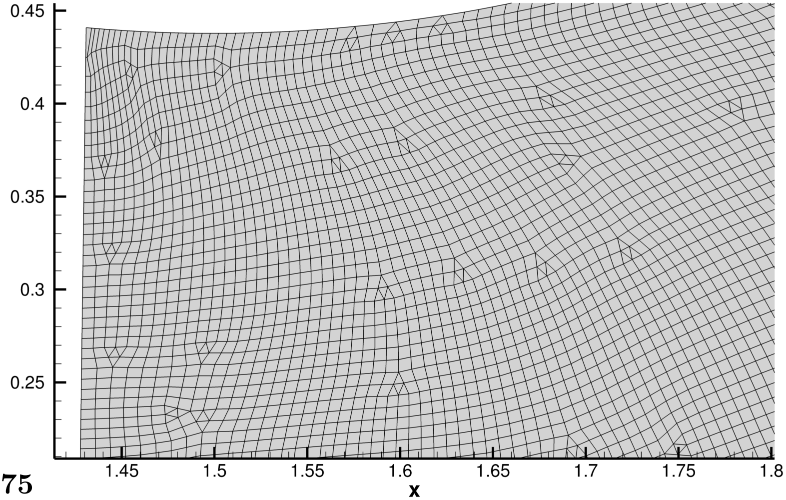


Mesh Detail



$t = 260$

Mesh Detail



$t = 475$

Conclusions and Acknowledgements

- To be completed

The authors would like to acknowledge the following people:

- Dr. Ben Kirk (NASA JSC, EG3 Branch Chief)
- Giovanni Salazar (NASA JSC)
- Brandon Oliver (NASA JSC)

For more details, see AIAA 2016-3386.