Tangle-Free Finite Element Mesh Motion for Ablation Problems

Justin Droba[*]

NASA Lyndon B. Johnson Space Center, Houston, TX 77058

JSC Engineering, Technology, and Science (JETS): Jacobs Technology and HX5, LLC

In numerical simulations involving boundaries that evolve in time, the primary challenge is updating the computational mesh to reflect the physical changes in the domain. In particular, the fundamental objective for any such "mesh motion" scheme is to maintain mesh quality and suppress unphysical geometric anamolies and artifacts. External to a physical process of interest, mesh motion is an added component that determines the specifics of how to move the mesh given certain limited information from the main system. This paper develops a set of boundary conditions designed to eliminate tangling and internal collision within the context of PDE-based mesh motion (linear elasticity). These boundary conditions are developed for two- and three-dimensional meshes. The paper presents detailed algorithms for commonly occuring topological scenarios and explains how to apply them appropriately. Notably, the techniques discussed herein make use of none of the specifics of any particular formulation of mesh motion and thus are more broadly applicable. The two-dimensional algorithms are validated by an extensive verification procedure. Finally, many examples of diverse geometries in both two- and three-dimensions are shown to showcase the capabilities of the tangle-free boundary conditions.

Contents

[*]justin.c.droba@nasa.gov; Aerospace Engineer, Applied Aeroscience and CFD Branch, AIAA Member

Nomenclature

Symbol	**Description**	**Introduced**
$\langle \cdot, \cdot \rangle_{\mathscr{V}}$	Inner product on vector space $\mathscr{V}$; without subscript, standard dot product	Standard Mathematics
∇f	Gradient of *scalar* function $f : \mathbb{R}^d \to \mathbb{R}$, a $1 \times d$ vector	Standard Mathematics
$\nabla \mathbf{F}$	Tensor derivative of *vector* function $\mathbf{F} : \mathbb{R}^d \to \mathbb{R}^d$, a $d \times d$ matrix	Standard Mathematics
$\nabla \cdot \mathbf{F}$	Divergence of the vector field $\mathbf{F}$	Standard Mathematics
$\mathbf{u} \perp \mathbf{v}$	Vector $\mathbf{u}$ is orthogonal (perpendicular) to vector $\mathbf{v}$	Standard Mathematics
$a \mapsto b$	a maps to b	Standard Mathematics
$A \subset B$	A is a subset of B	Standard Mathematics
$a \in A$	a is an element of the set A	Standard Mathematics
$\mathbb{R}$	The set of real numbers	Standard Mathematics
$\mathbb{R}^d$	The set of d tuples of real numbers	Standard Mathematics
$L^2(\Omega; \mathbb{R}^d)$	The set of functions with range in $\mathbb{R}^d$ defined on and component-wise square-integrable on $\Omega \subset \mathbb{R}^d$	Standard Mathematics
$H^1(\Omega; \mathbb{R}^d)$	The set of $L^2(\Omega; \mathbb{R}^d)$ functions with first derivatives also in $L^2(\Omega; \mathbb{R}^d)$	Standard Mathematics
$H^1_0(\Omega; \mathbb{R}^d)$	The set of $H^1(\Omega; \mathbb{R}^d)$ functions that vanish on the boundary $\partial\Omega$	Standard Mathematics
δ_{ij}	Kronecker δ; has value 1 when $i = j$, 0 otherwise	Standard Mathematics
ΔH^0_f	Latent heat of fusion	Section IV.D.2
Δm	Mass loss due to recession	Eq. (2)
Δs	Total change in surface position over time period Δt	Eq. (3)
Δt	Time step for finite element simulation	Section I
$\boldsymbol{\nu}$	Unit normal vector	Convention
ε	Penalty factor (as reciprocal) for setting degrees of freedom	Section IV.B
ζ	Radial basis function	Section V.C.8
$\boldsymbol{\varphi}$	Vector shape function in standard basis $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$	Eq. (5), Eq. (42)

$\tilde{\boldsymbol{\varphi}}$	Vector shape function in basis $\{\mathbf{p}, \mathbf{q}, \mathbf{r}\}$	Eq. (7), Eq. (43)
ψ_k	Scalar finite element shape function; $\psi_k(\mathbf{x}_{k'}) = \delta_{kk'}$ whenever $\mathbf{x}_{k'}$ is a node	Eq. (5)
Ω	Domain, a subset of $\mathbb{R}^d$	Convention
$\partial\Omega$	Boundary of domain Ω	Convention
$\partial\Omega_R$	Receding portion of $\partial\Omega$	Section III
$\partial\Omega_S$	Sliding portion of $\partial\Omega$	Section III
$\partial\Omega_F$	Stationary (fixed) portion of $\partial\Omega$	Section III
$\mathcal{A}$	Global finite element stiffness matrix	Section IV.B
$\mathbf{B}_i$	Perturbation to $\tilde{\mathbf{K}}$ for i directions of motion enforced	Eq. (16)
$\mathbf{b}_i$	Perturbation to $\tilde{\mathbf{E}}$ for i directions of motion enforced	Eq. (16)
$\mathbf{E}$	Element contribution to right-hand side, standard basis $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$	Eq. (12)
$\tilde{\mathbf{E}}$	Element contribution to right-hand side, transformed bases $\{\mathbf{p}_k, \mathbf{q}_k, \mathbf{r}_k\}$	Eq. (13)
$\hat{\mathbf{E}}_i$	Correction to $\mathbf{E}$ for i directions of motion enforcement	Eq. (16)
$\mathcal{F}$	Global righthand-side vector of finite element system	Section IVB
$\mathbf{K}$	Element stiffness matrix (local) in standard basis $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$	Eq. (10)
$\tilde{\mathbf{K}}$	Element stiffness matrix (local) in transformed bases $\{\mathbf{p}_k, \mathbf{q}_k, \mathbf{r}_k\}$	Eq. (11)
$\hat{\mathbf{K}}_i$	Correction to $\mathbf{K}$ for i directions of motion enforcement	Eq. (16)
$\mathcal{M}$	Transformation matrix from $\{\boldsymbol{\varphi}_k\}$ to $\{\tilde{\boldsymbol{\varphi}}_k\}$	Eq. (15), Eq. (44)
m	Number of nodes on moving portions of the boundary of given face	Section IV.A
N	Total number of nodes on a given face	Section IV.A
$\mathbf{p}_k$	Primary direction of motion for k^{th} node $\mathbf{x}_k$	Eq. (7), Eq. (43)
$\mathbf{q}_k$	Secondary direction of motion for k^{th} node $\mathbf{x}_k$	Eq. (7), Eq. (43)
$\mathbf{r}_k$	Tertiary direction of motion for k^{th} node $\mathbf{x}_k$	Eq. (7), Eq. (43)
$\dot{s}$	Surface recession rate (true physical)	Section I
$\hat{s}$	Surface displacement (mesh motion scheme)	Section IV.B
$\hat{\mathbf{s}}$	Vector of surface displacements for each moving node	Eq. (17)
$\mathcal{T}$	Single element (e.g., triangle or hexahedron) in finite element mesh	Section IV.A
T	Temperature (in thermal response)	Convention
$\mathbf{x}_i$	i^{th} node on a given face	Section IV.A
$\mathbf{z}_i$	i^{th} quadrature point on a given face	Algorithm 1

Per convention, vector quantities are denoted in **bold-face**. Components of vectors are indexed with superscript numbers in parentheses, beginning at one: for instance, $x^{(1)}$ denotes the first component of $\mathbf{x}$.

Units

Unless otherwise noted, the following units should be assumed for the frequently occuring quantities below:

Time	seconds
Position	meters
Temperature	Kelvin

These unit labels are generally omitted from tables, figures, and equations involving these quantities.

I. Introduction

With a few noted exceptions, numerical simulations in engineering and mathematics are performed on computational domains that are stationary. It is these exceptions, ones in which the boundary of the domain evolves with the system in time, for which the techniques of this paper are developed. The process by which the computational grid shifts to accomodate the changes in the domain is called "mesh motion." Despite the similarity in name, mesh motion is entirely different from moving mesh methods, a class of adaptive mesh redistribution techniques designed to increase solution accuracy [10]

The mechanics of mesh motion algorithms fall into two broad classes:

1. Algebraic methods powered by interpolation.
2. Dynamical systems methods powered by partial differential equations (PDEs).

Interpolation methods work by computing the explicit deformation of the boundary nodes and then adjusting the inlaid nodes based on distance from the boundary. The algorithms that power these methods are based on inverse distance weighting [14] or radial basis functions [6]. [14] combines its interpolation with a detangling post-processing step to smooth out sections of poor mesh quality.

The PDE methods augment the main system (e.g., thermal response, electromagnetics, etc.) with an extra set of dynamics. These additional equations can be either solved simultaneously with the main system or decoupled into a pre- or post-processing step. In either case, the time dependence of the supplementary PDE is generally limited to the surface recession value $\Delta s(t)$ in the direction $\mathbf{d}(\mathbf{x}, t)$ and has the form

$$\left.\begin{aligned} \mathcal{L}\mathbf{u} &= \mathbf{f} && \mathbf{x} \in \Omega(t) \\ \mathbf{u} &= \Delta s(\mathbf{x}, t)\, \mathbf{d}(\mathbf{x}, t) && \mathbf{x} \in \partial\Omega(t) \end{aligned}\right\} \quad (1)$$

where $\mathbf{u}(\cdot) \in \mathbb{R}^d$ is the displacement of each point (not position!) in the domain $\Omega(t)$ and $\mathcal{L}$ is a linear differential operator. Most commonly, $\mathcal{L}$ is the biharmonic [9] or linear elastic [15] operator. A finite element (FEM) formulation of the latter will be used for the verification and demonstration problems of Sections IV.D, IV.E, and V.D. However, the nature of $\mathcal{L}$ is not important to the developments of this paper.

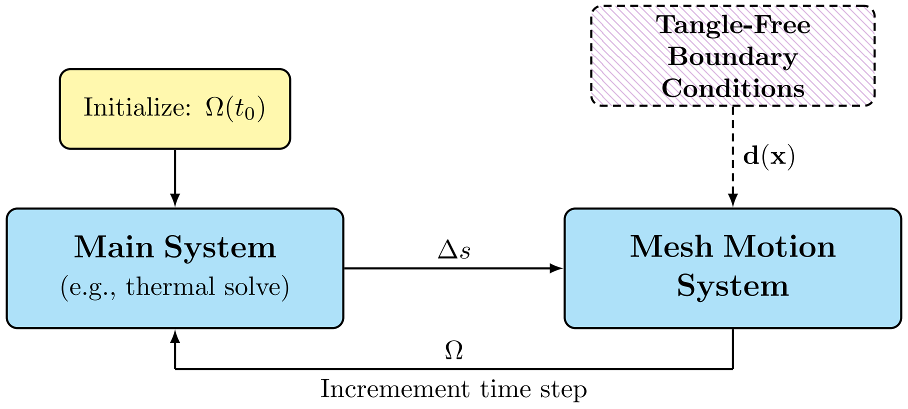


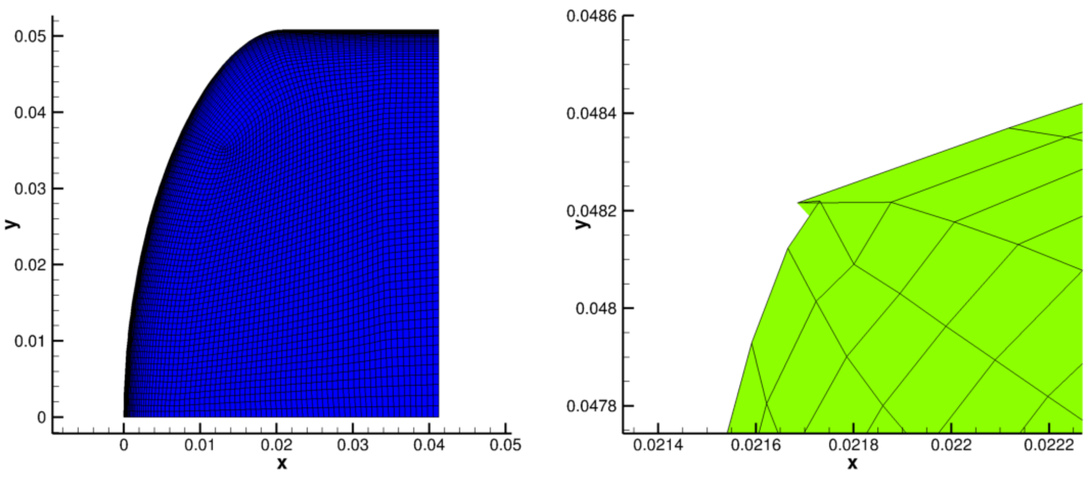
Figure 1. Explicit Mesh Motion. The process begins with an initial time t_0 and domain $\Omega(t_0)$. The main system is then solved on this domain and the values of Δs are provided to the mesh motion system, which is solved on $\Omega(t_0)$. The process of motion creates a new domain $\Omega(t_1)$, which is then fed back and the main system at $t = t_1$ solved on this domain. The process continues for the duration of the simulation: solve the main system at $t = t_n$, obtain $\Delta s(t_n)$, generate a new domain $\Omega(t_{n+1})$, and return to the main system. Note the lag in index: the values of $\Delta s(t_n)$ are used to generate $\Omega(t_{n+1})$. The tangle-free conditions are part of the mesh motion scheme but separated in the diagram for emphasis.

In the numerical simulation, the main system will be solved at the discrete times $t_0, t_1, t_2, \ldots, t_f$. In a decoupled (explicit) model of mesh motion, $\Omega(t_n)$, $\Delta s(\mathbf{x}, t_n)$ and $\mathbf{d}(\mathbf{x}, t_n)$ are assumed to be known *a priori*. With these parameters set, Eq. (1) becomes a time-invariant problem on a fixed domain. Solving it provides *displacements* for each node in $\Omega(t_n)$, application of which generates a new mesh $\Omega(t_{n+1})$ for the main system to be solved at $t = t_{n+1}$. Figure 1 above illustrates the process.

Because of the propensity to tangle, purely PDE-based mesh motion algorithms require additional considerations that the interpolation methods do not. In particular, the PDEs themselves provide no inherent direction of motion and the quality of the result depends heavily on these boundary conditions. A simple approach, which we dub the "simple normal boundary conditions" for lack of other established name, takes $\mathbf{d}$ to be the outward unit normal $\boldsymbol{\nu}$ at each point of $\partial\Omega$ [19]. While simple to implement, the approach has two unfortunate and undesirable consequences:

1. As we shall see in a coming section, it precludes sliding motion along anything other than a coordinate axis. Portions of the boundary declared "sliding" are forbidden to move in the direction normal to the surface; the dynamics Eq. (1) determine the motion in the directions orthogonal to $\boldsymbol{\nu}$.

2. On curved surfaces, the normal vectors from element to element are not parallel. This makes the mesh very likely to collide with itself after significant recession. An example is depicted in Figure 2 below.



(a) **(b)**

Figure 2. Collision Course. The starting geometry is depicted on the left in (a). Along the curved portion of the boundary, $\Delta s > 0$ and on the two flat portions, which are declared as sliding, $\Delta s = 0$. Because the flat sides are aligned with coordinate axes, the simple normal scheme has no issue with them. The curved portion, however, is a different story. (b) shows the result after a long time, zoomed-in to highlight detail. The mesh has tangled within itself and will induce a failure in the main system dynamics shortly after this moment.

The objective of this paper is to craft context-based recipes for the direction $\mathbf{d}$ that prevent tangling. Because the focus is solely on boundary conditions, our developments are applicable to any FEM-based mesh motion; with some adaption, they could be applied outside FEM. We present a scheme for two- and three-dimensional meshes that improves upon the simple normal scheme removes the first restriction while not suffering from the second drawback. We dub this method "tangle-free mesh motion."

In one-dimension, there is only one valid direction—whichever of left or right is into the material—and there is no concept of "sliding." With no possibilty of tangle, mesh motion in one-dimension is fully resolved.

II. Recession Values from Ablation

Problems involving mesh motion arise naturally in the study of the thermal response of ablative materials. Ablation is the process by which a material that is exposed to an extreme thermal environment sacrifices mass through thermal and chemical processes such as sublimination, oxidation, nitridation, or other reaction in order to expel energy and mitigate internal transfer of heat [1]. In doing so, the surface of the material erodes, making mesh motion an intrinsic component within thermal response simulations of ablative materials. For an overview of modeling ablation, see [1]. In this section, we overview how we obtain values for Δs.

In ablation problems, the main system (as we have used the term in preceding section) determines the thermal response of a block of ablating material. Such a solver will be introduced in Section IV.D. For the moment, the details are not important beyond one key fact: it does not furnish the Δs we require. Rather, ablation physics provides us with the total mass loss Δm. Over an interval of time $[t_0, t_f]$, this quantity is related to the *mass loss rate due to ablation* $\dot{m}$ in the following manner:

$$\Delta m = \int_{t_0}^{t_f} \int_{\partial\Omega} \dot{m}\, dS\, dt \tag{2}$$

By solving a surface energy balance equation [1], the rate $\dot{m}$ can be obtained at surface quadrature points. Assuming a constant material density ρ, $\dot{m}$ can be converted to the surface recession rate $\dot{s}$:

$$\dot{s} = \frac{\dot{m}}{\rho}$$
$$\left[\dot{s}\right] = {}^{\mathrm{m}}/_{\mathrm{s}} \quad \left[\dot{m}\right] = {}^{\mathrm{kg}}/_{\mathrm{m \cdot s^2}}$$

The value $\dot{s}$ is a surface velocity. Change in position Δs over an interval of time is found by integrating this value. However, the discrete time stepping of the numerical thermal solve makes $\dot{s}$ available only at the current time step. We assume that over a time period of $\Delta t = t_{n+1} - t_n$, $\dot{s}$ remains constant so that

$$\Delta s = \dot{s}\Delta t \tag{3}$$

More properly, Δs and Δt should be subscripted to indicate time step. Because we are concerned only with the mesh motion module, which is time indpendent in our decoupled approach, we need not track time. We consider Δs as input to mesh motion and worry not about how it was obtained.

III. Apportioning the Boundary

We have mentioned "sliding" portions of the boundary several times but not fully explained the concept. In fact, the boundary $\partial\Omega$ is comprised of three distinct subsets:

$\partial\Omega_R$:	Receding portion	Specified motion of Δs along $\mathbf{d}$
$\partial\Omega_S$:	Sliding portion	Motion in the normal direction forbidden
$\partial\Omega_F$:	Stationary (fixed) portion	No motion in any direction permitted

These sets need not be completely disjoint. However, we must have $|\partial\Omega_R \cap \partial\Omega_S|_{\mathbb{R}^{d-1}} = 0$, $|\partial\Omega_R \cap \partial\Omega_F|_{\mathbb{R}^{d-1}} = 0$, and $|\partial\Omega_S \cap \partial\Omega_F|_{\mathbb{R}^{d-1}} = 0$ so that each pair of subsets intersects only at single points in 2D and lines in 3D. We illustrate practically how these sets might be declared with an example in the figure below.

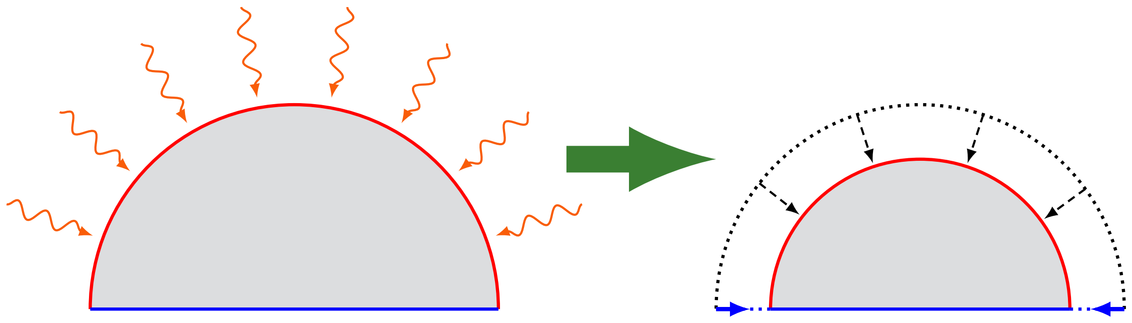


Figure 3. Sliding vs. Receding Subsets. In the left diagram, uniform heating is applied along the circular arc (red), which is declared as $\partial\Omega_R$, the receding portion of the boundary. The flat bottom, in blue, is $\partial\Omega_S$. After recession is applied, the circular arc recedes in to a smaller version of itself (right). The sliding portion moves in reaction to this applied motion.

In the above figure, $\partial\Omega_R$ is the portion of $\partial\Omega$ to which heating is directly applied. The sliding subset $\partial\Omega_S$ has no heating applied to it and therefore does not induce its own motion; it moves in its current plane of existence in response to the adjacent $\partial\Omega_R$ in order to maintain the correct shape.

In general, $\partial\Omega_R$ will be where $\Delta s > 0$ is expected. $\partial\Omega_S$ will be the pieces of the boundary adjacent to members of $\partial\Omega_R$ and have $\Delta s = 0$. The stationary parts establish walls for locking the geometry in place.

IV. Two-Dimensional Meshes

A. General Framework

In this subsection, we lay the foundation for tangle-free boundary conditions. The core idea powering tangle-free mesh motion is local change of basis. In this section, we define new shape functions and develop the transformation matrix that expresses contributions to the new element stiffness matrix in terms of the old.

Suppose Ω has been discretized with a finite element mesh and let $\mathcal{T}$ be an element in this mesh such that $|\mathcal{T} \cap \partial\Omega|_{\mathbb{R}^{d-1}} > 0$. Let $\mathbf{x}_1, \cdots, \mathbf{x}_m \in \partial\mathcal{T} \cap \partial\Omega$ be the boundary nodes of $\mathcal{T}$ and $\mathbf{x}_{m+1}, \cdots, \mathbf{x}_N$ be the non-boundary nodes. On $\mathcal{T}$, the finite element solution to Eq. (1) is a given by a representation in terms of local shape functions:

$$\mathbf{u}\big|_{\mathcal{T}} = \sum_{k=1}^{N} \alpha_k \psi_k(\mathbf{x})\mathbf{e}_1 + \beta_k \psi_k(\mathbf{x})\mathbf{e}_2 \tag{4}$$

The $\mathbf{e}_i$ are the standard (Cartesian) coordinate vectors: $e_i^{(j)} = \delta_{ij}$. As defined in finite element software packages (e.g., `libMesh` [12], `deal.II` [2]), a shape function ψ is the Lagrange interpolant at the nodes of a single element $\mathcal{T}$. It has the property $\psi_i(\mathbf{x}_j) = \delta_{ij}$ whenever $\mathbf{x}_j$ is a node of $\mathcal{T}$. The finite element basis functions are assembled piecewise on neighboring elements from these local shape functions.

While decomposing into components as in Eq. (4) is advantageous in software, it is inconvenient for this discussion. We thus consider a reindexing scheme by creating vector-valued shape functions

$$\boldsymbol{\varphi}_k(\mathbf{x}) = \begin{cases} \psi_k(\mathbf{x})\,\mathbf{e}_1 & 1 \le k \le N \\ \psi_{k-N}(\mathbf{x})\,\mathbf{e}_2 & N+1 \le k \le 2N \end{cases} \tag{5}$$

so that $\mathbf{u}$ is more concisely written as

$$\mathbf{u}\big|_{\mathcal{T}} = \sum_{k=1}^{2N} \alpha_k \boldsymbol{\varphi}_k(\mathbf{x}) \tag{6}$$

It is this separation into Cartesian components that prohibits sliding along any axis other than a coordinate one. The boundary condition is a Dirichlet one of movement specified along $\mathbf{d}$. Suppose we enforce this condition by setting the appropriate degrees of freedom in the above representation. In $\mathbb{R}^2$, if $\mathbf{d} \neq \alpha\mathbf{e}_i$ for some α and i, then for each k such that $\mathbf{x}_k \in \partial\Omega$, we must set the boundary conditions

$$\alpha_k = \frac{\Delta s(\mathbf{x}_k)}{d^{(1)}} \qquad\qquad \alpha_{k-N} = \frac{\Delta s(\mathbf{x}_{k-N})}{d^{(2)}}$$

Because $d^{(i)} \neq 0$ by assumption and $\Delta s \equiv 0$ on $\partial\Omega_S$, all available degrees of freedom go to representing $\mathbf{d}$ in the standard basis so that the above quashes *all* motion when $\mathbf{d}$ is not aligned with a coordinate axis.

The solution, therefore, is to change the basis from the standard $\{\mathbf{e}_1, \mathbf{e}_2\}$ to one more appropriate for local conditions. Define, instead, the following vector basis functions:

$$\tilde{\boldsymbol{\varphi}}_k(\mathbf{x}) = \begin{cases} \psi_k(\mathbf{x})\,\mathbf{p}_k & 1 \le k \le m \\ \psi_k(\mathbf{x})\,\mathbf{e}_1 & m+1 \le k \le N \\ \psi_{k-N}(\mathbf{x})\,\mathbf{q}_{k-N} & N+1 \le k \le m+N \\ \psi_{k-N}(\mathbf{x})\,\mathbf{e}_2 & m+N+1 \le k \le 2N \end{cases} \tag{7}$$

For each k, we require $\mathbb{R}^2 = \text{span}\,\{\mathbf{p}_k, \mathbf{q}_k\}$ so that all solution vectors are representable in $\{\tilde{\boldsymbol{\varphi}}_k\}_{k=1}^{N}$. For the moment, suppose that each $\mathbf{p}_k$ and $\mathbf{q}_k$ is known *a priori*; defining these directions is a matter of circumstance and the focus of a future section. The new shape functions can be written in terms of the old:

$$\tilde{\boldsymbol{\varphi}}_k(\mathbf{x}) = \begin{cases} p_k^{(1)}\boldsymbol{\varphi}_k(\mathbf{x}) + p_k^{(2)}\boldsymbol{\varphi}_{k+N}(\mathbf{x}) & 1 \le k \le m \\ \boldsymbol{\varphi}_k(\mathbf{x}) & m+1 \le k \le N \\ q_{k-N}^{(1)}\boldsymbol{\varphi}_{k-N}(\mathbf{x}) + q_{k-N}^{(2)}\boldsymbol{\varphi}_k(\mathbf{x}) & N+1 \le k \le m+N \\ \boldsymbol{\varphi}_k(\mathbf{x}) & m+N+1 \le k \le 2N \end{cases} \tag{8}$$

We may encode the transformation as matrix multiplication

$$\tilde{\boldsymbol{\varphi}}_k(\mathbf{x}) = \sum_{l=1}^{2N} \mathcal{M}_{kl}\boldsymbol{\varphi}_l(\mathbf{x}) \tag{9}$$

where $\mathcal{M}_{kl}$ are the entries of the matrix

$$\mathcal{M} = \left[\begin{array}{ccc:ccc|ccc:ccc|c}
p_1^{(1)} & & \mathbf{0} & 0 & \cdots & 0 & p_1^{(2)} & & \mathbf{0} & 0 & \cdots & 0 & \\
 & \ddots & & \vdots & \ddots & \vdots & & \ddots & & \vdots & \ddots & \vdots & \\
\mathbf{0} & & p_m^{(1)} & 0 & \cdots & 0 & \mathbf{0} & & p_m^{(2)} & 0 & \cdots & 0 & \\ \hdashline
0 & \cdots & 0 & 1 & & \mathbf{0} & 0 & \cdots & 0 & 0 & \cdots & 0 & \\
\vdots & \ddots & \vdots & & \ddots & & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \}\, N-m \\
0 & \cdots & 0 & \mathbf{0} & & 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & \\ \hline
q_1^{(1)} & & \mathbf{0} & 0 & \cdots & 0 & q_1^{(2)} & & \mathbf{0} & 0 & \cdots & 0 & \\
 & \ddots & & \vdots & \ddots & \vdots & & \ddots & & \vdots & \ddots & \vdots & \\
\mathbf{0} & & q_m^{(1)} & 0 & \cdots & 0 & \mathbf{0} & & q_m^{(2)} & 0 & \cdots & 0 & \\ \hdashline
0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 1 & & \mathbf{0} & \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & & \ddots & & \}\, N-m \\
0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & \mathbf{0} & & 1 & \\
 & & & \multicolumn{3}{c}{\underbrace{\hphantom{0 \cdots 0}}_{N-m}} & & & & \multicolumn{3}{c}{\underbrace{\hphantom{0 \cdots 0}}_{N-m}} &
\end{array}\right]$$

Throughout this report, solid lines in partitioned matrices indicate a block size of $N \times N$.

The element stiffness matrix $\mathbf{K}$ is the Gram matrix for the associated bilinear form $a(\,\cdot\,,\,\cdot\,)$ when the basis functions are restricted only to a single element. It has entries

$$K_{ij} = a(\boldsymbol{\varphi}_i, \boldsymbol{\varphi}_j) \tag{10}$$

Let $\tilde{\mathbf{K}}$ be the element stiffness matrix for the new basis functions $\tilde{\boldsymbol{\varphi}}_k$. Then using Eq. (9), we have

$$\begin{aligned} \tilde{K}_{ij} &= a(\tilde{\boldsymbol{\varphi}}_i, \tilde{\boldsymbol{\varphi}}_j) \\ &= a\left(\sum_{l=1}^{2N} \mathcal{M}_{il}\boldsymbol{\varphi}_l(\mathbf{x}), \sum_{l'=1}^{2N} \mathcal{M}_{jl'}\boldsymbol{\varphi}_{l'}(\mathbf{x})\right) \\ &= \sum_{l=1}^{2N}\sum_{l'=1}^{2N} \mathcal{M}_{jl'}\, a(\boldsymbol{\varphi}_l(\mathbf{x}), \boldsymbol{\varphi}_{l'}(\mathbf{x}))\mathcal{M}_{il} \\ &= \sum_{l=1}^{2N}\left(\sum_{l'=1}^{2N} \mathcal{M}_{jl'} K_{ll'}\right)\mathcal{M}_{il} \end{aligned}$$

which we recognize, though matrix multiplication, as the congruence relation

$$\tilde{\mathbf{K}} = \mathcal{M}^T \mathbf{K} \mathcal{M} \tag{11}$$

The element contribution to righthand-side vector, which we denote as the vector $\mathbf{E}$ which has entries

$$E_k = \langle \mathbf{f}, \boldsymbol{\varphi}_k \rangle_{L^2(\mathcal{T})} \tag{12}$$

can be transformed in a similar manner. Let $\tilde{\mathbf{E}}$ have entries

$$\tilde{E}_k = \langle \mathbf{f}, \tilde{\boldsymbol{\varphi}}_k \rangle_{L^2(\mathcal{T})}$$

Then using Eq. (9), we have

$$\tilde{E}_k = \sum_{l=1}^{2N} \mathcal{M}_{kl} \langle \mathbf{f}, \boldsymbol{\varphi}_l \rangle_{L^2(\mathcal{T})}$$

or in terms of matrix-vector multiplication

$$\tilde{\mathbf{E}} = \mathcal{M}\mathbf{E} \tag{13}$$

A bit of forward thinking reveals that that one-dimensional subspaces are sufficient to describe the motion of a node. Once we've determined the new location of a receding node, we can compute a single vector of motion by subtracting the new position from the old. On $\partial\Omega_S$, motion is forbidden in the normal direction but allowed in the orthogonal complement of the normal—this is again a one-dimensional subspace. Consequently, nonzero motion will occur in the direction of at most one of $\mathbf{p}_k$ and $\mathbf{q}_k$. We can therefore simplify matters considerably by choosing that $\mathbf{q}$ places $\mathcal{M}$ into a favorable form. By taking

$$\mathbf{q}_k = \left[p_k^{(2)}, -p_k^{(1)}\right]^T \tag{14}$$

$\mathcal{M}$ becomes symmetric:

$$\mathcal{M} = \left[\begin{array}{ccc|ccc|ccc|ccc}
p_1^{(1)} & & 0 & 0 & \cdots & 0 & p_1^{(2)} & & 0 & 0 & \cdots & 0 \\
 & \ddots & & \vdots & \ddots & \vdots & & \ddots & & \vdots & \ddots & \vdots \\
0 & & p_m^{(1)} & 0 & \cdots & 0 & 0 & & p_m^{(2)} & 0 & \cdots & 0 \\
\hline
0 & \cdots & 0 & 1 & & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & & \ddots & & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & 0 & & 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\
\hline
p_1^{(2)} & & 0 & 0 & \cdots & 0 & -p_1^{(1)} & & 0 & 0 & \cdots & 0 \\
 & \ddots & & \vdots & \ddots & \vdots & & \ddots & & \vdots & \ddots & \vdots \\
0 & & p_m^{(2)} & 0 & \cdots & 0 & 0 & & -p_m^{(1)} & 0 & \cdots & 0 \\
\hline
0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 1 & & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & & \ddots & \\
0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & & 1
\end{array}\right] \tag{15}$$

If $|\mathbf{p}_k| = 1$, then $\mathcal{M}$ becomes an orthogonal matrix, making inversion trivial. As long as $\mathcal{M}$ is invertible, we can obtain the stiffness for the old basis vectors from that of the new ones. This is especially convenient during implementation because the component-decomposition of $\boldsymbol{\varphi}_k$ greatly simplifies storage and bookeeping requirements. For a true representation involving the $\tilde{\boldsymbol{\varphi}}_k$, we must store thousands of vector components and track what maps where. It is therefore more advantageous to use the $\tilde{\boldsymbol{\varphi}}$ as an intermediate local step and ultimately return to a representation in $\boldsymbol{\varphi}_k$. This is the essence of what this subsection establishes.

B. Enforcing the Motion

In this section and its two subsections, we derive explicitly the contributions to the Cartesian-coordinate element stiffness matrix when motion in one direction and two directions is enforced. We assume that each $\mathbf{p}_k$ and $\mathbf{q}_k$ have been determined before this analysis. It is convenient to present this step before explaining how to choose these directions because it allows us to write concise algorithms for the three motion scenarios.

As long as $\mathcal{M}$ is invertible, we can write $\mathbf{K}$ and $\mathbf{E}$ in terms of $\tilde{\mathbf{K}}$ and $\tilde{\mathbf{E}}$ reversing Eq. (11) and Eq. (13):

$$\mathbf{K} = \mathcal{M}^{-T}\tilde{\mathbf{K}}\mathcal{M}^{-1} = \mathcal{M}\tilde{\mathbf{K}}\mathcal{M}$$
$$\mathbf{E} = \mathcal{M}^{-1}\tilde{\mathbf{E}} = \mathcal{M}\tilde{\mathbf{E}}$$

with the second equalities following from the orthorgonality *and* symmetry of $\mathcal{M}$.

The symmetric $\mathcal{M}$ was constructed under the assertation that only a single direction would describe the motion, but this does not mean that we will always *enforce* a single direction of motion. When only one direction is enforced, the dynamics Eq. (1) determine the motion in the reciprocal basis direction. This is desirable on $\partial\Omega_S$ but not on $\partial\Omega_R$, on which all motion should be specified precisely.

Analagous to Eq. (6), we can write the finite element solution in terms of the new basis as

$$\mathbf{u}\big|_{\mathcal{T}} = \sum_{k=1}^{2N} \tilde{\alpha}_k \tilde{\boldsymbol{\varphi}}_k(\mathbf{x})$$

If, for example, $\mathbf{x}_k \in \partial\Omega_R$, then $\mathbf{x}_k$ should move only along $\mathbf{p}_k$. Consequently, we can set the boundary condition exactly by equating this specification with the corresponding degrees of freedom:

$$\begin{aligned} \tilde{\alpha}_k &= \Delta s(\mathbf{x}_k) \\ \tilde{\alpha}_{k-N} &= 0 \end{aligned}$$

Ordinarily, when degrees of freedom are set in this manner, the corresponding variables are removed from the system and references to those quantities replaced by the predetermined value. Unfortunately, as discussed at the end of Section IV.A, this is unfeasible here because the above sets degrees of freedom in $\tilde{\boldsymbol{\varphi}}_k$, which are local shape functions. Instead, we exploit the nature of finite precision arithmetic. If we have

$$\varepsilon^{-1}a + b = \varepsilon^{-1}c$$

then $a \to c$ as $\varepsilon \to 0+$ if $\min\left\{\left|\frac{a}{b}\right|, \left|\frac{b}{a}\right|\right\} \ll \varepsilon^{-1}$. On a computer, by taking $\varepsilon^{-1} \sim 10^{15}$, for instance, it is possible to obtain $a \approx c$ to nearly machine precision. This technique is known as a penalty method [17].

To enforce the condition via the penalty method, we modify $\tilde{\mathbf{K}}$ and $\tilde{\mathbf{E}}$ by adding perturbations:

$$\begin{aligned} \tilde{\mathbf{K}} &\mapsto \tilde{\mathbf{K}} + \mathbf{B}_i \\ \tilde{\mathbf{E}} &\mapsto \tilde{\mathbf{E}} + \mathbf{b}_i \end{aligned}$$

The subscripts on $\mathbf{B}$ and $\mathbf{b}$ match the number of degrees of freedom being set (single or double enforcement) and will be specified in the next two subsections. With these modifications, the contributions from the element $\mathcal{T}$ to the global stiffness matrix $\mathcal{A}$ and right-hand vector $\mathcal{F}$ become, per Eq. (11),

$$\begin{aligned} \mathcal{A}\big|_{\mathcal{T}} &= \mathbf{K} \hookleftarrow \mathcal{M}(\tilde{\mathbf{K}} + \mathbf{B}_i)\mathcal{M} &\implies \quad \mathcal{A}\big|_{\mathcal{T}} &= \mathbf{K} + \hat{\mathbf{K}}_i & \hat{\mathbf{K}}_i &\triangleq \mathcal{M}\mathbf{B}_i\mathcal{M} \\ \mathcal{F}\big|_{\mathcal{T}} &= \mathbf{E} \hookleftarrow \mathcal{M}(\tilde{\mathbf{E}} + \mathbf{b}_i) &\implies \quad \mathcal{F}\big|_{\mathcal{T}} &= \mathbf{E} + \hat{\mathbf{E}}_i & \hat{\mathbf{E}}_i &\triangleq \mathcal{M}\mathbf{b}_i \end{aligned} \tag{16}$$

It is convenient to partition $\mathcal{M}$ along the lines of Eq. (15):

$$\mathcal{M} = \begin{bmatrix} \mathbf{P}_1 & \mathbf{0} & \mathbf{P}_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{P}_2 & \mathbf{0} & -\mathbf{P}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}$$

where $\mathbf{I}$ denotes the $(N-m)\times(N-m)$ identity matrix. The subscripts on $\mathbf{P}_1$ and $\mathbf{P}_2$ match the component (superscript) of $\mathbf{p}$ which supplies the matrix's entries:

$$\begin{aligned} \mathbf{P}_1 &= \operatorname{diag}\left[p_1^{(1)}, \ldots, p_m^{(1)}\right] \\ \mathbf{P}_2 &= \operatorname{diag}\left[p_1^{(2)}, \ldots, p_m^{(2)}\right] \end{aligned}$$

In the next two subsections, we detail explicitly Eq. (16) for $i = 1$ (single enforcement) and $i = 2$ (double enforcement). These will be used in Section IV.C as follows:

Corner Nodes	Double Enforcement
Inlaid Nodes: Receder	Double Enforcement
Inlaid Nodes: Slider	Single Enforcement

In the coming sections, we introduce the notation $\hat{s}$ to denote the recession at nodes. The physical value Δs is specified at quadrature points; $\hat{s}$ is a value computed by the tangle-free algorithm and may be nonphysical. The algorithm's recession values may differ from the physical values Δs.

1. *Single Enforcement*

When only one degree of freedom is set, the perturbations take the form

$$\mathbf{B}_1 \triangleq \varepsilon^{-1}\begin{bmatrix}\mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\end{bmatrix} \qquad \mathbf{b}_1 \triangleq \varepsilon^{-1}\begin{bmatrix}\hat{\mathbf{s}}(\mathbf{p}_1,\ldots,\mathbf{p}_m)\\ \mathbf{0}\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}$$

where the sizes of the blocks of $\mathbf{B}_1$ mirror those in Eq. (15); the number of rows in each block of $\mathbf{V}_1$ match the rows in Eq. (15) as well. The block $\hat{\mathbf{s}}(\mathbf{d}_1,\ldots,\mathbf{d}_m) \in \mathbb{R}^m$ is defined as

$$\hat{\mathbf{s}}(\mathbf{d}_1,\ldots,\mathbf{d}_m) \triangleq \begin{bmatrix}\hat{s}(\mathbf{x}_1;\mathbf{d}_1)\\ \vdots\\ \hat{s}(\mathbf{x}_m;\mathbf{d}_m)\end{bmatrix} \tag{17}$$

The inclusion of a direction $\mathbf{d}_i$ indicates that the recession value $\hat{s}(\mathbf{x}_i)$ is in the direction $\mathbf{d}_i$. Performing the multiplications blockwise, we have

$$\mathcal{M}\mathbf{B}_1\mathcal{M} = \varepsilon^{-1}\begin{bmatrix}\mathbf{P}_1^2 & \mathbf{0} & \mathbf{P}_1\mathbf{P}_2 & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{P}_2\mathbf{P}_1 & \mathbf{0} & \mathbf{P}_2^2 & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\end{bmatrix} \qquad \mathcal{M}\mathbf{b}_1 = \varepsilon^{-1}\begin{bmatrix}\mathbf{P}_1\hat{\mathbf{s}}(\mathbf{p}_1,\ldots,\mathbf{p}_m)\\ \mathbf{0}\\ \mathbf{P}_2\hat{\mathbf{s}}(\mathbf{p}_1,\ldots,\mathbf{p}_m)\\ \mathbf{0}\end{bmatrix}$$

so that obtain a nicely simplifed result for the corrections to the stiffness matrix and right-side vector:

$$\hat{\mathbf{K}}_1 = \frac{1}{\varepsilon}\left[\begin{array}{cccccc|cccccc}
\left[p_1^{(1)}\right]^2 & & & & & \Large 0 & p_1^{(1)}p_1^{(2)} & & & & & \Large 0\\
 & \ddots & & & & & & \ddots & & & & \\
 & & \left[p_m^{(1)}\right]^2 & & & & & & p_m^{(1)}p_m^{(2)} & & & \\
 & & & 0 & & & & & & 0 & & \\
 & & & & \ddots & & & & & & \ddots & \\
\Large 0 & & & & & 0 & \Large 0 & & & & & 0\\
\hline
p_1^{(1)}p_1^{(2)} & & & & & \Large 0 & \left[p_1^{(2)}\right]^2 & & & & & \Large 0\\
 & \ddots & & & & & & \ddots & & & & \\
 & & p_m^{(1)}p_m^{(2)} & & & & & & \left[p_m^{(2)}\right]^2 & & & \\
 & & & 0 & & & & & & 0 & & \\
 & & & & \ddots & & & & & & \ddots & \\
\Large 0 & & & & & 0 & \Large 0 & & & & & 0
\end{array}\right] \qquad \hat{\mathbf{E}}_1 = \frac{1}{\varepsilon}\begin{bmatrix}\hat{s}(\mathbf{x}_1)p_1^{(1)}\\ \vdots\\ \hat{s}(\mathbf{x}_m)p_m^{(1)}\\ \left.\begin{matrix}0\\ \vdots\\ 0\end{matrix}\right\}N-m\\ \hat{s}(\mathbf{x}_1)p_1^{(2)}\\ \vdots\\ \hat{s}(\mathbf{x}_m)p_m^{(2)}\\ \left.\begin{matrix}0\\ \vdots\\ 0\end{matrix}\right\}N-m\end{bmatrix}$$

2. Double Enforcement

When both degrees of freedom are set, we have

$$\mathbf{B}_2 \triangleq \varepsilon^{-1}\begin{bmatrix}\mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\end{bmatrix} \qquad \mathbf{b}_2 \triangleq \varepsilon^{-1}\begin{bmatrix}\hat{\mathbf{s}}(\mathbf{p}_1,\dots,\mathbf{p}_m)\\ \mathbf{0}\\ \hat{\mathbf{s}}(\mathbf{p}_1^\perp,\dots,\mathbf{p}_m^\perp)\\ \mathbf{0}\end{bmatrix}$$

The direction $\mathbf{p}_k^\perp$ is $\mathbf{q}_k$ of Eq. (14). Performing the multiplication to form $\hat{\mathbf{K}}_2$, we have

$$\hat{\mathbf{K}}_2 = \varepsilon^{-1}\begin{bmatrix}\mathbf{P}_1^2+\mathbf{P}_2^2 & \mathbf{0} & \mathbf{P}_1\mathbf{P}_2-\mathbf{P}_2\mathbf{P}_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{P}_2\mathbf{P}_1-\mathbf{P}_1\mathbf{P}_2 & \mathbf{0} & \mathbf{P}_2^2+\mathbf{P}_1^2 & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\end{bmatrix}$$

Because both $\mathbf{P}_1$ and $\mathbf{P}_2$ are diagonal, they commute, causing the off-diagonal blocks of $\hat{\mathbf{K}}_2$ to vanish. Furthermore, unit length of $\mathbf{p}_k$ means that the (i,i) entry of the on-diagonal blocks has value

$$\left[\mathbf{P}_1^2+\mathbf{P}_2^2\right]_{ii} = \left[p_i^{(1)}\right]^2 + \left[p_i^{(2)}\right]^2 = 1$$

so that we obtain an amazingly simple result for $\hat{\mathbf{K}}_2$ and slightly complicated one for $\hat{\mathbf{E}}_2$:

$$\hat{\mathbf{K}}_2 = \frac{1}{\varepsilon}\left[\begin{array}{cc|cc} \underbrace{\begin{matrix}1 & & \\ & \ddots & \\ & & 1\end{matrix}}_{m} & & \underbrace{\begin{matrix}0 & & \\ & \ddots & \\ & & 0\end{matrix}}_{m} & \\ & \begin{matrix}0 & & \\ & \ddots & \\ & & 0\end{matrix} & & \begin{matrix}0 & & \\ & \ddots & \\ & & 0\end{matrix} \\ \hline \begin{matrix}0 & & \\ & \ddots & \\ & & 0\end{matrix} & & \begin{matrix}1 & & \\ & \ddots & \\ & & 1\end{matrix} & \\ & \begin{matrix}0 & & \\ & \ddots & \\ & & 0\end{matrix} & & \begin{matrix}0 & & \\ & \ddots & \\ & & 0\end{matrix} \end{array}\right] \begin{array}{l} \}\, m \\ \\ \}\, m \\ \\ \end{array} \qquad \hat{\mathbf{E}}_2 = \frac{1}{\varepsilon}\begin{bmatrix}\left\langle \mathbf{p}_1, \hat{\mathbf{s}}(\mathbf{x}_1;\mathbf{p}_1,\mathbf{p}_1^\perp)\right\rangle\\ \vdots\\ \left\langle \mathbf{p}_m, \hat{\mathbf{s}}(\mathbf{x}_m;\mathbf{p}_m,\mathbf{p}_m^\perp)\right\rangle\\ 0\\ \vdots\\ 0\\ \left\langle \mathbf{p}_1^\perp, \hat{\mathbf{s}}(\mathbf{x}_1;\mathbf{p}_1,\mathbf{p}_1^\perp)\right\rangle\\ \vdots\\ \left\langle \mathbf{p}_m^\perp, \hat{\mathbf{s}}(\mathbf{x}_m;\mathbf{p}_m,\mathbf{p}_m^\perp)\right\rangle\\ 0\\ \vdots\\ 0\end{bmatrix}\begin{array}{l} \\ \\ \\ \left.\begin{matrix}\\ \\ \\ \end{matrix}\right\}N-m \\ \\ \\ \\ \left.\begin{matrix}\\ \\ \\ \end{matrix}\right\}N-m\end{array}$$

We reuse the previous notation Eq. (17) for compactness; for clarity, the entries of $\hat{\mathbf{E}}_2$ are of the form

$$\left\langle \mathbf{p}, \hat{\mathbf{s}}(\mathbf{x}^*;\mathbf{p},\mathbf{p}^\perp)\right\rangle = \hat{s}(\mathbf{x}^*;\mathbf{p})p^{(1)} + \hat{s}(\mathbf{x}^*;\mathbf{p}^\perp)p^{(2)}$$
$$\left\langle \mathbf{p}^\perp, \hat{\mathbf{s}}(\mathbf{x}^*;\mathbf{p},\mathbf{p}^\perp)\right\rangle = \hat{s}(\mathbf{x}^*;\mathbf{p})p^{(2)} - \hat{s}(\mathbf{x}^*;\mathbf{p}^\perp)p^{(1)}$$

C. Determining the Directions

The previous section supposes that the recession values and directions of motion are known *a priori*. This section describes how to determine them. First, we need to define a simple but critical concept:

Definition (Side set).

A *side set* is a subset of $\partial\Omega$. Each of $\partial\Omega_R$, $\partial\Omega_S$, and $\partial\Omega_F$ is the union of side sets.

How the directions of motion are constructed depend on the problem geometry and the location of the node. Side sets allow for quick and easy classification that eliminates the need for complex geometric detection algorithms. With their help, we reduce the analysis to two node types in two-dimensions:

Definition (2D Node Classification).

In two-dimensional meshes, a node is a *corner node* if it is contained in two side sets. It is an *inlaid node* if it is contained in only one side set.

It is important to note the distinction between our definition of a corner and the geometric notion of a corner. Automatic detection based on the latter is precarious, as it is easy for an algorithm to misclassify coarse portions of the mesh, which can occur even in well-constructed meshes after extensive recession.

1. Element-by-Element, Node-by-Node

Sections IV.B1 and IV.B.2 seemingly assume that all nodes in an element will have only a single type of enforcement. This is not necessarily the case, as an element may contain, for example, three boundary nodes, one of which is a corner, one of which lies on a sliding side, and one of which lies on a receding side. Fortunately, we see that both $\hat{\mathbf{K}}_1$ and $\hat{\mathbf{K}}_2$ are diagonal. Consequently, we can assemble the overall contribution to the stiffness matrix and right-side vector by taking the corresponding row from the appropriate correction matrix and vector. Beyond element-by-element assembly, this allows us to do node-by-node updates, negating the need to form a full contribution matrix and reducing bookkeeping.

2. Corner Nodes

In two dimensions, a node is contained in exactly two boundary faces (edge). It is possible for a node to be contained in an arbitrary number of *elements* but our requirement that intersection with the boundary be non-trivial excludes from consideration all but one or two. It is beneficial to realize that the elements themselves are not important—it is the edges that give us the critical information. We therefore simplify our analysis by disregarding the elements once we've identified the boundary edges.

The physical recession values Δs are known only at quadrature points along the edge, yet the results of the previous section require a value at the node itself. A potential solution is apparent:

1. Choose the value of Δs at the closest quadrature point for each edge.
2. Interpolate between these two values to construct an estimate for Δs at the node.

While seemingly sound, this method has a major deficiency: with only two values, we must use linear interpolation in step 2. Physically incorrect, this flattens any approximated curvature and is likely to lead to skewed geometry. Instead, the solution is, in essence, to move the entire edge at once:

1. For each face, compute the "new quadrature points" by moving Δs along $\boldsymbol{\nu}$, both of which are known *a priori*, and then compute the best fit line through these new points to determine the new edge.
2. Compute the intersection of the new edges and make that the new location of the node.
3. The motion $\hat{s}$ and direction vector $\mathbf{p}$ are obtained by subtracting the old location from the new.

This idea is realized in Algorithm 1 below. Further explanation of the steps follows the presentation.

Algorithm 1 (Corner Motion, 2D).

1. Let $\mathcal{E}$ be the edge containing the node $\mathbf{x}^*$ from one side set and $\mathcal{E}'$ be the other.
2. Let $\mathbf{z}_1, \ldots, \mathbf{z}_q$ be the quadrature points of $\mathcal{E}$. Let $\boldsymbol{\nu}_k$ and Δs_k be the normal and recession values, respectively, specified at the quadrature point $\mathbf{z}_k$. Let $\mathbf{z}'_1, \ldots, \mathbf{z}'_{q'}$, $\boldsymbol{\nu}'_k$, and $\Delta s'_k$ be the similarly named and corresponding quantities from $\mathcal{E}'$.
3. Compute the centroids of the sets $\{\mathbf{z}_k + \Delta s_k \boldsymbol{\nu}_k\}_{k=1}^{q}$ and $\{\mathbf{z}'_k + \Delta s'_k \boldsymbol{\nu}'_k\}_{k=1}^{q'}$:

$$\mathbf{c} \triangleq \frac{1}{q}\sum_{k=1}^{q}\left[\mathbf{z}_k + \Delta s_k \boldsymbol{\nu}_k\right] \qquad \mathbf{c}' \triangleq \frac{1}{q'}\sum_{k=1}^{q'}\left[\mathbf{z}'_k + \Delta s'_k \boldsymbol{\nu}'_k\right]$$

4. Form the $2 \times q$ matrix $\mathbf{A}$ and $2 \times q'$ matrix $\mathbf{A}'$

$$\begin{aligned}\mathbf{A} &\triangleq \left[\ \mathbf{z}_1 + \Delta s_1 \boldsymbol{\nu}_1 - \mathbf{c}\ \middle|\ \cdots\ \middle|\ \mathbf{z}_q + \Delta s_q\ \boldsymbol{\nu}_q - \mathbf{c}\ \right] \\ \mathbf{A}' &\triangleq \left[\ \mathbf{z}'_1 + \Delta s'_1 \boldsymbol{\nu}'_1 - \mathbf{c}'\ \middle|\ \cdots\ \middle|\ \mathbf{z}'_q + \Delta s'_{q'} \boldsymbol{\nu}'_q - \mathbf{c}'\ \right]\end{aligned}$$

(continued).

4. Compute the singular value decompositions of $\mathbf{A}$ and $\mathbf{A}'$:

$$\mathbf{A} = \mathbf{U\Sigma V} \qquad \mathbf{A}' = \mathbf{U}'\mathbf{\Sigma}'\mathbf{V}'$$

where $\mathbf{U} \in \mathbb{R}^{2\times 2}$, $\mathbf{\Sigma} \in \mathbb{R}^{2\times q}$, $\mathbf{V} \in \mathbb{R}^{q\times q}$, $\mathbf{U}' \in \mathbb{R}^{2\times 2}$, $\mathbf{\Sigma}' \in \mathbb{R}^{2\times q'}$, and $\mathbf{V}' \in \mathbb{R}^{q'\times q'}$. Let $\boldsymbol{\nu} \triangleq \mathbf{u}_3$ and $\bar{\boldsymbol{\nu}} \triangleq \mathbf{u}'_3$, the third columns of $\mathbf{U}$ and $\mathbf{U}'$, respectively.
5. Solve the linear system

$$\begin{bmatrix} -\nu^{(2)} & \bar{\nu}^{(2)} \\ \nu^{(1)} & -\bar{\nu}^{(1)} \end{bmatrix}\begin{bmatrix} s^* \\ t^* \end{bmatrix} = \mathbf{c}' - \mathbf{c}$$

and set $\mathbf{x}^*_{\text{new}} = \mathbf{c} + s^*\left[-\nu^{(2)}, \nu^{(1)}\right]$.
6. Set $\hat{s} = |\mathbf{x}^*_{\text{new}} - \mathbf{x}^*|$ and $\mathbf{p} = \hat{s}^{-1}\left(\mathbf{x}^*_{\text{new}} - \mathbf{x}^*\right)$.
7. Follow Section IV.B.2 with the above $\mathbf{p}$ and $\hat{s}$. Take $\hat{s}(\mathbf{x}^*; \mathbf{p}^\perp) = 0$.

The points $\tilde{\mathbf{z}}_k \triangleq \mathbf{z}_k + \Delta s_k \boldsymbol{\nu}_k$ are the "new quadrature points," merely the result of applying the motion at each quadrature point. They will not necessarily be the quadature points of the new edge. In fact, the $\tilde{\mathbf{x}}_k$ need not be colinear if $q > 2$. As a result, the new edge is determined by best-fit $\mathcal{B}$, mathematically

$$\mathcal{B} = \operatorname*{arg\,min}_{\dim \mathcal{S}=1} \sum_{k=1}^{q} |\tilde{\mathbf{z}}_k - \Pi_{\mathcal{S}}(\tilde{\mathbf{z}}_k)|$$

where $\Pi_{\mathcal{S}} : \mathbb{R}^2 \to \mathcal{S}$ denotes the orthogonal projection operator for the subspace $\mathcal{S}$. The following well-known result is of considerable use in helping us solve this problem:

Lemma 1.

The least-squares best-fit plane of $\{\mathbf{z}_k\}_{k=1}^{N} \subset \mathbb{R}^d$ contains the centroid $\mathbf{c} = \frac{1}{N}\sum_{k=1}^{N} \mathbf{z}_k$.

We seek the "best one-dimensional subspace" spanned by the columns of $\mathbf{A}$. To find this subspace, we call upon the singular value decomposition

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$$

where $\mathbf{U} \in \mathbb{R}^{2\times 2}$ and unitary, $\mathbf{V} \in \mathbb{R}^{q\times q}$ and unitary, and $\boldsymbol{\Sigma}_k \in \mathbb{R}^{2\times q}$ of the form

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_r & \mathbf{0}_{r\times(q-r)} \\ \mathbf{0}_{(2-r)\times r} & \mathbf{0}_{(2-r)\times(q-r)} \end{bmatrix} \qquad \boldsymbol{\Sigma}_r = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}$$

in which $r = \operatorname{rank}\mathbf{A}$ and $\sigma_1 \geq \cdots \geq \sigma_r > 0$. This second lemma further aids us in our quest:

Lemma 2.

The first s columns of $\mathbf{U}$ represent the "biggest" s-dimensional subspace, the one that has the smallest representation error in Euclidean norm, contained in range($\mathbf{A}$).

The proof of Lemma 2 can be found in [8]. These two lemma explain the mysterious subtraction of $\mathbf{c}$ from each column of $\mathbf{A}$ and $\mathbf{c}'$ from $\mathbf{A}'$. First, we note that range($\mathbf{A}$) = range$\big[\mathbf{u}_1 \cdots \mathbf{u}_r\big]$:

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{U}\Big[\sigma_1\mathbf{V}^T\mathbf{x} \quad \cdots \quad \sigma_r\mathbf{V}^T\mathbf{x} \quad \mathbf{0} \quad \cdots \quad \mathbf{0}\Big] \\ &= \sum_{i=1}^{r} \sigma_i \langle \mathbf{x}, \mathbf{v}_i \rangle \mathbf{u}_i \end{aligned}$$

Since the centroid is contained in the best-fit plane, by subtracting it from each data point creates a candidate for the best-fit line we seek. The task therefore becomes to find a single vector that best encapsulate the columnspaces of $\mathbf{A}$ (and similarly for $\mathbf{A}'$), a problem to which we can directly apply Lemma 2.

The above machinery explains how steps 1-5 of Algorithm 1 compute the new edges; we have $\mathcal{E} \mapsto (\mathbf{c}, \boldsymbol{\nu})$ and $\mathcal{E}' \mapsto (\mathbf{c}', \hat{\boldsymbol{\nu}})$, where the pairing indicates the normal vector and point inside the new edge "plane." From the contents of these pairings we can write parametric expressions for the new edges:

$$\left.\begin{aligned} \mathcal{E} \mapsto \ell(s) &= \mathbf{c} + s\big[-\nu^{(2)}, \nu^{(1)}\big]^T \\ \mathcal{E}' \mapsto \ell'(t) &= \mathbf{c}' + t\big[-\bar{\nu}^{(2)}, \bar{\nu}^{(1)}\big]^T \end{aligned}\right\} \quad (18)$$

Step 6, therefore, computes the intersection of ℓ and ℓ' in terms of the above parameterization and sets the location of $\mathbf{x}^*_{\text{new}}$ to be the point $\ell(s^*) = \ell'(t^*)$. As promised, the direction of motion $\mathbf{p}$ is the vector that takes us from the original location to $\mathbf{x}^*_{\text{new}}$; normalizing this vector and saving the original length gives us $\hat{s}$. Since this completely describes the desired displacement, we quash movement in the orthogonal complement.

While Figure 4 above depicts the case of a double receder, the algorithm cares not for the specific combination of sliding and receding edges. When one side is fixed, Algorithm 1 will not display the correct behavior when the other side is receding; in implementation, it is advantageous to detect the presence of a fixed side set and immediately suppress all motion, eliminating the unnecessary overhead of the algorithm. When both sides are sliding, the end result is that the node will remain in its original location, so including logic to process that case quickly is also a boon to speed, although Algorithm 1 will also produce this effect.

3. Inlaid Nodes: Receders

In the corner case, it was possible for the node to be contained in either one or two elements. For an inlaid node, there are always exactly two elements. We can exploit this guarantee to simplify our analysis slightly.

To recede edges, we use the same technique of computing a best-fit plane through post-recession quadrature points. However, the intersection between the new edge planes may not be defined. For a flat boundary (such

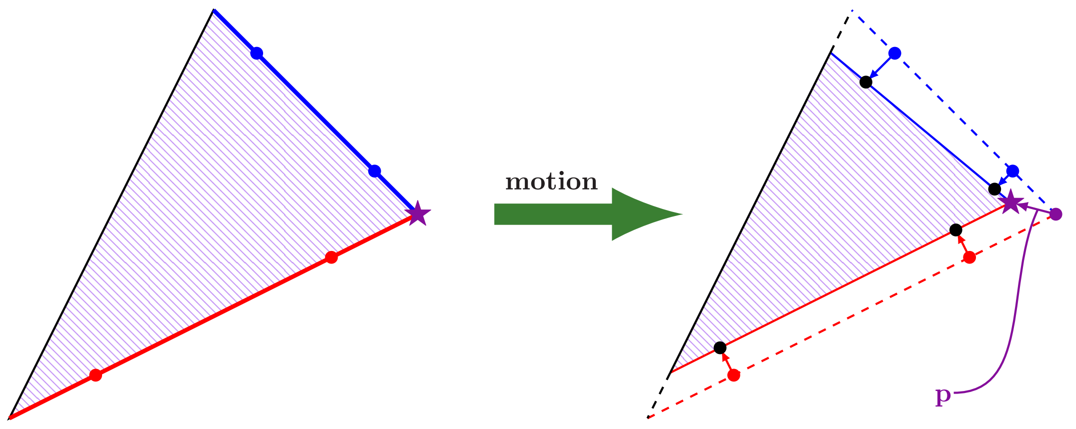


Figure 4. Illustration of Algorithm 1. The left shows an element with two receding sides (in red and blue) with the quadrature points on each edge. The corner node $\mathbf{x}^*$ is the purple star. Algorithm 1 computes the receded edges independently and then finds the location of the new node by intersecting the lines formed by a best-fit plane through the perturbed quadrature points (on left). Note that recession need not be uniform across an edge, as shown here.

the side of a square), the edges will be parallel, and if the recession is uniform, will remain so. Consequently, the linear solve will fail because the system matrix (Step 6 of Algorithm 1) will be singular.

To remedy this situation, we use the internal edge from the element containing the receding edges. The internal edge is guaranteed to be unique and the intersection between it and the post-recession edge necessarily exists. Furthermore, because there are always two elements, the computational burden is low. We end up with two candidates for the new node location and resolve the situation by averaging the two. From there, we proceed as before. Algorithm 2 below explicitly defines the process.

Algorithm 2 (Inlaid Receder Motion, 2D).

1. Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be the boundary edges containing $\mathbf{x}^*$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the elements which contain these edges.
2. Let $\mathbf{z}_{k,1}, \dots, \mathbf{z}_{k,q_k}$ be the quadrature points of $\mathcal{E}_k$. Let $\boldsymbol{\nu}_{k,i}$ and $\Delta s_{k,i}$ be the normal and recession values, respectively, specified at the quadrature point $\mathbf{z}_{k,i}$.
3. Set $k = 1$. Compute the centroid of the set $\left\{\mathbf{z}_{k,i} + \Delta s_{k,i}\boldsymbol{\nu}_{k,i}\right\}_{i=1}^{q_k}$:
$$\mathbf{c}_k \triangleq \frac{1}{q_k}\sum_{i=1}^{q_k}\left[\mathbf{z}_{k,i} + \Delta s_{k,i}\boldsymbol{\nu}_{k,i}\right]$$
4. Form the $2 \times q_k$ matrix $\mathbf{A}$
$$\mathbf{A}_k \triangleq \left[\begin{array}{c|c|c} \mathbf{z}_{k,1} + \Delta s_{k,1}\boldsymbol{\nu}_{k,1} - \mathbf{c}_k & \cdots & \mathbf{z}_{k,q_k} + \Delta s_{k,q_k}\boldsymbol{\nu}_{k,q_k} - \mathbf{c}_k \end{array}\right]$$
5. Compute the singular value decomposition of $\mathbf{A}$:
$$\mathbf{A} = \mathbf{U}_k\boldsymbol{\Sigma}_k\mathbf{V}_k$$
where $\mathbf{U}_k \in \mathbb{R}^{2\times 2}$, $\boldsymbol{\Sigma}_k \in \mathbb{R}^{2\times q_k}$, and $\mathbf{V}_k \in \mathbb{R}^{q_k\times q_k}$. Let $\boldsymbol{\nu}_k \triangleq \mathbf{u}_3$, the third column of $\mathbf{U}_k$.
6. Let $\mathcal{E}'_k$ be the internal (non-boundary) edge that contains $\mathbf{x}^*$ of $\mathcal{T}_k$. Let $\bar{\boldsymbol{\nu}}_k$ and $\mathbf{c}'_k$ be the normal and midpoint of $\mathcal{E}'_k$, respectively.
7. Solve the linear system
$$\begin{bmatrix} -\nu_k^{(2)} & \bar{\nu}_k^{(2)} \\ \nu_k^{(1)} & -\bar{\nu}_k^{(1)} \end{bmatrix}\begin{bmatrix} s^* \\ t^* \end{bmatrix} = \mathbf{c}'_k - \mathbf{c}_k$$
and set $\mathbf{x}_{\text{new}}^k = \mathbf{c}_k + s^*\left[-\nu_1^{(2)}, \nu_1^{(1)}\right]$.
8. Repeat Steps 2-7 for $k = 2$.
9. Set $\mathbf{x}_{\text{new}}^* = \frac{1}{2}\left[\mathbf{x}_{\text{new}}^1 + \mathbf{x}_{\text{new}}^2\right]$. Set $\hat{s} = |\mathbf{x}_{\text{new}}^* - \mathbf{x}^*|$ and $\mathbf{p} = \hat{s}^{-1}\left(\mathbf{x}_{\text{new}}^* - \mathbf{x}^*\right)$.
10. Follow Section IV.B.2 with the above $\mathbf{p}$ and $\hat{s}$. Take $\hat{s}(\mathbf{x}^*; \mathbf{p}^\perp) = 0$.

Steps 1-5 are identical to those of Algorithm 1 and do nothing more than compute the "new edge" after recession. In Step 6, we identify the internal edge that contains $\mathbf{x}^*$ of the element of which $\mathcal{E}_k$ is the boundary edge. To proceed, we need this edge's normal vector and midpoint. The midpoint is merely a matter of convenience because it is easy to compute; any point on the edge will work in practice.

Step 7 is similar to Step 6 of Algorithm 1 in that it computes the intersection of lines. In this case, the two lines ℓ and ℓ', identical to those of Eq. (18) except with subscript k on every ν symbol, represent the post-recession edge and the stationary internal edge $\mathcal{E}'_k$.

Finally, after repeating the process for the other edge, we can compute the new node location by averaging (here, the centroid of) the two resultant points, normalizing, and storing the original length to $\hat{s}$. Again, $\mathbf{p}$ describes all the motion we desire, so we use double enforcement with zero motion in the direction of $\mathbf{p}^\perp$.

4. Inlaid Nodes: Sliders

A slider is characterized by forbidden motion in the normal direction. As elements are polygonal, the computational boundary is only Lipschitz, which means that the normal is only defined almost everywhere.

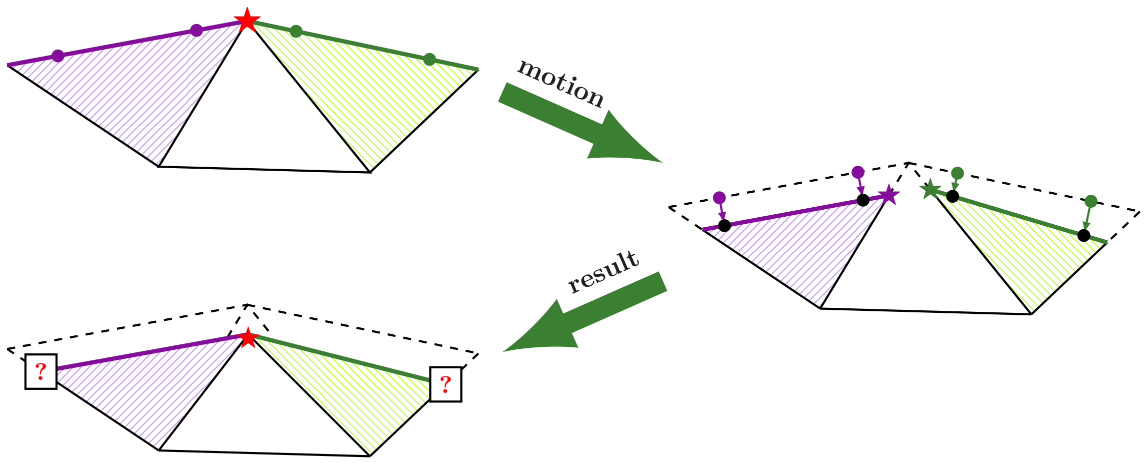


Figure 5. Illustration of Algorithm 2. In the upper left, the purple and green elements have receding edges (bold, colored edge); the red star marks $\mathbf{x}^*$ and the circles mark the quadrature points. As in the corner case, we form the "new face" by moving the quadrature points by the corresponding values of Δs along the normals and computing the best fit, shown as the solid colored lines. The stars now indicate the intersection of this receded edge with the internal edge from that element. The lower left is the final result: the red star indicates the new node location, the centroid of the purple and green stars of the previous step. The locations of the other nodes on these edges, marked with a '?,' are *not* set here. The true new edge can only be determined after completing this process for every node on the receding side set.

The set of points where it is not defined is precisely the set of nodes—exactly where it is needed.

Because edges are straight lines between nodes, the normal is the same everywhere on a given edge. If only a single element is considered, the normal at the node can be taken to be this value. Unfortunately, we must consider both elements that contain the node. The question of which to choose has no natural answer: should we pick at random? Always choose the one on the "right"? Choose the one of the element larger in volume? Mathematically, the answer requires the utmost care. Because the finite element basis functions are constructed from shape functions, the same vector must be selected for the shape function for each element containing $\mathbf{x}^*$ to preserve the continuity of the basis function.

The solution is to combine the two. Let $\boldsymbol{\nu} \triangleq \boldsymbol{\nu}(\mathbf{z}; \mathcal{T})$ and $\boldsymbol{\nu}' \triangleq \boldsymbol{\nu}(\mathbf{z}'; \mathcal{T}')$ be the well-defined normals at the quadrature point $\mathbf{z} \in \mathcal{T}$ and $\mathbf{z}' \in \mathcal{T}'$ each closest to $\mathbf{x}^*$. We then define the normal at $\mathbf{x}^*$ to be the "average"

$$\boldsymbol{\nu}^* = \frac{\boldsymbol{\nu} + \boldsymbol{\nu}'}{|\boldsymbol{\nu} + \boldsymbol{\nu}'|} \tag{19}$$

Determining the normal is the biggest challenge for inlaid sliders. There is no recession, only a forbidden direction, so there are no new edges to compute, no intersections to find. All we need do is enforce a single no-motion condition. In stark contrast to the previous two cases, the algorithm is the simplest yet:

Algorithm 3 (Inlaid Slider Motion, 2D).

1. Let $\mathcal{E}$ and $\mathcal{E}'$ be the boundary edges containing $\mathbf{x}^*$.
2. Let $\mathbf{z}$ be any quadrature point of $\mathcal{E}$; $\mathbf{z}'$, of $\mathcal{E}'$. Let $\boldsymbol{\nu}$ and $\boldsymbol{\nu}'$ be the normals, respectively, defined at these points. Set
$$\mathbf{p} = \frac{\boldsymbol{\nu} + \boldsymbol{\nu}'}{|\boldsymbol{\nu} + \boldsymbol{\nu}'|}$$
3. Follow Section IV.B.1 with the above $\mathbf{p}$ and $\hat{s}(\mathbf{x}^*) = 0$.

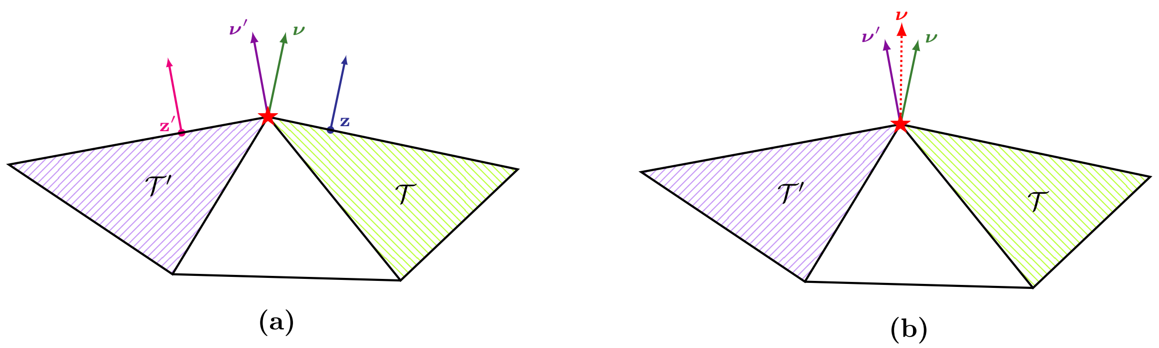


Figure 6. Normal at a Node. (a) depicts the process of generating "one-sided" normal vectors for the element containing the node (starred): we simply choose any point that isn't a node on the edge. (b) shows how the two are combined to a single vector via Eq. (19)**.**

D. Verification

The hallmark of mesh motion problems is domains that evolve in time with solutions. The set of problems with moving boundaries possessing known analytic solutions is a small one; instead, we can construct them from members of the slightly-larger set of thermal problems with known analytic solutions. The veracity of the mesh motion scheme presented in this paper will be established by the following procedure:

1. A problem with $\dot{s} \equiv$ constant specified. This will verify second order convergence of the temporal and spatial integrators, including the mesh convective terms, of the thermal response code `CHAR`.
2. An infinite cylinder modeled via axisymmetric 1D domain with a time varying $\dot{s}$ specified. This example will feature temporal convergence lower than second order. The unambiguity of one-dimension allows us to attribute the reduction to `CHAR`'s handling of nonconstant ablation rather than mesh motion.
3. The above with specified recession replaced by an equivalent heat flux condition. This will authenticate `CHAR`'s implementation of such boundary conditions, used extensively in the demonstration problems.
4. The full two-dimensional version of the cylinder. As the grid is refined in the radial direction, the numerical solution converges to the solution to the above at first order.

The numerics of thermal response are handled by `CHAR`, a code developed in the Applied Aeroscience and CFD branch at NASA's Johnson Space Center. The mesh motion algorithms described in this paper have become a permanent part of this software package. For an overview of the extensive capabilities of `CHAR`, see [1] and [16]. The PDE of choice for mesh motion in `CHAR` is linear elasticity, which has operator $\mathcal{L}$

$$\mathcal{L}\mathbf{u} \triangleq -\nabla\lambda(\nabla\cdot\mathbf{u}) - (\nabla\cdot\mu\nabla)\mathbf{u} - \nabla\cdot\mu(\nabla\mathbf{u})^T$$

In general, the homogenous case $\mathbf{f} \equiv \mathbf{0}$ is the only one of practical interest. The variational problem corresponding to Eq. (1) is

$$a(\mathbf{u},\mathbf{v}) = \langle\mathbf{f},\mathbf{v}\rangle_{L^2(\Omega;\mathbb{R}^d)} \quad \forall\,\mathbf{v}\in H^1(\Omega;\mathbb{R}^d)$$

The bilinear form corresponding to $\mathcal{L}$ is

$$\begin{aligned} a(\mathbf{u},\mathbf{v}) = \lambda\int_\Omega(\nabla\cdot\mathbf{u})(\nabla\cdot\mathbf{v})\,d\mathbf{x} \quad &+\mu\int_\Omega\langle\nabla\mathbf{u}^T,\nabla\mathbf{v}\rangle_{\mathbb{R}^{d\times d}}\,d\mathbf{x} + \mu\int_\Omega\langle\nabla\mathbf{u},\nabla\mathbf{v}\rangle_{\mathbb{R}^{d\times d}}\,d\mathbf{x} \\ \lambda\int_{\partial\Omega}(\nabla\cdot\mathbf{u})\langle\mathbf{v},\boldsymbol{\nu}\rangle_{\mathbb{R}^d}\,dS &- \mu\int_{\partial\Omega}\langle(\nabla\mathbf{u})^T\mathbf{d},\mathbf{v}\rangle_{\mathbb{R}^d}\,dS - \mu\int_{\partial\Omega}\langle(\nabla\mathbf{u})\mathbf{d},\mathbf{v}\rangle_{\mathbb{R}^d}\,dS \end{aligned} \tag{20}$$

where we use the standard inner products

$$\langle\mathbf{A},\mathbf{B}\rangle_{\mathbb{R}^{d\times d}} = \mathrm{trace}(\mathbf{A}^T\mathbf{B}) \quad = \sum_{i=1}^d\sum_{j=1}^d A_{ij}B_{ji}$$

$$\langle\mathbf{f},\mathbf{g}\rangle_{L^2(\Omega;\mathbb{R}^d)} = \int_\Omega\langle\mathbf{f},\mathbf{g}\rangle_{\mathbb{R}^d}\,d\mathbf{x} = \sum_{i=1}^d\int_\Omega f_i(\mathbf{x})g_i(\mathbf{x})\,d\mathbf{x}$$

Because the sole boundary condition in the problem is of Dirichlet type and will be enforced as an essential boundary condition, we require $\mathbf{v} \in H_0^1(\Omega; \mathbb{R}^d)$ so that the surface integrals of Eq. (20) all vanish. It is important to note that the implementation in `CHAR` of the linear elastic equations is used solely for mesh motion and its use has no relation to physical stress or strain. In this setting, the parameters λ and μ may be varied to tailor the mesh deformation and have no connection to any physical material.

As part of the verification, we will successively refine the time step Δt and characteristic element size Δh in order to show order of convergence. We assume the error due to discretization is of the form

$$\mathcal{E} = \alpha(\Delta t)^p + \beta(\Delta h)^q + \text{h.o.t} \tag{21}$$

By choosing a time step $(\Delta t)^p \ll (\Delta h)^q$, the temporal discretization error (second term) can be neglected and the order of convergence in space can be obtained. A similar procedure can be followed to obtain the order of convergence in time. Alternatively, we can verify both simultaneously if we have some intuition about p and q (here, we expect $p = q = 2$). We can then refactor Eq. (21), dropping the higher order terms:

$$\mathcal{E} = \left(1 + \frac{\alpha(\Delta t)^p}{\beta(\Delta h)^q} + 1\right)\beta(\Delta h)^q$$

If $\frac{\alpha(\Delta t)^p}{\beta(\Delta h)^q} \equiv$ constant, then the order of convergence q can be then be verified by working in log-log space:

$$\log \mathcal{E} = q\log(\Delta h) + \underbrace{\log\left[\beta\left(1 + \frac{\alpha(\Delta t)^p}{\beta(\Delta h)^q}\right)\right]}_{\triangleq\, b} \tag{22}$$

so long as the intercept $b \gg q\log(\Delta h)$. By determining the slope of the best-fit line through the points $(\log \Delta h, \log\,(\text{simulation error}))$ for several levels of refinement, we can confirm q. While exceptionally efficient because separate refinement studies for time and space are not required, this method cannot demarcate differences from the computed slope from the expected q into temporal and spatial error. However, if refinement in space only fails to decrease the error, then the source must be the time step, and vice versa.

In general finite element analysis, Δh is given as the maximum side length of an element [11, p.28]. If the grid is carefully constructed and the problem geometry simple or one-dimensional—as it shall in the coming verification problems, Δh can be replaced by the number of elements along a line N_e:

$$\Delta h = \frac{L}{N_e}$$

so that the error takes the form

$$\mathcal{E} = \left(\frac{\alpha(\Delta t)^p}{\beta(\Delta h)^q} + 1\right)\beta L^q N_e^{-q}$$
$$\Longrightarrow \log \mathcal{E} = -q\log N_e + \log\left[\beta L^q\left(1 + \frac{\alpha(\Delta t)^p}{\beta(\Delta h)^q}\right)\right]$$

or that the slope is negated and the intercept $b \mapsto b + q\log L$.

1. *Validation of `CHAR`: Constant Specified Melt*

We open with a problem adapted from the collected works of Benjamin Blackwell. Consider a slab of material occupying the right half-space $\mathcal{H} = \{\mathbf{x} \in \mathbb{R}^2 : x^{(1)} \geq 0\}$, subject to a constant surface temperature on $\partial\mathcal{H} = \{\mathbf{x} \in \mathbb{R}^2 : x^{(1)} = 0\}$. If we assume all relevant properties are constant, the symmetry in the $x^{(2)}$ direction permits reduction to a one dimensional, time-independent equation:

$$\begin{gathered} k\frac{\partial^2 T}{\partial \hat{x}^2} + \rho c \dot{s}\frac{\partial T}{\partial \hat{x}} = 0 \\ T(\hat{x} = 0, \cdot\,) = 800 \\ T(\hat{x} = \infty, \cdot\,) = 300 \end{gathered} \tag{23}$$

in which $\hat{x}$ is the $x^{(1)}$ coordinate such that $\hat{x} = 0$ is the terminal face of the half-plane. We can easily obtain a simple analytic solution to Eq. (23):

$$T_{\text{exact}}(\hat{x}) = 300 + 500e^{-\dot{s}\hat{x}/\alpha}$$

in which $\alpha = \frac{k}{\rho c}$. It is important to note that $\hat{x}$ is a *moving* frame of reference; it is casting the equation in this changing variable that enables the above representation. When we solve the problem with CHAR, we solve instead an unsteady problem in which the time dependence tracks the location of $\hat{x} = 0$. To compare the numerical solutions with moving grids to the above analytic solution, we must evaluate at

$$\hat{x} = x^{(1)} - \underbrace{\min_{\mathbf{x} \text{ a node}} x^{(1)}}_{x_{\text{surf}}}$$

To implement numerically, we must choose a finite thickness L_0 for the slab. Taking a small material diffusivity α allows choice of a modest value of L_0 without violating the semi-infinite assumption. The material property parameters used for the simulation are listed in the table below.

Material density	ρ	2000 kg/m³	Material diffusivity	α	1.0×10^{-7} m²/s
Material specific heat	c	1000 J/kg-K	Specified recession	$\dot{s}$	4.0×10^{-4} m/s
Material conductivity	k	0.2 W/m-K	Slab thickness	L_0	0.03 m

The error metric is the root mean-square error of the temperature field over the set of nodes $\{\mathbf{x}_i\}_{i=1}^{N}$:

$$\mathcal{E}_{\text{rms}}(t) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\big(T_{\text{exact}}(\hat{x}_i) - T_{\text{num}}(\mathbf{x}_i, t)\big)^2} \tag{24}$$

where T_{num} is the numerically-obtained solution to the unsteady melting problem

$$\begin{aligned} T_t - \alpha\nabla^2 T = 0 \quad & \text{in } \Omega(t) \times \{t : t > 0\} \\ T = T_{\text{exact}}(x) \quad & \text{in } \Omega(0) \times \{t = 0\} \\ T \equiv 800 \quad & \text{on } \{\mathbf{x} : x^{(1)} = x_{\text{surf}}(t)\} \times \{t : t > 0\} \\ \frac{\partial T}{\partial \boldsymbol{\nu}} \equiv 0 \quad & \text{on } \partial\Omega(t) \backslash \{\mathbf{x} : x^{(1)} = x_{\text{surf}}(t)\} \times \{t : t > 0\} \end{aligned} \tag{25}$$

in which $x_{\text{surf}}(t) = t\dot{s}$ so that $\Omega(t) = [x_{\text{surf}}(t), L_0] \times [0, L_0]$. Note that when the domain is truly infinite, the Neumann condition becomes vacuous as $\partial\Omega(t) = \{\mathbf{x} : x_{\text{surf}}(t) = 0\}$ and we have

$$\nabla^2 T = \frac{\partial T^2}{\partial \hat{x}^2} \qquad\qquad T_t = \frac{\partial T}{\partial t} = \frac{\partial T}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial t} = -\dot{s}\frac{\partial T}{\partial \hat{x}}$$

so that Eq. (25) becomes exactly Eq. (23) since $T(\hat{x} = \infty, \cdot) = 300$ is automatic from the initialization and Lebesgue's Dominated Convergence Theorem applied to the fundamental solution to the heat equation.

For the simulation, the initial mesh consists of rectangles of width L_0/N_e and height L_0 so that there are N_e uniformly-spaced elements in the $x^{(1)}$ direction and only one in the $x^{(2)}$ direction. An example is shown in Figure 7 below. Because of the constant specified ablation boundary condition, this uniformity is preserved for all time, although this is not guaranteed by the mesh motion scheme in general.

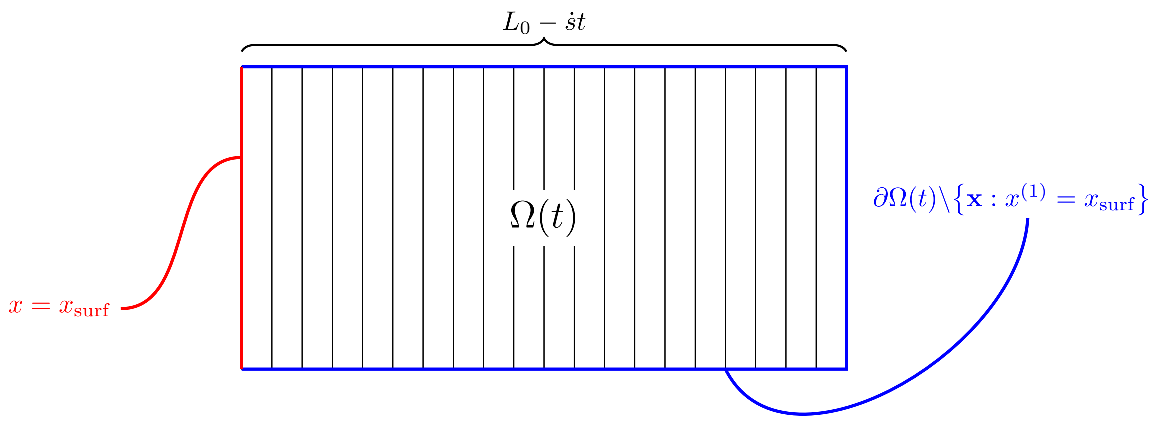

$L_0 - \dot{s}t$
$\Omega(t)$
$\partial\Omega(t)\backslash\{\mathbf{x} : x^{(1)} = x_{\text{surf}}\}$
$x = x_{\text{surf}}$

Figure 7. Specified-Melt Finite Element Mesh. This figure shows an example finite element domain $\Omega(t)$. Although not appearing so due to a non-unitary scaling aspect ratio, the height of the rectangle is also L_0 (and remains so for all time), but in practice, this is not important.

We perform three levels of refinement with the following time steps and grid spacings:

	Δt	N_e
Level 1	0.500	50
Level 2	0.250	100
Level 3	0.125	200

At the first time step, `CHAR` always uses a first-order time integrator. The large Δt means there is a significant initial error that takes some time to dissipate. However, this does not cause loss of second-order convergence. Table 2 below lists the error $\mathcal{E}_{\text{rms}}(t)$ in the temperature field at various common values of t.

	t									
	2.0	**4.0**	**6.0**	**8.0**	**10.0**	**12.0**	**14.0**	**16.0**	**18.0**	**20.0**
Level 1	8.1253	11.1144	11.2470	10.8870	10.4247	9.9402	9.4543	8.9732	8.4990	8.0327
Level 2	3.6338	2.9709	2.7085	2.5471	2.4157	2.2950	2.1796	2.0676	1.9583	1.8517
Level 3	2.1760	1.0126	0.7317	0.6338	0.5847	0.5507	0.5219	0.4953	0.4698	0.4450

Table 2. Temperature Error at Various Times. This table shows the unnormalized RMS errors in the temperature field computed using Eq. (24)**. If we normalize by dividing by average temperature over the entire domain, the error declines in time from about 4% to 2.6% at Level 1, from 1.4% to 0.6% at Level 2, falling below 0.2% at the end of Level 3.**

	t									
	2.0	**4.0**	**6.0**	**8.0**	**10.0**	**12.0**	**14.0**	**16.0**	**18.0**	**20.0**
$\boldsymbol{m}$	-0.9504	-1.7281	-1.9711	-2.0512	-2.0780	-2.0870	-2.0895	-2.0896	-2.0886	-2.0870

Table 3. Slopes of log-log Plots (Expanded). This table shows the actual values of the slope m in the best-fit equation $\log \mathcal{E}_{\textbf{rms}} = m \log N_e + b$ for the entries of Table 2.

Of primary interest to us is the order of convergence as described in the procedure at the close of D. Figure 8 plots in log-log space the number of elements against the temperature field error (values taken from Table 2) for several values of t. In addition, Table 3 lists the numerical values of the slopes of best-fit lines of

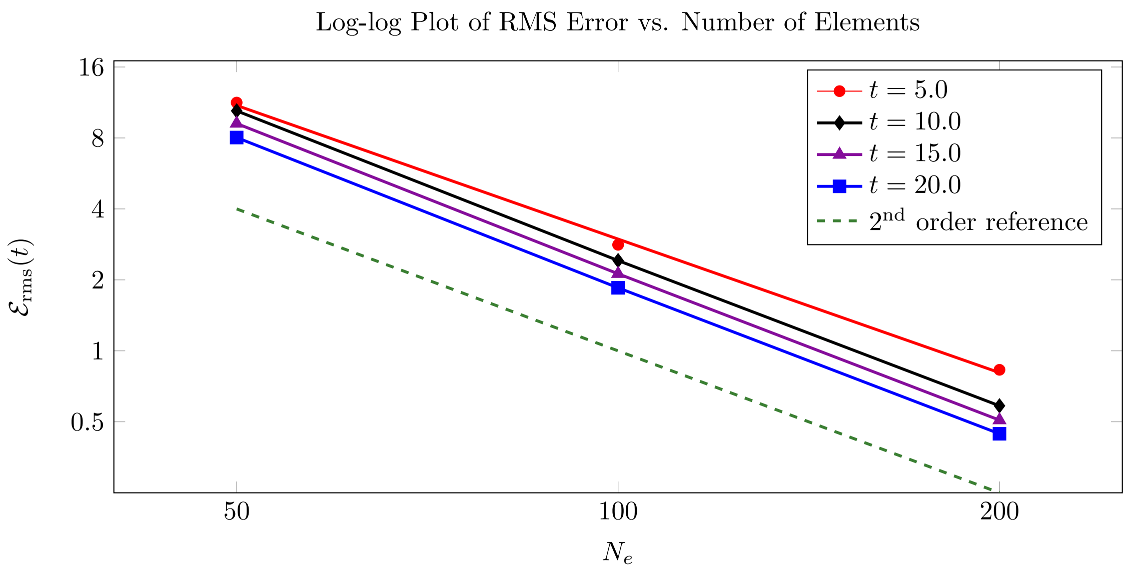


Figure 8. Log-log Plots of Temperature Field Error. Show for four values of t are plots of the information from Table 2 along with a reference line that has the expected slope of -2.

log-log plots of the values in Table 2. These diagrams show that `CHAR` exhibits the second order convergence in time, space, and mesh convective terms that is claimed, establishing the legitimacy of the thermal solver. As the tables show, this exercise establishes that in the presence of constant mesh motion, the thermal solver `CHAR` produces solutions with second-order accuracy, convincing us of its correct implementation.

2. *Validation of `CHAR`: Time-Varying Specified Melt*

This section will show that variable mesh motion reduces the order of convergence. To do so, we will consider an one-dimensional axisymmetric cylinder with known analytic solution. One-dimensional mesh motion eliminates ambiguity in the direction and magnitude and removes possibility of tangle or troubles.

Consider an infinitely long cylinder of radius R initially at temperature T_0. If suddenly placed in a convective environment at temperature T_∞, its temperature will rise rapidly rising until it matches the ambient. Because of azimuthal (rotational) symmetry, the equation describing this situation is simple in polar coordinates:

$$\left.\begin{array}{ll} \rho c \dfrac{\partial T}{\partial t} = \dfrac{k}{r}\dfrac{\partial}{\partial r}\left[r\dfrac{\partial T}{\partial r}\right], r \in [0,R], t > 0 & \\ T(r,0) = T_0 & \\ T(R,t) = T_\infty & \dfrac{\partial T}{\partial r}(0,\cdot) \equiv 0 \end{array}\right\} \tag{26}$$

The problem parameters are

Analytic radius	R	0.10526315789 m	Material density	ρ	1000 kg/m³
Initial temperature	T_0	300 K	Material conductivity	k	100 W/m-K
Ambient temperature	T_∞	1300 K	Material specific heat	c	500 J/kg-K

The exact solution to Eq. (26) is given by the series expansion

$$T(r,t) = T_\infty + 2(T_0 - T_\infty)\sum_{n=1}^{\infty} e^{-\alpha\lambda_n^2 t}\frac{J_0(\lambda_n r)}{R\lambda_n J_1(R\lambda_n)} \tag{27}$$

where J_k is the Bessel function of the first kind of order k. The values λ_n are the roots of

$$J_0(R\lambda_n) = 0 \tag{28}$$

To transform this into a mesh motion problem, we select a melt temperature $T_0 < T_{\text{melt}} < T_\infty$. We track the propogation of level set (isotherm) $T \equiv T_{\text{melt}}$ in time: for fixed t^*, we find r^* such that $T(r^*, t^*) = T_{\text{melt}}$ with T given by Eq. (27). By the maximum principle [7, p54], we must have $T(r, t^*) > T_{\text{melt}}$ for $r > r^*$. Consequently, we may consider this portion melted away so that $[0, r^*]$ becomes the new domain.

Choose $T_{\text{melt}} = 800$. Then the r^* that we seek is the solution to $F(r^*) = 0$, where F is the expression

$$F(r) = -\frac{1}{4} + \sum_{n=1}^{\infty} e^{-\alpha\lambda_n^2 t} \frac{J_0(\lambda_n r)}{R\lambda_n J_1(\lambda_n R)}$$

For each fixed t, the series converges uniformly in r by the M-Test (J_0 is bounded and $\lambda_n \to \infty$) so that

$$F'(r) = -\sum_{n=1}^{\infty} e^{-\alpha\lambda_n^2 t} \frac{J_1(\lambda_n r)}{R J_1(R\lambda_n)} \tag{29}$$

which gives all we need to use the Newton iteration $r_{k+1} = r_k - [F'(r_k)]^{-1} F(r_k)$. To compute this derivative, we have used the relationship $J_0'(r) = -J_1(r)$. A list of r^* for a small sample of t^* is below in Table 4.

t^*	r^*	t^*	r^*
0.00000	0.104565	6.00000	0.061091
0.14347	**0.100000**	7.50000	0.051801
1.50000	0.086854	9.00000	0.040139
3.00000	0.077620	10.50000	0.021925
4.50000	0.069391	11.10937	0.000000

Table 4. Time and Radius of 800 K Isotherm. This table lists values such that $T(r^*, t^*) = 800$. Of particular interest is the value of t^* for which $r^* = 0.1$. For $t > 11.10937$, $T > 800$ for all r.

We will again construct a specified ablation boundary condition so that divergence from second order cannot be attributed to an improperly implemented boundary condition. The recession rate $\dot{s}$ is given by

$$\dot{s} = \left.\frac{\partial r}{\partial t}\right|_{T=T_{\text{melt}}} \tag{30}$$

By considering Eq. (27) as $T(r, t) = f(r, t)$, we can find this quantity by considering the total derivative

$$\frac{\partial T}{\partial t} = \frac{\partial f}{\partial r}\frac{\partial r}{\partial t} + \frac{\partial f}{\partial t} \tag{31}$$

We can form the lefthand-side by rearranging Eq. (26) and using Eq. (29):

$$\frac{\partial T}{\partial t} = \frac{\alpha}{r}\left[\frac{\partial T}{\partial r} + r\frac{\partial^2 T}{\partial r^2}\right] = \frac{2\alpha(T_0 - T_\infty)}{r}\Big[F'(r) + rF''(r)\Big]$$

To compute the required second derivative, we utilize the relationship $J_k'(z) = \frac{1}{2}(J_{k-1}(z) - J_{k+1}(z))$ so that

$$F''(r) = -\sum_{n=1}^{\infty} e^{-\alpha\lambda_n^2 t} \frac{\lambda_n (J_0(\lambda_n r) - J_2(\lambda_n r))}{2R J_1(R\lambda_n)}$$

Differentiating the righthand-side of Eq. (27) directly and assembling the pieces, we have

$$\frac{\partial r}{\partial t} = \frac{\alpha}{r} + \frac{\alpha F''(r)}{F'(r)} + \frac{\alpha}{F'(r)} \sum_{n=1}^{\infty} e^{-\alpha\lambda_n^2 t} \frac{\lambda_n J_0(\lambda_n r)}{R J_1(R\lambda_n)} \tag{32}$$

To obtain $\dot{s}(t)$, we evaluate Eq. (32) at the values of r and t such that $T(r, t) = 800$; examples of such pairs are contained Table 4. Figure 9 below shows the result of this process executed on a finer grid in t.

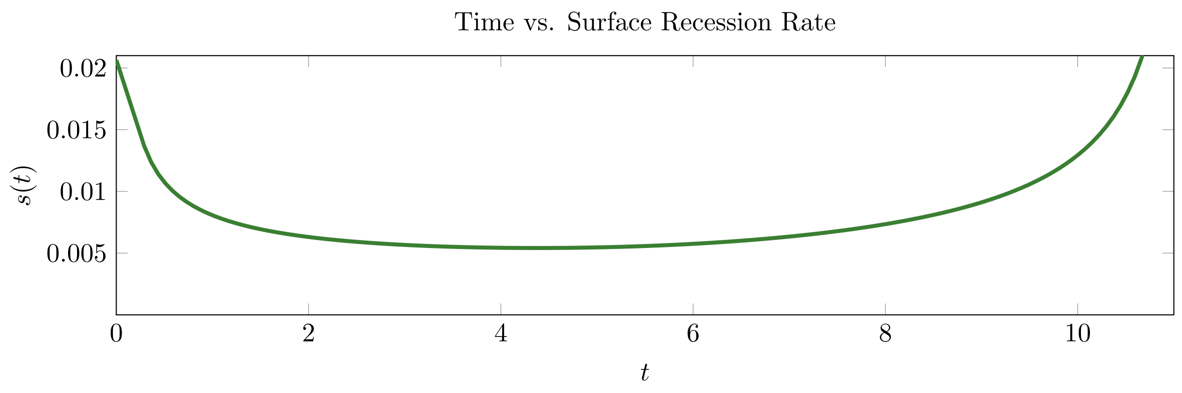


Figure 9. Time vs. Surface Recession Rate. This figure depicts the surface recession rate $\dot{s}$ computed via Eq. (32) and evaluated at (r,t) pairs such as in Table 4.

There is one final piece to the puzzle: the temperature distribution T_{init} when the isotherm is at $\hat{R}$. Per the bolded entry in Table 4, we have $T_{\text{init}} = T(\hat{R}, 0.14347)$. The figure below illustrates T_{init} for $\hat{R} = 0.1$.

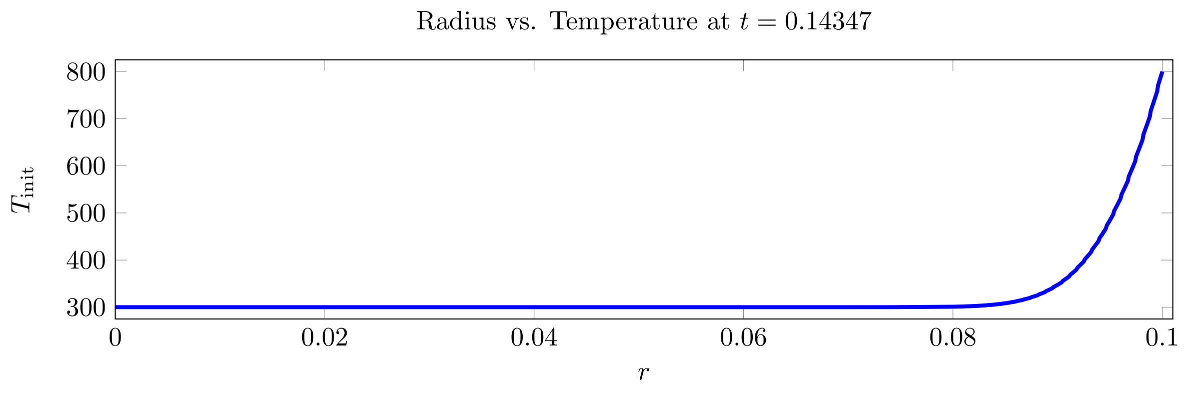


Figure 10. Temperature Distribution When 800 K Isotherm Reaches $\hat{R} = 0.1$. We have $T(\hat{R}, 0.14347) = 800$. This figure shows the temperature as computed by Eq. (27) through the sphere for $r < \hat{R}$. Tabulated data from this figure is needed to initialize the numerical problem.

Similar to Eq. (25), the numerical problem we shall solve is

$$\left.\begin{aligned} \rho c \frac{\partial T}{\partial t} &= \frac{k}{r}\frac{\partial}{\partial r}\left[r\frac{\partial T}{\partial r}\right] && \text{in } \left[0, \hat{R}(t)\right] \times \left\{t : t > t_0\right\} \\ T &= T_{\text{init}} && \text{in } \left[0, \hat{R}(t_0)\right] \times \left\{t = t_0\right\} \\ \frac{\partial T}{\partial r} &= 0 && \text{on } \left\{x = 0\right\} \times \left\{t : t > t_0\right\} \\ T &\equiv 800 && \text{on } \left\{x = \hat{R}(t)\right\} \times \left\{t : t > t_0\right\} \end{aligned}\right\} \tag{33}$$

The specified ablation condition manifests itself in an explicit update to the domain upon each time step:

$$\hat{R}(t_{n+1}) = \hat{R}(t_n) - \dot{s}(t_n)\Delta t \tag{34}$$

As we shall see, this update, which essentially approximates $\dot{s}$ with a step function by assuming it is constant between time steps, destroys second-order convergence. Instead, we expect solutions that converge only first-order in time despite the use of a second-order time integrator. Consequently, we take a more classical approach to the convergence study, deviating from the procedure of the preceding section: we refine only in time (so that we now expect positive slopes). We use the following grid and time spacings:

	Δt	N_e
Level 1	0.05000	100
Level 2	0.02500	100
Level 3	0.01250	100
Level 4	0.00625	100

The grid size, which corresponds to $\Delta h = 10^{-3}$ was selected because subsequent refinements yielded no change in the error metrics, of which we introduce two additional: mass loss and radius.

Letting ψ be an appropriate test function, the weak formulation of Eq. (33) includes a boundary term

$$\int_{\partial\Omega(t)} \psi \cdot \frac{\partial T}{\partial r}\, dx = \psi\big(\hat{R}(t)\big) \cdot \left.\frac{\partial T}{\partial r}\right|_{r=\hat{R}(t)} - \cancelto{0}{\psi\big(\hat{R}(0)\big) \cdot \frac{\partial T}{\partial r}}\bigg|_{r=0}$$

where the second term vanishes because of the zero-flux condition imposed at $r = 0$. We recognize $\frac{\partial T}{\partial r}$ as the solid conductive heat flux $\dot{q}_{\text{cond}}$. Per the above, this quantity is required even when $\dot{s}$ is specified.

In the melting model implemented in `CHAR`, it is assumed that

1. There are no chemical reactions on the surface.
2. Phase change from solid to liquid (melt) occurs at a constant temperature.
3. Phase change occurs instantaneously and melted material is immediately removed from the system.

As a consequence of these assumptions, the mass loss of solid must equal the mass gain in liquid:

$$\dot{m}_{\text{sol}} = \dot{m}_{\text{liq}}$$

The total energy balance at the surface is

$$\dot{q}_{\text{app}} + \dot{q}_{\text{liq}} - \dot{q}_{\text{sol}} - \dot{q}_{\text{cond}} = 0 \tag{35}$$

where $\dot{q}_{\text{app}}$ is the sum of all applied heat fluxes (from convection, radiation, etc.). The middle quantities are

$$\dot{q}_{\text{liq}} = \dot{m}_{\text{liq}} h_{\text{liq}} \qquad\qquad \dot{q}_{\text{sol}} = \dot{m}_{\text{sol}} h_{\text{sol}}$$

Because temperature cannot exceed T_{melt},

$$h_{\text{liq}} = h_{\text{sol}} + \Delta H_f^0$$

where ΔH_f^0 is the latent heat of fusion. The solid mass flux is given by

$$\dot{m}_{\text{sol}} = -\rho \dot{s}$$

so that we may obtain the needed $\dot{q}_{\text{cond}}$ via the formula

$$\dot{q}_{\text{cond}} = \dot{q}_{\text{app}} - \rho \dot{s} \Delta H_f^0 \tag{36}$$

As illustrated in Figure 1, decoupled mesh motion means that there is a mismatch between the time stepping of the main and mesh motion systems when $\dot{s} \not\equiv$ constant. Recalling Eq. (2), the theoretical mass loss over the interval $[t_0, t]$ is computed by the iterated integral

$$\Delta m_{\text{exact}}(t) = \int_{t_0}^{t} \int_{\partial\Omega(\tau)} \rho \dot{s}(\mathbf{x}, \tau)\, dS\, d\tau$$

The boundary integral should be interpreted as integration against a point measure (Dirac delta) so that

$$\int_{\partial\Omega(\tau)} \rho \dot{s}(\mathbf{x}, \tau)\, dS = \rho \dot{s}(\tau)$$

The spatial dependence disappears because, here, $\dot{s}$ is specified radially and r^* may be considered a function of t. Therefore, the expression for mass loss simplifies to

$$\Delta m_{\text{exact}}(t) = \rho \int_{t_0}^{t} \dot{s}(\tau)\, d\tau \tag{37}$$

Lack of analytic formula for $r(t)$ means that symbolic substitution of Eq. (32) leads to a complicated formula offering no additional insight; we may, however, compute it to arbitrary accuracy as we did to make Table 4. This enables use of a highly accurate Gaussian quadrature rule to compute the integral in t.

To compute the mass loss for the numerical solution, we merely track the total length of the domain. The mass loss over the time interval $[t_0, t]$ is simply

$$\Delta m_{\text{num}}(t) = \rho \left[\hat{R}(t_0) - \hat{R}(t)\right] \tag{38}$$

In the verification problem of Section IV.D.1, $\dot{s}$ was constant so the mesh motion was "perfect" in that there was no error in surface location. Furthermore, the location of the surface was not especially important because the temperature field was evaluated not at the nodes' absolute locations but rather their locations relative to the surface (depth), possible because of the semi-infinite construction and moving frame of reference.

Here, we cannot replicate this trick and must account for the difference in the exact domain $\left[0, r^*(t)\right]$ and the numerical $\left[0, \hat{R}(t)\right]$. While we are shielded from attempt to evalute $T_{\text{num}}(x,t)$ for $x > \hat{R}(t)$ because the evaluation points are set by the finite element mesh, we must take care not to evaluate the exact solution for $x > r^*(t)$, a well-defined quantity per Eq. (27) so long as $x < R = 0.10526315789$. To this end, define

$$\mathcal{N}(t) \triangleq \left\{ x_i : x_i \text{ a node and } x_i \le \max\left\{ r^*(t), \hat{R}(t) \right\} \right\}$$

	$t = 0.99347$			**$t = 1.99347$**			**$t = 2.99347$**		
	T	**Δm**	**$\hat{R}$**	**T**	**Δm**	**$\hat{R}$**	**T**	**Δm**	**$\hat{R}$**
Level 1	6.4145	0.3790	0.0006612	5.5145	0.3260	0.0006146	5.1431	0.2951	0.0005975
Level 2	3.2648	0.1909	0.0003327	2.7977	0.1644	0.0003090	2.6079	0.1489	0.0002997
Level 3	1.6441	0.0958	0.0001662	1.4039	0.8262	0.0001537	1.3066	0.0749	0.0001485
Level 4	0.8208	0.0480	0.0000824	0.6974	0.4146	0.0000754	0.6468	0.0377	0.0000722
	$t = 4.99347$			**$t = 6.99347$**			**$t = 8.99347$**		
	T	**Δm**	**$\hat{R}$**	**T**	**Δm**	**$\hat{R}$**	**T**	**Δm**	**$\hat{R}$**
Level 1	5.1625	0.2518	0.0005901	5.2377	0.2165	0.0006094	4.6800	0.1772	0.0006752
Level 2	2.6167	0.1273	0.0002949	2.6411	0.1097	0.0003035	2.3350	0.0901	0.0003350
Level 3	1.3078	0.0642	0.0001450	1.3109	0.0557	0.0001481	1.1463	0.0463	0.0001624
Level 4	0.6439	0.0325	0.0000694	0.6374	0.0285	0.0000698	0.5468	0.0243	0.0000755

Table 5. Quantity Errors at Various Times. This table shows the errors $\mathcal{E}(t\,|\,\square)$, where $\square$ is one of T, Δm, or $\hat{R}$. We see that for each quantity, the error declines as time evolves. Within a time block, halving of the time step halves the error, indicating first-order convergence.

We then define the temperature field error to be

$$\mathcal{E}(t\,|\,T) = \left[|\mathcal{N}(t)|^{-1} \sum_{x \in \mathcal{N}(t)} \big(T_{\text{exact}}(x,t) - T_{\text{num}}(x,t)\big)^2 \right]^{\frac{1}{2}}$$

The metrics for the mass loss and radius are exactly what one would expect:

$$\mathcal{E}(t\,|\,\hat{R}) = \left|\hat{R}(t) - r^*(t)\right| \qquad\qquad \mathcal{E}(t\,|\,\Delta m) = \left|\Delta m_{\text{num}}(t) - \Delta m_{\text{exact}}(t)\right|$$

Table 5 lists the values of these errors metrics for several times. Figure 11 plots these values with the best-fit line for a subset of the times in the table. Finally, Table 6 lists the orders of convergence at many times. In computing the order, we assume that $(\Delta h)^q \ll (\Delta t)^p$ in Eq. (21) so that the spatial discretization error may be discarded with the higher order terms. We then best-fit the data of Table 5 to $\log \mathcal{E} = p \log \Delta t + \log \beta$.

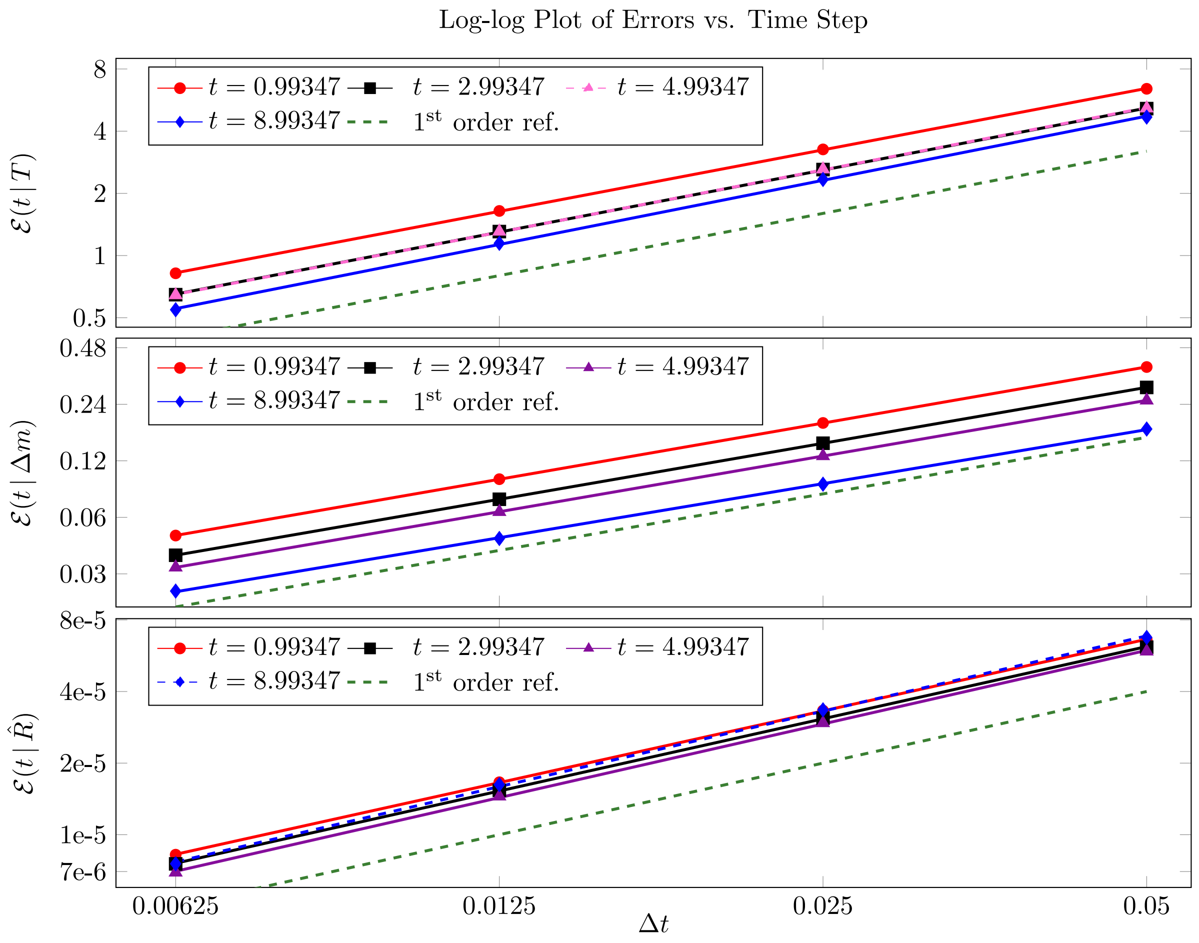


Figure 11. Log-log Plots of Error Metrics. Shown for four values of t are plots of the information from Table 5 along with a reference line that has the expected slope of 1.

	t									
	0.993	**1.993**	**2.993**	**3.993**	**4.993**	**5.993**	**6.993**	**7.993**	**8.993**	**9.993**
$\boldsymbol{T}$	0.9888	0.9944	0.9971	0.9983	1.0010	1.0060	1.0127	1.0213	1.0319	1.0473
$\boldsymbol{\Delta m}$	0.9937	0.9918	0.9899	0.9876	0.9847	0.9809	0.9758	0.9684	0.9567	0.9344
$\boldsymbol{\hat{R}}$	1.0014	1.0089	1.0158	1.0224	1.0288	1.0351	1.0413	1.0474	1.0529	1.0554

Table 6. Orders of Convergence at Various Times. This table shows the values of the slope p in the best-fit line $\log \mathcal{E}(t\,|\,\square) = p \log \Delta t + \log \beta$, where $\square$ is one of T, Δm, or $\hat{R}$.

Section IV.D.1 showed that constant specified melt yields second-order convergence in time and space. The one-dimensional axisymmetric problem of this section, perfect in its geometry (no spatial discretization error), removes the specifics of mesh motion from the picture. Consequently, reduction to first order cannot be attributed to poor mesh motion and must be the fault of the time-varying specified melt.

3. Validation of `CHAR`: Time-Varying Heat Flux

In the demonstration problem to come, we will to prescribe the heat flux on the surface rather than use specified recession. We must, therefore, demonstrate `CHAR`'s correct handling of this boundary condition. We can construct the heat flux from specified surface recession via Eq. (36). The conductive heat flux is

$$q_{\text{cond}}(t) = k\frac{\partial T}{\partial r} = 2k(T_0 - T_\infty)F'(r(t)) \tag{39}$$

where $r(t)$ parameterized as in Table 4 and F' is as in Eq. (29). This distribution is plotted in Figure 12.

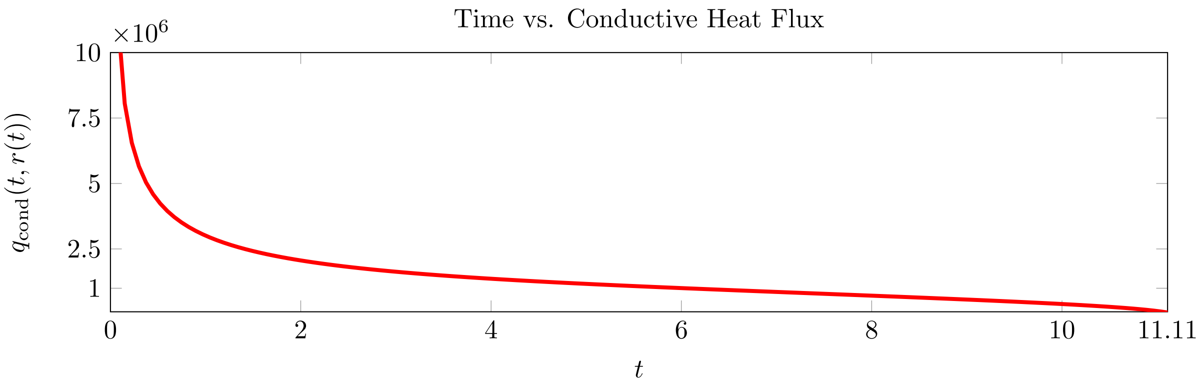


Figure 12. Time vs. Heat Flux. This figure depicts time versus conductive heat flux at the $800K$ **isotherm (radius** r^* **within the sphere). The value** $r(t)$ **is computed as in Table 4.**

We retain the same material properties as before and add $\Delta H_f^0 = 6 \times 10^6$ J/kg. The Cauchy problem is then

$$\left.\begin{aligned} \rho c\frac{\partial T}{\partial t} &= \frac{k}{r}\frac{\partial}{\partial r}\left[r\frac{\partial T}{\partial r}\right] && \text{in } \left[0, \hat{R}(t)\right] \times \left\{t : t > t_0\right\} \\ T &= T_{\text{init}} && \text{in } \left[0, \hat{R}(t_0)\right] \times \left\{t = t_0\right\} \\ \frac{\partial T}{\partial r} &= 0 && \text{on } \left\{x = 0\right\} \times \left\{t : t > t_0\right\} \\ \frac{\partial T}{\partial r} &= q_{\text{cond}}(t) + \rho\dot{s}(t)\Delta H_f^0 && \text{on } \left\{x = \hat{R}(t)\right\} \times \left\{t : t > t_0\right\} \\ T &\equiv 800 && \text{on } \left\{x = \hat{R}(t)\right\} \times \left\{t : t > t_0\right\} \end{aligned}\right\} \tag{40}$$

where $t_0 \triangleq 0.14347$ and $R'(0) = 0.1$ as before. Table 7 below is the analog of Table 5; Table 8, of Table 6.

	$t = 0.99347$			$t = 1.99347$			$t = 2.99347$		
	T	Δm	$\hat{R}$	T	Δm	$\hat{R}$	T	Δm	$\hat{R}$
Level 1	6.1285	0.3591	0.0006266	5.2228	0.3067	0.0005773	4.8536	0.2761	0.0005590
Level 2	3.0505	0.1759	0.0003066	2.5723	0.1488	0.0002794	2.3797	0.1337	0.0002687
Level 3	1.4748	0.0838	0.0001452	1.2190	0.0696	0.0001289	1.1157	0.0619	0.0001219
Level 4	0.6778	0.0376	0.0000642	0.5360	0.2982	0.0000533	0.4780	0.0259	0.0000482
	$t = 4.99347$			$t = 6.99347$			$t = 8.99347$		
	T	Δm	$\hat{R}$	T	Δm	$\hat{R}$	T	Δm	$\hat{R}$
Level 1	4.8610	0.2355	0.0005514	4.9237	0.2034	0.0005720	4.4027	0.1683	0.0006406
Level 2	2.3729	0.1138	0.0002628	2.3820	0.0985	0.0002714	2.1008	0.0822	0.0003038
Level 3	1.0998	0.0524	0.0001168	1.0858	0.0456	0.0001193	0.9382	0.0390	0.0001335
Level 4	0.4572	0.0216	0.0000435	0.4323	0.0192	0.0000429	0.3539	0.0173	0.0000480

Table 7. Quantity Errors at Various Times (Heat Flux). This table shows the errors $\mathcal{E}(t\,|\,\square)$**, where** $\square$ **is one of** T**,** Δm**, or** $\hat{R}$ **when the heat flux boundary condition is used, as in Eq.** (40)**. The errors are slightly lower than for the specified ablation condition.**

	t									
	0.993	**1.993**	**2.993**	**3.993**	**4.993**	**5.993**	**6.993**	**7.993**	**8.993**	**9.993**
T	1.0579	1.0931	1.1124	1.1232	1.1340	1.1486	1.1166	1.1863	1.2074	1.2278
Δm	1.0832	1.1177	1.1348	1.1431	1.1453	1.1422	1.1337	1.1182	1.0922	1.0442
$\hat{R}$	1.0937	1.1429	1.1748	1.1985	1.2167	1.2304	1.2398	1.2441	1.2405	1.2187

Table 8. Orders of Convergence at Various Times (Heat Flux). This table shows the values of the slope p in the best-fit line $\log \mathcal{E}(t \,|\, \square) = p \log \Delta t + \log \beta$, where $\square$ is one of T, Δm, or $\hat{R}$.

Quite curiously, despite the equivalence of the boundary conditions, the errors for the heat flux are slightly lower than those of the specified melt and the convergence superlinear.

4. *Validation of Mesh Motion: The Analytic Cylinder*

Having established the soundness of our thermal response code and identified that the best possible result is first-order convergence in the case of "perfect mesh motion," we can finally present the verification of the two-dimensional tangle-free mesh motion that is the subject of this report. Consider

$$\left.\begin{aligned} \rho c \frac{\partial T}{\partial t} &= k \nabla^2 T && \text{in } \Omega(t) \times \{t : t > t_0\} \\ T &= T_{\text{init}} && \text{in } \Omega(t_0) \times \{t = t_0\} \\ \frac{\partial T}{\partial \boldsymbol{\nu}} &= 0 && \text{on } (\Gamma_1 \cup \Gamma_2) \times \{t : t > t_0\} \\ \frac{\partial T}{\partial \boldsymbol{\nu}} &= q_{\text{cond}}(t) + \rho \dot{s}(t) \Delta H_f^0 && \text{on } \Gamma_3 \times \{t : t > t_0\} \\ T &\equiv 800 && \text{on } \Gamma_3 \times \{t : t > t_0\} \end{aligned}\right\} \quad (41)$$

The initial domain is a quarter circle of radius $\hat{R} = 0.1$ for $\theta \in \left[\frac{\pi}{4}, \frac{3\pi}{4}\right]$, shown below in Figure 13. This is a fully two-dimensional version of Eq. (40) that exploits symmetry to reduce computational burden.

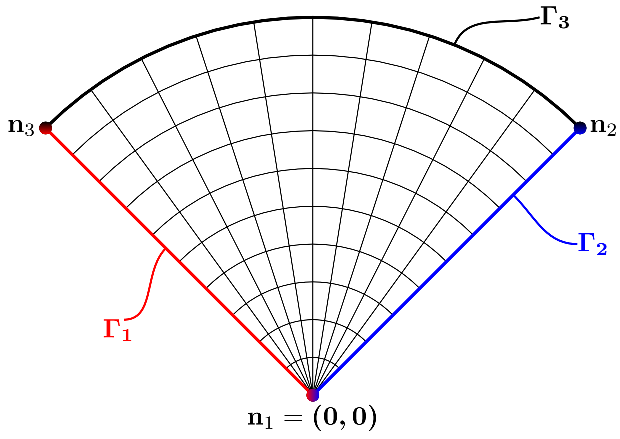


Figure 13. Domain for Two-dimesional Problem. The grid is exemplary and is not to be taken literally in the number or shape of elements. Symmetry allows the use of a quarter circle; we choose this off-axis wedge to highlight the capability of tangle-free mesh motion.

Each Γ_k defines a distinct side set for mesh motion; in particular, Γ_1 and Γ_2 are separated so that $\mathbf{n}_1$ is correctly identified as a corner. If the two were combined into a single set, $\mathbf{n}_1$ would be incorrectly processed as an inlaid slider. We declare Γ_1 and Γ_2 to be *sliding* and Γ_3 to be *melting (receding)*.

We use the following grid and time spacings:

	Δt	N_e
Level 1	0.02500	160
Level 2	0.01250	160
Level 3	0.00625	160

These time steps mirror what we have used up to this point except that we've eliminated $\Delta t = 0.05$ because this was found not to be accurate enough and produced errors not in line with the order of convergence indicated by the deeper levels. We've also increased the spatial resolution by 60% for good measure.

N_e is the "number of elements on a line." It is the number of elements along each ray like the longitudinal grid lines in Figure 13. The finite element mesh for this example is generated by dividing the segment Γ_2 into N_e uniform pieces. We then rotate this line though an angle of $\frac{\pi}{4N_e}$ and repeat until Γ_1 is reached. This generates a grid exactly like the one in the figure. For each t, $\Omega(t)$ then contains N_e^2 total elements.

Direct comparision of the entries of the Tables 9 and 10 below reveals that the errors and orders of convergence are very similar to those of the one-dimensional heat flux problem (listed in Tables 7 and 8).

	$t = 0.99347$			**$t = 1.99347$**			**$t = 2.99347$**		
	T	**Δm**	**$\hat{R}$**	**T**	**Δm**	**$\hat{R}$**	**T**	**Δm**	**$\hat{R}$**
Level 1	3.1797	0.1786	0.0003113	2.6571	0.1521	0.0002857	2.4472	0.1372	0.0002757
Level 2	1.5571	0.0867	0.0001503	1.2761	0.0731	0.0001355	1.1680	0.0655	0.0001293
Level 3	0.7301	0.0406	0.0000694	0.5830	0.0331	0.0000599	0.5229	0.0295	0.0000555
	$t = 4.99347$			**$t = 6.99347$**			**$t = 8.99347$**		
	T	**Δm**	**$\hat{R}$**	**T**	**Δm**	**$\hat{R}$**	**T**	**Δm**	**$\hat{R}$**
Level 1	2.4217	0.1170	0.0002705	2.4280	0.1030	0.0002794	2.1434	0.0843	0.0003120
Level 2	1.1468	0.0558	0.0001249	1.1354	0.0486	0.0001277	0.9894	0.0411	0.0001421
Level 3	0.5017	0.0250	0.0000515	0.4820	0.0221	0.0000513	0.4042	0.0195	0.0000565

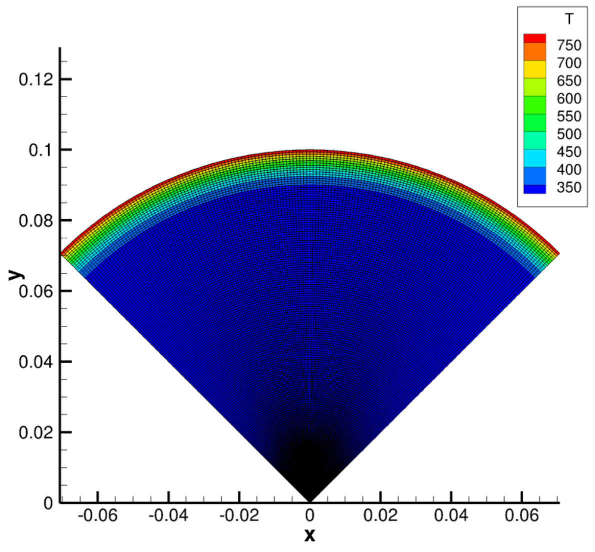
Table 9. Quantity Errors at Various Times (2D Heat Flux). This table shows the errors $\mathcal{E}(t\,|\,\square)$, where $\square$ is one of T, Δm, or $\hat{R}$ when the heat flux boundary condition is used, as in Eq. (40). The errors very much in line with those listed in Table 7.

	t								
	0.993	**1.993**	**2.993**	**3.993**	**4.993**	**5.993**	**6.993**	**7.993**	**8.993**
T	1.0614	1.0941	1.1133	1.1266	1.1356	1.1478	1.1663	1.1832	1.2034
Δm	1.0691	1.0958	1.1082	1.1134	1.1136	1.1093	1.1010	1.0838	1.0570
$\hat{R}$	1.0829	1.1266	1.1559	1.1785	1.1968	1.2117	1.2233	1.2310	1.2322

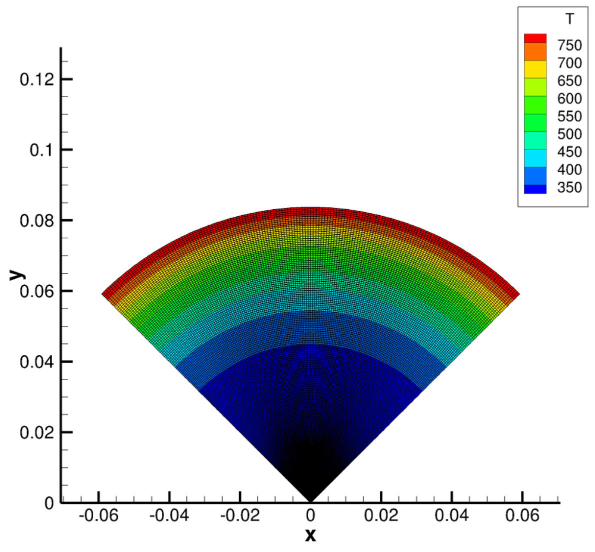
Table 10. Orders of Convergence at Various Times (2D Heat Flux). This table shows the values of the slope p in the best-fit line $\log \mathcal{E}(t\,|\,\square) = p \log \Delta t + \log \beta$, where $\square$ is T, Δm, or $\hat{R}$.

While not reproduced here, the simulation was also run with a specified ablation boundary condition in the vein of Section IV.D.2. The errors and orders of convergence were again remarkably similar to those in that section. We omit plots similar to Figure 11 because of the similarity of the data with that case. Instead, Figure 14 shows the process of mesh motion at several times, showcasing tangle-free's superb performance.

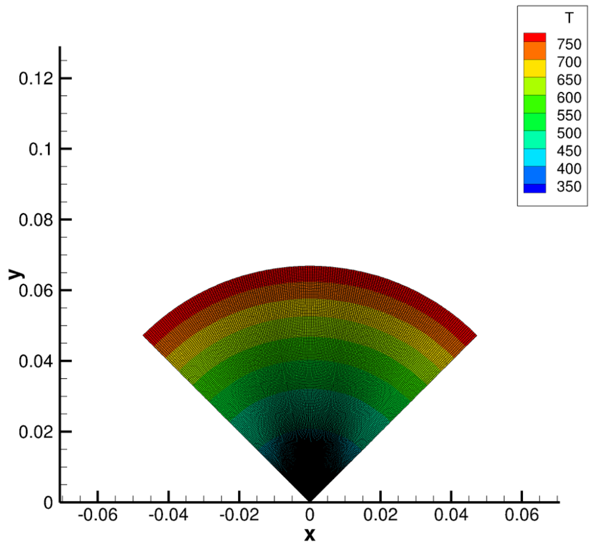
This concludes the verification, comparison of simulation results of problems whose exact responses are known. The lengthy procedure established first that `CHAR`, the thermal response code which tangle-free mesh motion calls home, is fundamentally sound by confirming second-order convergence in time and space when ablation is constant. We then showed that time-variance reduces order—independent of mesh motion scheme, by working in one-dimension—and verified the correctness of `CHAR` for heat-flux boundary conditions. In



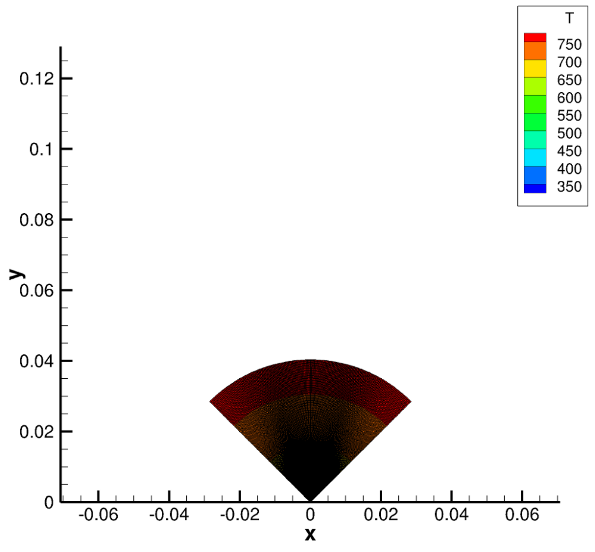
(a) $t = t_0$



(b) $t = 1.99347$



(c) $t = 4.99347$



(d) $t = 8.99347$

Figure 14. `CHAR` Simulation Results. This figure shows the temperature distribution $T(r,t)$ at four values of t. As it evolves in time, the mesh retains all the quality it possessed at the beginning, never tangling along the curved portion or displaying any corruption. Sliding along the off-axis sides, a limitation of the simple normal scheme, is executed without issue.

particular, the very last of these examples corroborates the claim in two-dimensions by direct comparision to one-dimensional results, for which there exists only a single method of mesh motion.

E. Demonstration

Confident that our results will be correct with reasonable accuracy, we now demonstate the enhanced functionality provided by tangle-free mesh motion. First, we will show an iso-q, comparing directly to the simple algorithm whose limitations provided the motivation, as well as the name, to develop tangle-free mesh motion. The problems following will fully brandish the robust and cutting-edge capabilities of the scheme.

1. *Axisymmetric Iso-q with Aerodynamic Heating*

For the first demonstration problem, we consider the following geometry:

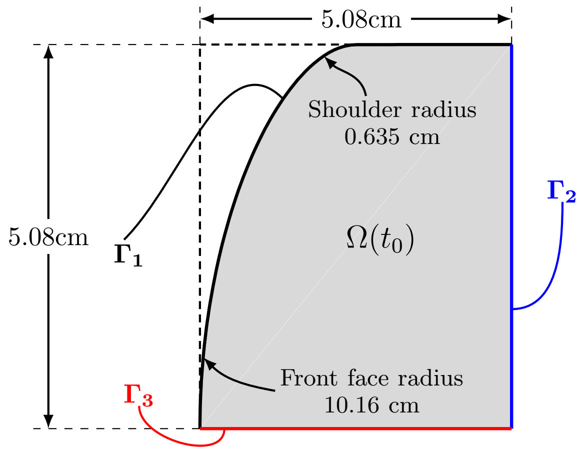


Figure 15. Iso-q Geometry. Geometry for the example of this section. The red boundary Γ_3 denotes the through-the-thickness portion while the blue Γ_2 marks the in-plane piece.

The figure depicts the cross-section of an axisymmetric body (axis of rotation coincident with Γ_3). This shape is referred to as an "iso-q" because the geometry causes the convective heating ("q") to be nearly constant ("iso") on the front face, a property accomplished by having the radius of the front face equal to the diameter of the cylinder formed after Γ_2 is rotated in-place about Γ_3. The geometry of this case is inspired by arcjet test pieces used for testing thermal protection systems on spacecraft. The actual test specimens contain additional material attached to the back, not included here to eliminate unnecessary computation.

The examples up to now have used simple melters with constant material properties. This time, we use a fictional ablator called TACOT (Thermal Ablative Composite for Open Testing) [13] whose properties are freely available[a]. As a charring ablator and porous material, the thermal response of TACOT is no longer governed by the simple heat equation, instead requiring gas and solid continuity equations and associated terms in the energy equation. These are implemented by `CHAR`; for details, please consult [1].

We use an aerodynamic heating condition on Γ_1, in which convective applied heat flux is

$$\dot{q}_{\text{app}} = \rho_e u_e C_H (h_r - h_w)$$

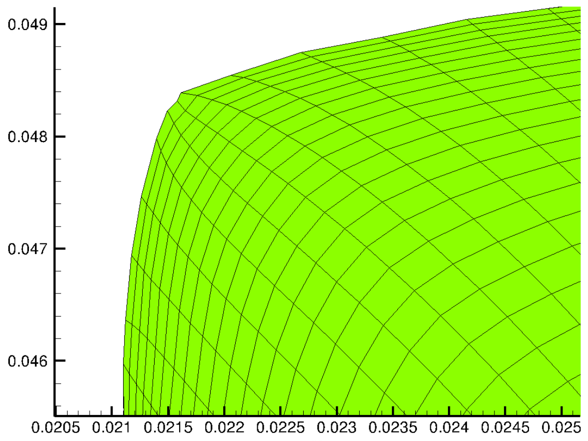
where ρ_e is the density at the boundary layer edge, u_e is the velocity of the gas at the boundary layer edge, C_H is the film coefficient, h_r is the recovery enthalpy, and h_w is the wall enthalpy. The first two of these and last of these are computed within `CHAR` while the others must be specified by input. For recovery enthalpy, we take $h_r \equiv 1.9 \times 10^7 \, \mathrm{J/kg}$. The film coefficient varies in position along Γ_3 but independent of t. To compute the wall enthalpy, the pressure P on the boundary is also needed, also spatially varying but constant in time. Both C_H and P have been precomputed from CFD simulations. These are input via table to `CHAR`.

On the remaining two subsets of the boundary, we impose zero flux conditions. To summarize,

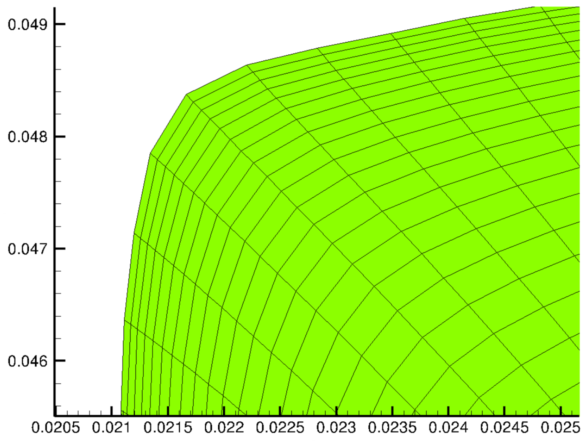
Γ_1	Receding	Aeroheating	$\dot{q} = \rho_e u_e C_H(\mathbf{x}) \cdot (h_r - h_w(\mathbf{x}))$
Γ_2	Sliding	Heat flux	$\dot{q} \equiv 0$
Γ_3	Sliding	Heat flux	$\dot{q} \equiv 0$

Finally, the initial temperature and pressure are taken to be $T_0 = 293.15$ K and $P_0 = 800$ Pa, respectively. A variable time step with minimum $(\Delta t)_{\min} = 0.01$ and maximum $(\Delta t)_{\max} = 0.25$ were used; the simulation was run for time from $t_0 = 0$ to $t_f = 250$. Figure 17 below shows the results at three time steps.

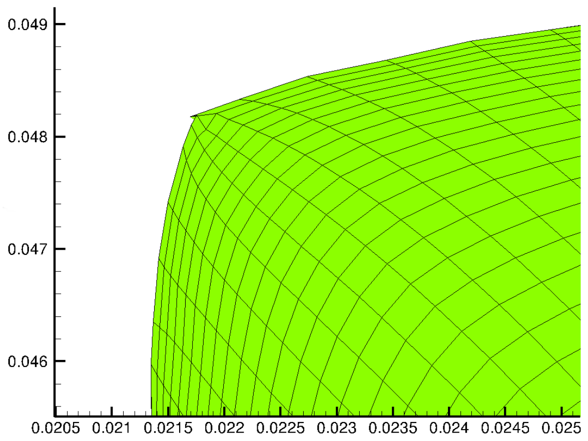
[a]http://ablation2015.engineering.uky.edu/code-comparison, under heading TACOT3.0



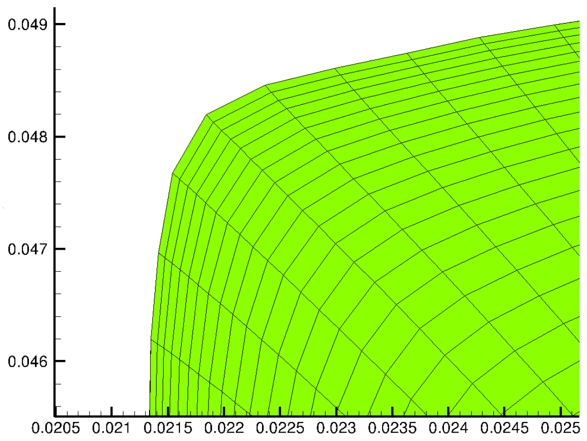
1(a) Simple Normal, $t \approx 144$



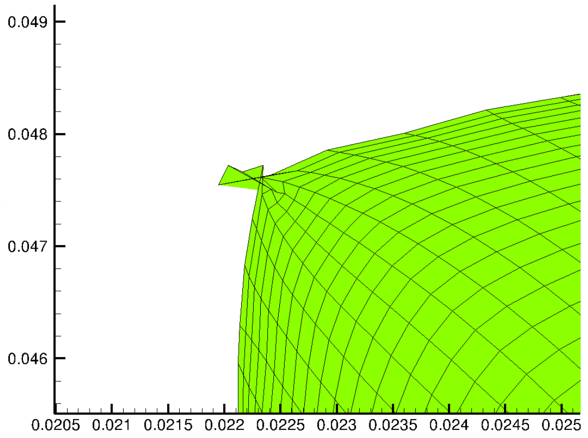
1(b) Tangle-free, $t \approx 144$



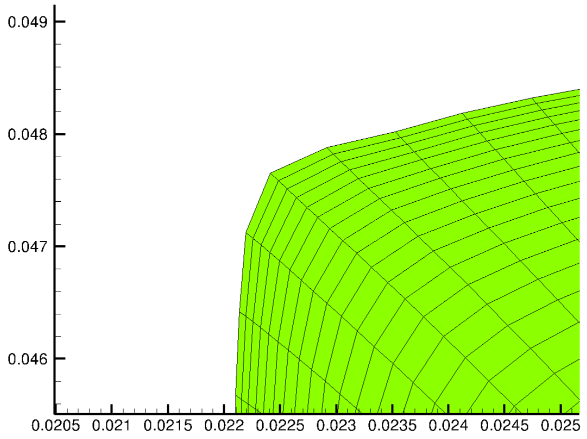
2(a) Simple Normal, $t \approx 157$



2(b) Tangle-free, $t \approx 157$



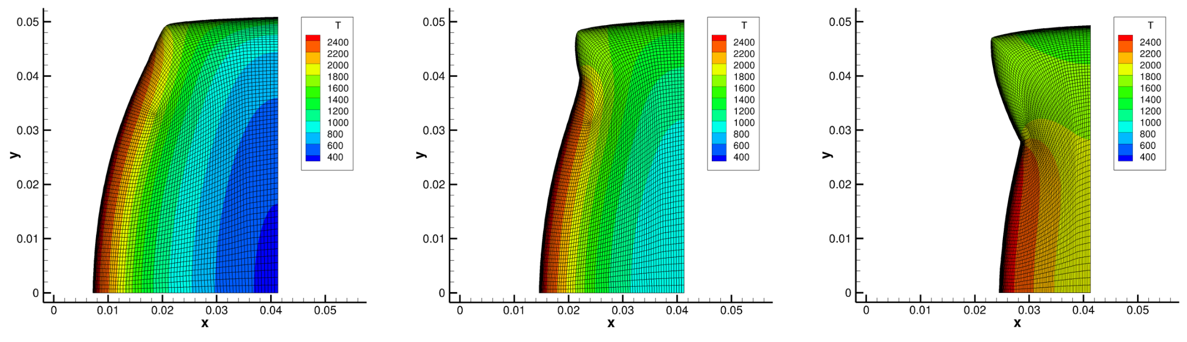
3(a) Simple Normal, $t = 200$



3(b) Tangle-free, $t = 200$

Figure 16. Shoulder Performance: Simple vs. Tangle-free. The left column shows the evolution of the mesh of the simple normal scheme; the right, when tangle-free is employed instead. At $t \approx 144$, the first hints that simple normal is walking a path to inevitable destruction surface, as we can see a small "tooth" beginning to develop in the shoulder. At $t \approx 157$, the mesh has tangled in itself as shown in 2(a). Surprisingly, the thermal solve continues to run, as elements remain acute and the finite element package (`libmesh`) cannot detect interelement collisions. At $t = 200$, it is a tangled mess. Tangle-free (right column) has none of these issues.

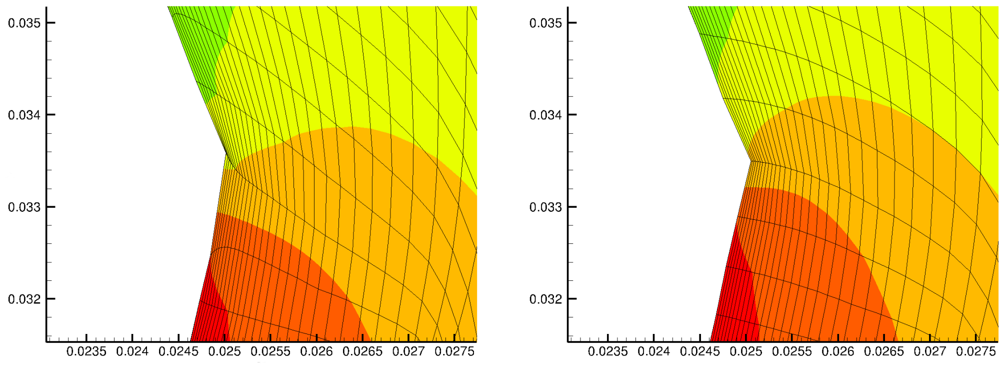
Of primary interest in this study is the performance gains over a "simple mesh motion," which takes the direction of motion to be the normal in all cases. As described before, such a scheme precludes sliding along anything other than a coordinate axis; fortuantely, the orientation of this problem geometry is compatible with that limitation. Figure 16 showcases the payoff of the more sophisticated tangle-free mesh motion.



(a) $t \approx 75$ **(b) $t \approx 150$** **(d) $t \approx 250$**

Figure 17. `CHAR` Simulation Results. This figure shows the temperature distribution $T(r,t)$ at four values of t. The kink that occurs late in time is nonphysical.

The shoulder is not the only spot on the mesh where tangle-free shines while simple flounders. Figure 18 below shows another magnificent improvement over the simple algorithm in a difficult region of the mesh.



(a) Simple, $t = 200$ **(b) Tangle-free, $t = 200$**

Figure 18. Resolving "The Kink." This figure shows the mesh in the vicinity of "the kink," the v-like formation that increases in severity with time, at $t = 250$. The anamoly is nonphysical but illustrative for this exercise in mesh motion. On the left, the mesh is skewed and of poor quality, while the tangle-free algorithm (right) remains clean and well-resolved.

The simple scheme is, strictly speaking, the "native way" to do mesh motion. Physically, the surface energy balance equation gives the recession normal to the surface. However, this example reveals that this theoretical construct translates poorly to the discrete geometry of numerical simulations. Tangle-free mimics moving along the normal by moving entire faces, maintaining the correct geometry in the process.

2. Sinusoidal Surface Slider

The next demonstration problem is intended to feature tangle-free mesh motion's adeptness with sliding side sets, enabling dynamic motion along complex boundaries. Consider the geometry of Figure 19.

We place simple zero heat-flux boundary conditions along each of Γ_1, Γ_3, and Γ_4, all of which are declared to be sliding boundaries. A constant heat-flux condition with $\dot{q} \equiv 2 \times 10^7\ \mathrm{W/m^2}$ is specified on Γ_2, which is set as a *receding* side set. The solid is a simple melter with the following constant material properties:

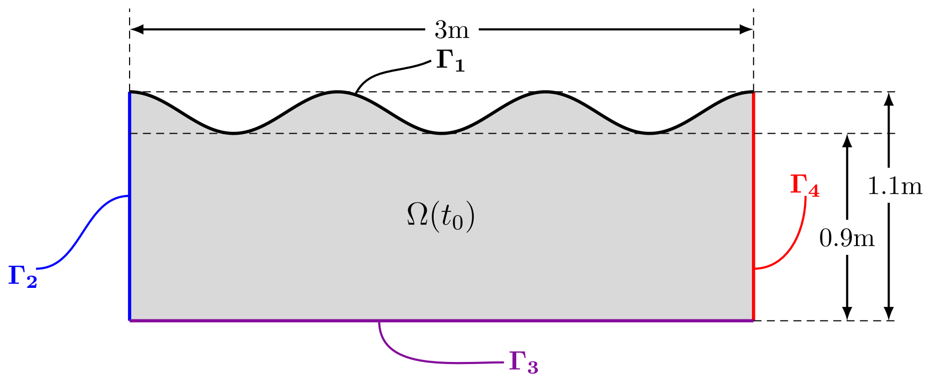
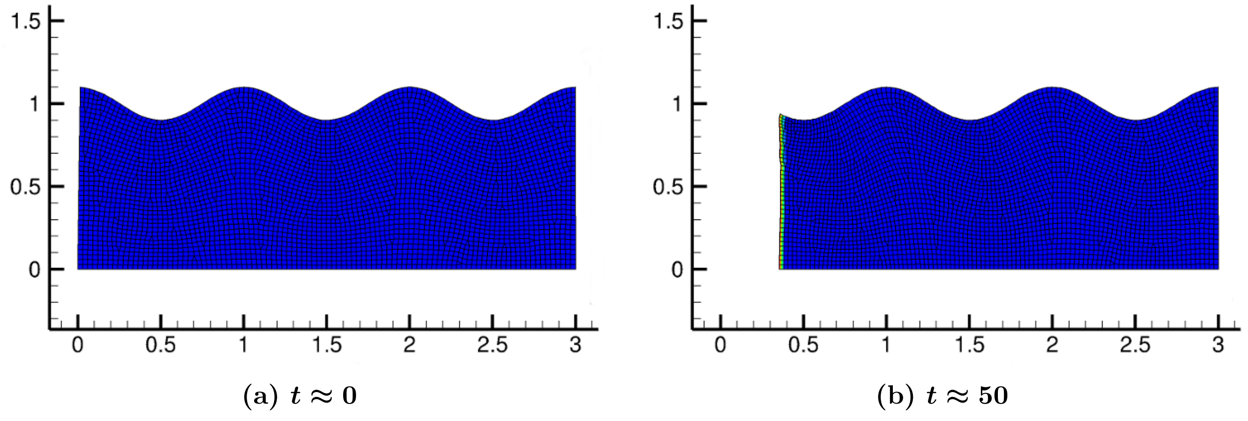


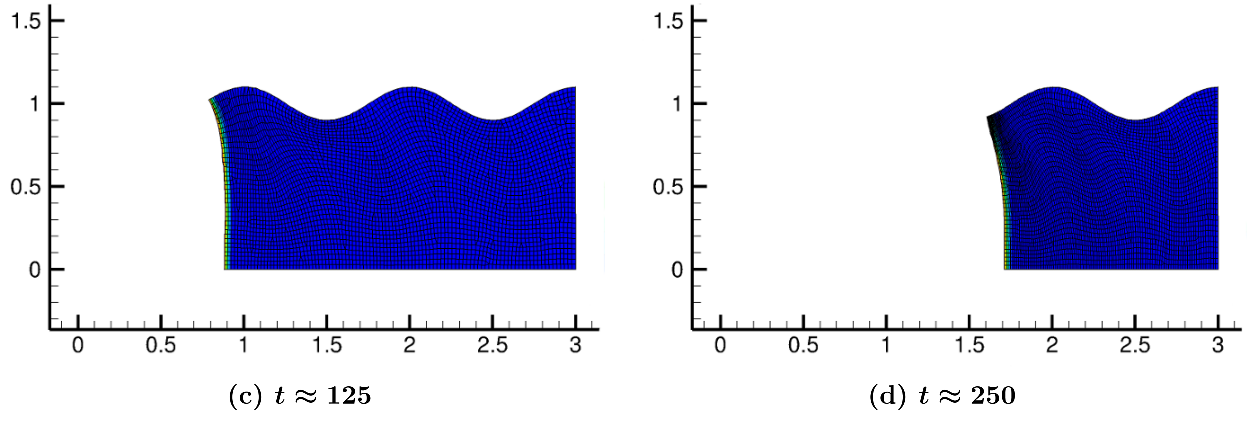
Figure 19. Sinusoidal Surface Slider. Problem geometry for the demonstration of this section. The topmost boundary Γ_1 is represented by the curve $\gamma_1(x) = 1 + 0.1\cos(2\pi x)$.

Solid density	ρ	8960 $^{\text{kg}}/_{\text{m}^3}$	Melt temperature	T_{melt}	1000 K
Thermal conductivity	k	394 $^{\text{W}}/_{\text{m-K}}$	Latent heat of fusion	ΔH_f^0	2.05×10^5 $^{\text{J}}/_{\text{kg}}$
Specific heat	c	383 $^{\text{J}}/_{\text{kg-K}}$			

The simulation was run with an adaptively chosen time step with $(\Delta t)_{\text{min}} = 0.001$ and maximum $(\Delta t)_{\text{max}} = 0.25$. The inital time is $t_0 = 0$ and a final time of $t_f = 500$ was requested, although it was expected the grid would fail before that. Figure 20 contains the results from the simulation at four times.



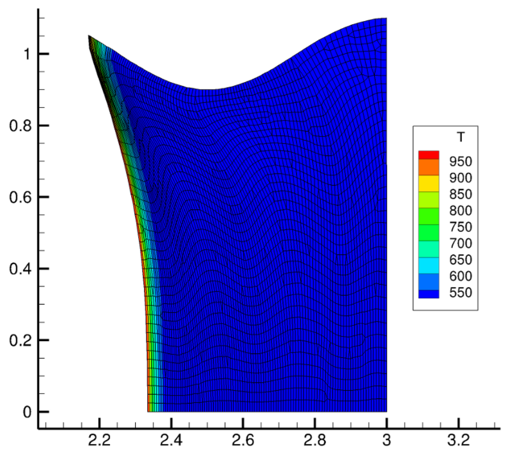
(a) $t \approx 0$ **(b) $t \approx 50$**



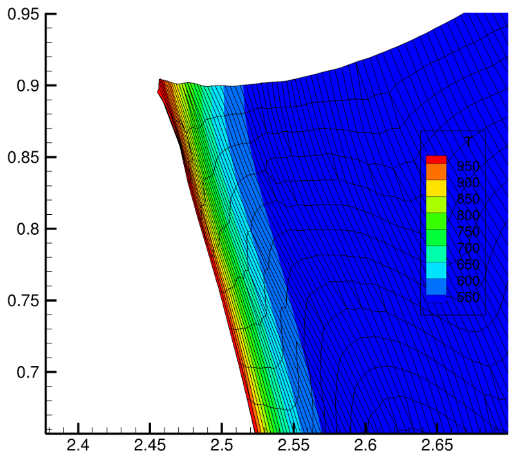
(c) $t \approx 125$ **(d) $t \approx 250$**

Figure 20. Sinusoidal Surface Slider CHAR Simulation Results. Due to the size of the solid and low material conductivity, there is minimal heat transfer deep into the bulk. As the material recedes on the left, the upper left corner perfectly follows the wave-like top surface.

Grid failure occurs at $t = 392.72$, at which point approximately 75% of the solid has melted away. Because nodes are never removed, elements decrease in size and increase in skew. If the time step is not adaptively scaled down, the dependence of Δs on time makes it likely that it will eventually exceed the size of an element, twisting the element and triggering failure due to element with negative volume. However, before this happens, limits of precision will often cause other corruption. Figure 21 illustrates this action.



(a) $t \approx 350$



(b) $t \approx 390$

Figure 21. The Dream Deflated. (a) shows at $t = 350$, at which time everything still looks acceptable. The elements initially began as squares but after such a long time, they have morphed into rectangles and other bizarre-looking shapes. (b) exposes the corruption in the mesh that will cause its demise.

Finally, we compare performance with simple normal mesh motion, shown in Figure 22. The simple normal scheme behaves like a compressed solid instead of one melting due to extreme thermal trauma.

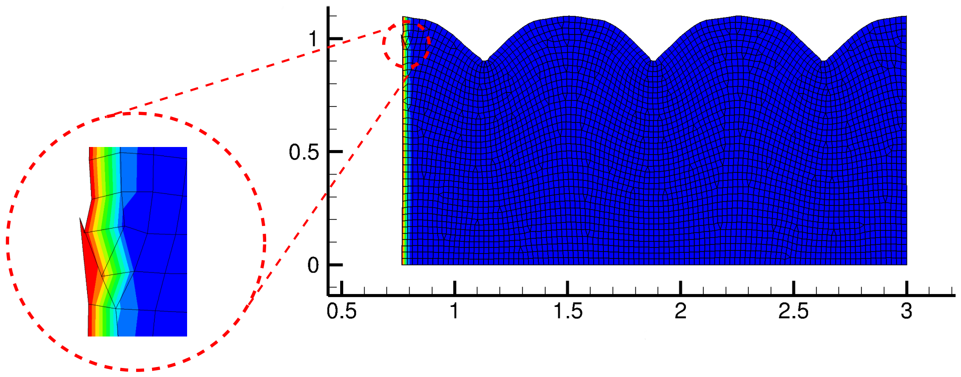


Figure 22. Performance of Simple Scheme. This is the final successful time step before the crash at $t = 107.47$s. It is an unmitigated disaster: the time to failure is short and the shape incorrect. As we see in Figure 20, the left side does not remain vertical as time evolves. The sudden curvature of this side is the ultimate cause of failure (zoomed region).

3. Doubly Receding Rectangle

Our final demonstration is one that, in theory, should not give the simple scheme difficulty. In practice, however, the story is rather different. As the reader surely expects, the simple scheme performs quite poorly again while tangle-free excels. Figure 23 depicts the simple geometry for concluding example.

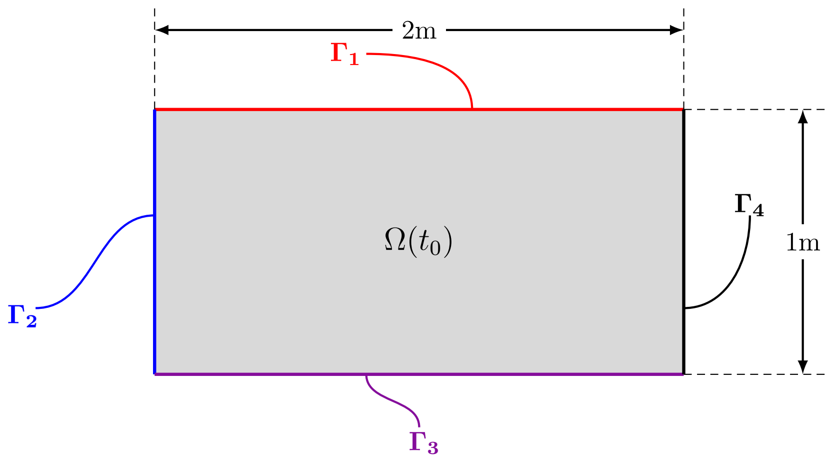


Figure 23. Double Receding Rectangle Geometry.

We use the same simple melting material as in the previous example. We declare Γ_1 and Γ_2 *both* to be receding, each equipped with the same constant heat-flux boundary condition $\dot{q} \equiv 4 \times 10^6\ {}^{\mathrm{W}}/_{\mathrm{m}^2}$. Γ_3 and Γ_4 are sliding side sets with zero heat-flux conditions. We use adaptive time stepping with $(\Delta t)_{\min} = 0.01$ and maximum $(\Delta t)_{\max} = 0.05$. Figure 25 displays tangle-free's handling of this problem.

The initial grid is a bit unusual and worthy of detailed explanation. Figure 24 reproduces a view enhanced from that which can be seen in Figure 25. The hotter burn of the corner from the double input heat flux causes the corner element to deform from its square shape quickly. Once the temperature field on the surface evens out, this is no longer an issue. The key, therefore, is successfully reaching that point of stabilization. The larger element in comparison to the rest of the grid helps accomplish precisely that: there is significantly more headroom so that as the corner caves in, the element doesn't trip over itself and cause failure.

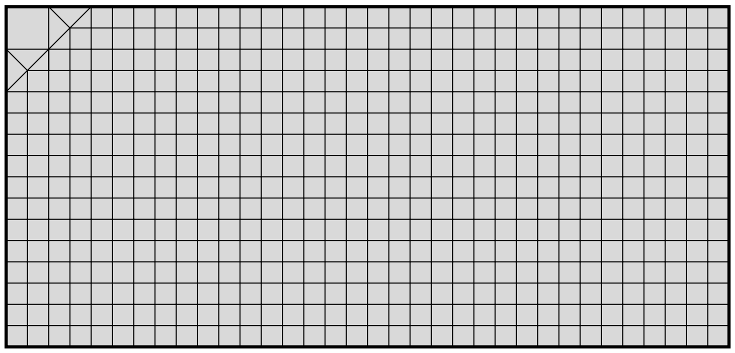
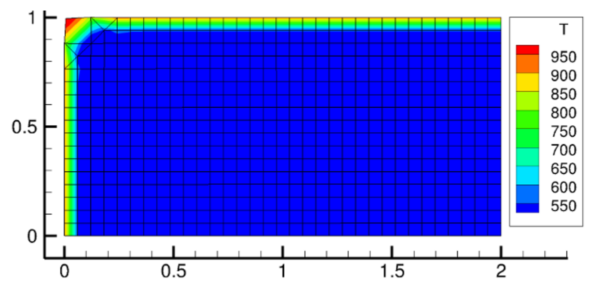
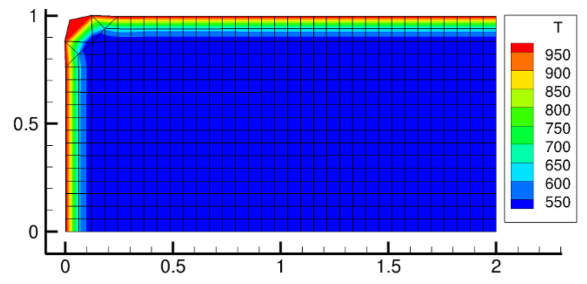


Figure 24. The Strange Grid. The large quadrilateral in the corner aids in delaying a cave-in long enough for a uniform temperature to be reached over the union of receding side sets.

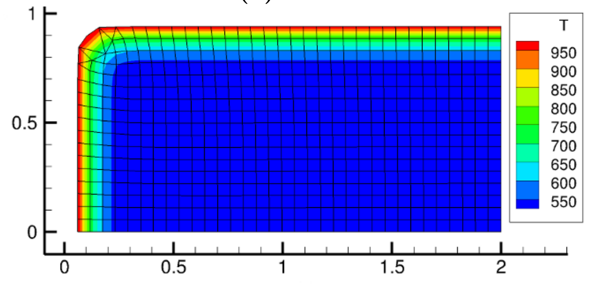
This demonstration case serves as an excellent illustration that even tangle-free mesh motion cannot handle just any random mesh haphazardly thrown at it. Some care and thought is required in designing meshes that play to the algorithm's strengths. In addition, we see that side sets must be specified carefully so that corners can be identified properly: we impose the same heat fluxes on Γ_1 and Γ_2, Γ_3 and Γ_4. It is tempting to combine these pairs and have only two side sets, but this will cause the corners not to be identified. In particular, there is no internal, non-boundary edge at any of the corners. Failure will occur on the first step when the scheme cannot locate such an edge, the direction on which internal nodes always move.



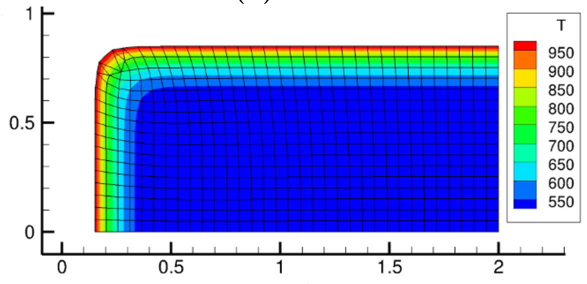
(a) $t \approx 13$



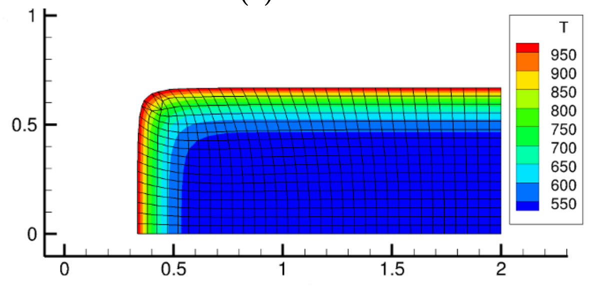
(b) $t \approx 20$



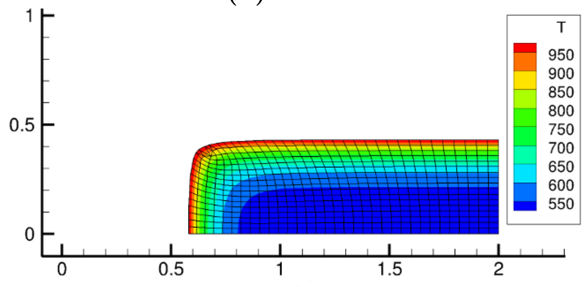
(c) $t \approx 75$



(d) $t \approx 150$



(c) $t \approx 300$



(d) $t \approx 500$

Figure 25. Doubly Receding Rectangle `CHAR` Simulation Results. (a) Early-stage recession: with heat flowing in from both top and bottom, the corner is hottest point. The value of Δs is highest at the quadrature point nearest to the corner. As a result, the face does not recede parallel to the original and begins to fold to a non-orthogonal peak as it recedes. (b) The corner retains the highest temperature and collapse exacerbates as time evolves. (c) The corner has seemingly collapsed and the element become a triangle. However, it actually remains a quadrilateral, just with a corner angle nearly $180°$. (d)-(f) Temperature across the two receding faces has equalized. The value of Δs is constant over the two side sets and the surface recedes evenly, retaining the shape of a rectangle with a rounded corner.

V. Three-Dimensional Meshes

The straightforward analysis possible in two-dimensions can be attributed to two key facts:

1. Each node can be contained in at most two elements that intersect the boundary non-trivially.
2. A vector's orthogonal complement can be constructed explicitly from its components in one of two manners. Choosing appropriately allows us to construct a symmetric transformation matrix $\mathcal{M}$.

Unfortunately, neither of these hold in three dimensions. Consequently, the script for analysis diverges significantly from that 2D. An example of how (1) above may be violated is depicted in the figure below.

As we did in 2D, we classify the boundary nodes by the number of side sets in which they are contained. However, the added geometric complexity the additional dimension makes classification into just corners and inlaid nodes insufficient. We must separate corners into two subtypes and distinguish three node classes:

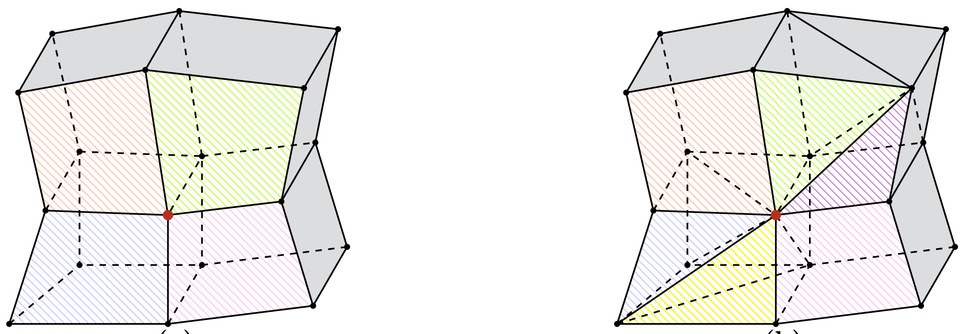


Figure 26. An Arbitrary Number of Boundary Faces: At the red node, well-placed hexahedra lead to the "nicest" case of four containing elements, shown in (a). Presence of mixed elements, such as tetrahedrons and pyramids in (b), complicates things considerably.

Definition (Node Classification, 3D).

A boundary node is a *pure corner* node if it is contained in three side sets. It is an *edge corner* node if it is contained in two side sets. An *inlaid* node is contained in a single side set.

A pure corner is every bit the traditional concept of "corner." An edge corner resides on a single topological edge (a pure corner lives on three) and demarcates its containing elements. It is, in that respect, a corner between elements but not a topological corner. Examples are shown below in Figure 27.

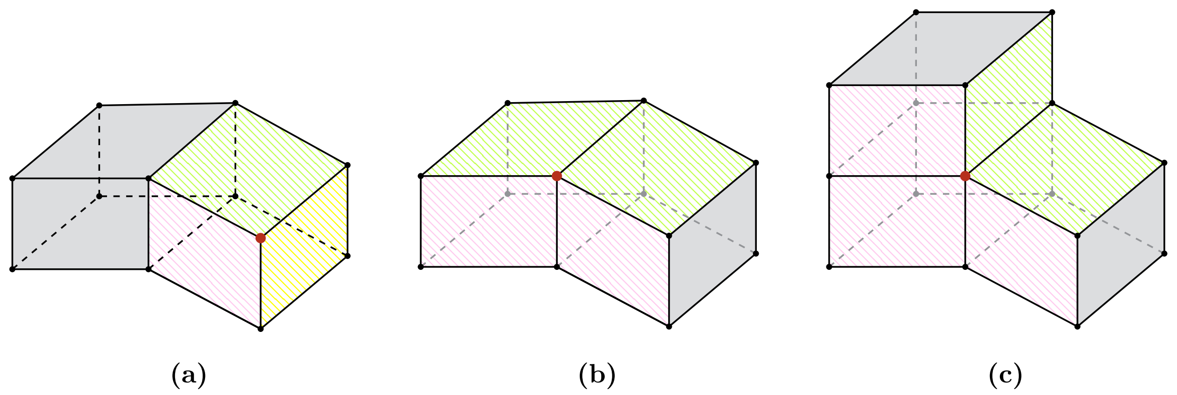


Figure 27. Corners in 3D. Each side set is indicated by a different color. (a) depicts a pure corner. In this example, all three containing faces are in the same element, but this is not always the case. (b) and (c) depict edge corners. A bevy of configurations is possible, but there can be only two side sets involved.

We follow the same outline as we did in 2D. After the general framework, a straightforward extension of the two-dimensional case, we derive the 3D analogs to $\hat{\mathbf{K}}_i$ and $\hat{\mathbf{E}}_i$. As one might expect, i now ranges from 1 to 3 because we may enforce up to three directions of motion. While construction of a symmetric transformation matrix $\mathcal{M}$ will not be possible as was in 2D, it can still be made orthogonal so that the $\mathcal{M}^{-1}$ can be obtained trivially. With that in place, we proceed through the three node class types and six intersection subcases. Demonstration of algorithm's effectiveness concludes the discussion. There is no verification this time.

A. General Framework

This section is the three-dimensional extension to the work of Section IV.A: define new shape functions and find the transformation matrix $\mathcal{M}$ that gives contributions to the Cartesian-coordinate stiffness matrix in terms of the new basis. As we did in 2D, we begin with the vector-valued shape functions on an element $\mathcal{T}$

$$\boldsymbol{\varphi}_k(\mathbf{x}) = \begin{cases} \psi_k(\mathbf{x})\,\mathbf{e}_1 & 1 \le k \le N \\ \psi_{k-N}(\mathbf{x})\,\mathbf{e}_2 & N+1 \le k \le 2N \\ \psi_{k-2N}(\mathbf{x})\,\mathbf{e}_3 & 2N+1 \le k \le 3N \end{cases} \tag{42}$$

and define the new shape functions

$$\tilde{\boldsymbol{\varphi}}_k(\mathbf{x}) = \begin{cases} \psi_k(\mathbf{x})\,\mathbf{p}_k & 1 \le k \le m \\ \psi_k(\mathbf{x})\,\mathbf{e}_1 & m+1 \le k \le N \\ \psi_{k-N}(\mathbf{x})\,\mathbf{q}_{k-N} & N+1 \le k \le m+N \\ \psi_{k-N}(\mathbf{x})\,\mathbf{e}_2 & m+N+1 \le k \le 2N \\ \psi_{k-2N}(\mathbf{x})\,\mathbf{r}_{k-2N} & 2N+1 \le k \le m+2N \\ \psi_{k-2N}(\mathbf{x})\,\mathbf{e}_3 & m+2N+1 \le k \le 3N \end{cases} \tag{43}$$

We mirror the procedure that led to Eq. (8) and seek the matrix $\mathcal{M}$ such that

$$\tilde{\boldsymbol{\varphi}}_k(\mathbf{x}) = \sum_{l=1}^{3N} \mathcal{M}_{kl}\boldsymbol{\varphi}_l(\mathbf{x})$$

As before, we suppose that the directions $\mathbf{p}_k$, $\mathbf{q}_k$, and $\mathbf{r}_k$ are known *a priori* and that the set spans $\mathbb{R}^3$. We further assume that this set is orthonormal so that no additional transforms be required. $\mathcal{M}$ has form

$$\mathcal{M} = \left[\begin{array}{c:c|c:c|c:c}
\begin{matrix} p_1^{(1)} & & 0 \\ & \ddots & \\ 0 & & p_m^{(1)} \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} p_1^{(2)} & & 0 \\ & \ddots & \\ 0 & & p_m^{(2)} \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} p_1^{(3)} & & 0 \\ & \ddots & \\ 0 & & p_m^{(3)} \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} \\ \hdashline
\begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} \\ \hline
\begin{matrix} q_1^{(1)} & & 0 \\ & \ddots & \\ 0 & & q_m^{(1)} \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} q_1^{(2)} & & 0 \\ & \ddots & \\ 0 & & q_m^{(2)} \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} q_1^{(3)} & & 0 \\ & \ddots & \\ 0 & & q_m^{(3)} \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} \\ \hdashline
\begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} \\ \hline
\begin{matrix} r_1^{(1)} & & 0 \\ & \ddots & \\ 0 & & r_m^{(1)} \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} r_1^{(2)} & & 0 \\ & \ddots & \\ 0 & & r_m^{(2)} \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} r_1^{(3)} & & 0 \\ & \ddots & \\ 0 & & r_m^{(3)} \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} \\ \hdashline
\begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{matrix}
\end{array}\right] \tag{44}$$

With each $\{\mathbf{p}_k, \mathbf{q}_k, \mathbf{r}_k\}$ orthonormal, $\mathcal{M}$ is an orthogonal matrix. This alone is sufficient to reduce substantially the level of involvement of the analysis compared to what is required of arbitrary direction sets.

B. Enforcing the Motion

As before, we seek the element stiffness matrix in the original basis vectors, obtained by reversing the transformations Eq. (11) and Eq. (13) that take us from $\boldsymbol{\varphi}_k$ to $\tilde{\boldsymbol{\varphi}}_k$:

$$\left.\begin{aligned} \mathbf{K} &= \mathcal{M}^{-T}\tilde{\mathbf{K}}\mathcal{M}^{-1} = \mathcal{M}\tilde{\mathbf{K}}\mathcal{M}^{T} \\ \mathbf{E} &= \mathcal{M}^{-1}\tilde{\mathbf{E}} = \mathcal{M}^{T}\tilde{\mathbf{E}} \end{aligned}\right\} \tag{45}$$

where the second equality follows from the orthogonality of $\mathcal{M}$. Separating from $\tilde{\mathbf{K}}$ and $\tilde{\mathbf{E}}$ perturbation $\mathbf{B}$ and $\mathbf{b}$ respectively, we construct the expressions corresponding to Eq. (16) in this higher dimension:

$$\begin{aligned} \mathbf{A}\big|_{\mathcal{T}} = \mathbf{K} \hookleftarrow \mathcal{M}\big(\tilde{\mathbf{K}} + \mathbf{B}_i\big)\mathcal{M}^T \quad &\Longrightarrow \quad \mathbf{A}\big|_{\mathcal{T}} = \mathbf{K} + \hat{\mathbf{K}}_i \qquad & \hat{\mathbf{K}}_i \triangleq \mathcal{M}\mathbf{B}_i\mathcal{M}^T \\ \mathbf{F}\big|_{\mathcal{T}} = \mathbf{E} \hookleftarrow \mathcal{M}^T\big(\tilde{\mathbf{E}} + \mathbf{b}_i\big) \quad &\Longrightarrow \quad \mathbf{F}\big|_{\mathcal{T}} = \mathbf{E} + \hat{\mathbf{E}}_i \qquad & \hat{\mathbf{E}}_i \triangleq \mathcal{M}^T\mathbf{b}_i \end{aligned}$$

The subscripts on $\mathbf{B}_i$ and $\mathbf{b}_i$ again indicate the number of degrees of freedom being set. For the coming sections, it is convenient to partition $\mathcal{M}$ along the solid lines of Eq. (44):

$$\mathcal{M} = \left[\begin{array}{cc|cc|cc} \mathbf{P}_1 & \mathbf{0} & \mathbf{P}_2 & \mathbf{0} & \mathbf{P}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{Q}_1 & \mathbf{0} & \mathbf{Q}_2 & \mathbf{0} & \mathbf{Q}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{R}_1 & \mathbf{0} & \mathbf{R}_2 & \mathbf{0} & \mathbf{R}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{array}\right]$$

The subscripts on $\mathbf{P}_i$, $\mathbf{Q}_i$, and $\mathbf{R}_i$ match the component (superscript) of $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{r}$, respectively, which supplies the matrix's entries:

$$\mathbf{P}_i = \operatorname{diag}\Big[p_1^{(i)}, \ldots, p_m^{(i)}\Big] \qquad \mathbf{Q}_i = \operatorname{diag}\Big[q_1^{(i)}, \ldots, q_m^{(i)}\Big] \qquad \mathbf{R}_i = \operatorname{diag}\Big[r_1^{(i)}, \ldots, r_m^{(i)}\Big]$$

As before, we assume that directions are ordered by importance: when one direction of motion is dictated, it will be $\mathbf{p}_k$ that is always enforced, not one of $\mathbf{q}_k$ or $\mathbf{r}_k$. Similarly, when two directions are prescribed, $\mathbf{r}_k$ will be the one left behind. In the next three subsections, we detail single, double, and triple enforcement of these directions. These will be used in the algorithms of Section V.C as follows:

Pure Corner Nodes	Triple Enforcement
Edge Corner Nodes: Receder/Receder	Triple Enforcement
Edge Corner Nodes: Receder/Slider	Triple Enforcement
Edge Corner Nodes: Slider/Slider	Double Enforcement
Inlaid Nodes: Receder	Triple Enforcement
Inlaid Nodes: Slider	Single Enforcement

1. *Single Enforcement*

The perturbations are identical to those in Section IV.1, except with a few additional blocks of zeros to account for the increase in dimension

$$\mathbf{B}_1 \triangleq \varepsilon^{-1}\left[\begin{array}{cc|cc|cc} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right] \qquad \mathbf{b}_1 \triangleq \varepsilon^{-1}\begin{bmatrix} \hat{\mathbf{s}}(\mathbf{p}_1, \ldots, \mathbf{p}_m) \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

It is not difficult to see the extension from the lower dimensional case:

$$
\hat{\mathbf{K}}_1 = \frac{1}{\varepsilon}\left[\begin{array}{cccccc|cccccc|cccccc}
\left[p_1^{(1)}\right]^2 & & & & & 0 & p_1^{(1)}q_1^{(1)} & & & & & 0 & p_1^{(1)}r_1^{(1)} & & & & & 0 \\
 & \ddots & & & & & & \ddots & & & & & & \ddots & & & & \\
 & & \left[p_m^{(1)}\right]^2 & & & & & & p_m^{(1)}q_m^{(1)} & & & & & & p_m^{(1)}r_m^{(1)} & & & \\
 & & & 0 & & & & & & 0 & & & & & & 0 & & \\
 & & & & \ddots & & & & & & \ddots & & & & & & \ddots & \\
0 & & & & & 0 & 0 & & & & & 0 & 0 & & & & & 0 \\
\hline
p_1^{(1)}q_1^{(1)} & & & & & 0 & \left[q_1^{(2)}\right]^2 & & & & & 0 & q_1^{(1)}r_1^{(1)} & & & & & 0 \\
 & \ddots & & & & & & \ddots & & & & & & \ddots & & & & \\
 & & p_m^{(1)}q_m^{(1)} & & & & & & \left[q_m^{(1)}\right]^2 & & & & & & q_m^{(1)}r_m^{(1)} & & & \\
 & & & 0 & & & & & & 0 & & & & & & 0 & & \\
 & & & & \ddots & & & & & & \ddots & & & & & & \ddots & \\
0 & & & & & 0 & 0 & & & & & 0 & 0 & & & & & 0 \\
\hline
p_1^{(1)}r_1^{(1)} & & & & & 0 & q_1^{(1)}r_1^{(1)} & & & & & 0 & \left[r_m^{(1)}\right]^2 & & & & & 0 \\
 & \ddots & & & & & & \ddots & & & & & & \ddots & & & & \\
 & & p_m^{(1)}r_m^{(1)} & & & & & & q_m^{(1)}r_m^{(1)} & & & & & & \left[r_m^{(1)}\right]^2 & & & \\
 & & & 0 & & & & & & 0 & & & & & & 0 & & \\
 & & & & \ddots & & & & & & \ddots & & & & & & \ddots & \\
0 & & & & & 0 & 0 & & & & & 0 & 0 & & & & & 0
\end{array}\right]
\qquad
\hat{\mathbf{E}}_1 = \frac{1}{\varepsilon}\begin{bmatrix}
\hat{s}(\mathbf{x}_1)p_1^{(1)} \\ \vdots \\ \hat{s}(\mathbf{x}_m)p_m^{(1)} \\
\left.\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}\right\} N-m \\
\hat{s}(\mathbf{x}_1)p_1^{(2)} \\ \vdots \\ \hat{s}(\mathbf{x}_m)p_m^{(2)} \\
\left.\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}\right\} N-m \\
\hat{s}(\mathbf{x}_1)p_1^{(3)} \\ \vdots \\ \hat{s}(\mathbf{x}_m)p_m^{(3)} \\
\left.\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}\right\} N-m
\end{bmatrix}
$$

2. Double Enforcement

When the first two directions of motion are enforced, the perturbations take the form

$$
\mathbf{P}_2 \triangleq \varepsilon^{-1}\left[\begin{array}{cc|cc|cc}
\mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\hline
\mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\hline
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}
\end{array}\right]
\qquad
\mathbf{B}_2 \triangleq \varepsilon^{-1}\begin{bmatrix}
\hat{\mathbf{s}}(\mathbf{p}_1,\ldots,\mathbf{p}_m) \\ \mathbf{0} \\ \hat{\mathbf{s}}(\mathbf{q}_1,\ldots,\mathbf{q}_m) \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0}
\end{bmatrix}
$$

Rather than merely assert the straightforwardness of extending the two-dimensional result as we did in the preceding section, it is useful to write out the multiplication:

$$
\hat{\mathbf{K}}_2 = \varepsilon^{-1}\left[\begin{array}{cc|cc|cc}
\mathbf{P}_1^2+\mathbf{P}_2^2 & \mathbf{0} & \mathbf{P}_1\mathbf{Q}_1+\mathbf{P}_2\mathbf{Q}_2 & \mathbf{0} & \mathbf{P}_1\mathbf{R}_1+\mathbf{P}_2\mathbf{R}_2 & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\hline
\mathbf{Q}_1\mathbf{P}_1+\mathbf{Q}_2\mathbf{P}_2 & \mathbf{0} & \mathbf{Q}_1^2+\mathbf{Q}_2^2 & \mathbf{0} & \mathbf{Q}_1\mathbf{R}_1+\mathbf{Q}_2\mathbf{R}_2 & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\hline
\mathbf{R}_1\mathbf{P}_1+\mathbf{R}_2\mathbf{P}_2 & \mathbf{0} & \mathbf{R}_1\mathbf{Q}_1+\mathbf{R}_2\mathbf{Q}_2 & \mathbf{0} & \mathbf{R}_1^2+\mathbf{R}_2^2 & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}
\end{array}\right]
$$

Consider the first diagonal block. We note that normality gives us that

$$
\left[\mathbf{P}_1^2+\mathbf{P}_2^2\right]_{ii} = \left[p_i^{(1)}\right]^2 + \left[p_i^{(2)}\right]^2 = 1 - \left[p_i^{(3)}\right]^2
$$

and similarly for the third and fifth diagonal blocks. Orthogonality of $\{\mathbf{p}_k,\mathbf{q}_k,\mathbf{r}_k\}$ provides the relations

$$
\left.\begin{aligned}
\langle\mathbf{p}_k,\mathbf{q}_k\rangle &= p_k^{(1)}q_k^{(1)} + p_k^{(2)}q_k^{(2)} + p_k^{(3)}q_k^{(3)} = 0 \\
\langle\mathbf{p}_k,\mathbf{r}_k\rangle &= p_k^{(1)}r_k^{(1)} + p_k^{(2)}r_k^{(2)} + p_k^{(3)}r_k^{(3)} = 0 \\
\langle\mathbf{q}_k,\mathbf{r}_k\rangle &= q_k^{(1)}r_k^{(1)} + q_k^{(2)}r_k^{(2)} + q_k^{(3)}r_k^{(3)} = 0
\end{aligned}\right\} \qquad (46)
$$

Rearranging slightly Eq. (46) by moving the last term to the rightmost side allows us to simplify $\hat{\mathbf{K}}_2$ nicely:

$$\hat{\mathbf{K}}_2 = \frac{1}{\varepsilon} \left[\begin{array}{c|c|c}
\begin{matrix} 1-\left[p_1^{(3)}\right]^2 &&&&& 0 \\ & \ddots &&&& \\ && 1-\left[p_m^{(3)}\right]^2 &&& \\ &&& 0 && \\ &&&& \ddots & \\ 0 &&&&& 0 \end{matrix} &
\begin{matrix} -p_1^{(3)}q_1^{(3)} &&&&& 0 \\ & \ddots &&&& \\ && -p_m^{(3)}q_m^{(3)} &&& \\ &&& 0 && \\ &&&& \ddots & \\ 0 &&&&& 0 \end{matrix} &
\begin{matrix} -p_1^{(3)}r_1^{(3)} &&&&& 0 \\ & \ddots &&&& \\ && -p_m^{(3)}r_m^{(3)} &&& \\ &&& 0 && \\ &&&& \ddots & \\ 0 &&&&& 0 \end{matrix} \\ \hline
\begin{matrix} -p_1^{(3)}q_1^{(3)} &&&&& 0 \\ & \ddots &&&& \\ && -p_m^{(3)}q_m^{(3)} &&& \\ &&& 0 && \\ &&&& \ddots & \\ 0 &&&&& 0 \end{matrix} &
\begin{matrix} 1-\left[q_1^{(3)}\right]^2 &&&&& 0 \\ & \ddots &&&& \\ && 1-\left[q_m^{(3)}\right]^2 &&& \\ &&& 0 && \\ &&&& \ddots & \\ 0 &&&&& 0 \end{matrix} &
\begin{matrix} -q_1^{(3)}r_1^{(3)} &&&&& 0 \\ & \ddots &&&& \\ && -q_m^{(3)}r_m^{(3)} &&& \\ &&& 0 && \\ &&&& \ddots & \\ 0 &&&&& 0 \end{matrix} \\ \hline
\begin{matrix} -p_1^{(3)}r_1^{(3)} &&&&& 0 \\ & \ddots &&&& \\ && -p_m^{(3)}r_m^{(3)} &&& \\ &&& 0 && \\ &&&& \ddots & \\ 0 &&&&& 0 \end{matrix} &
\begin{matrix} -q_1^{(3)}r_1^{(3)} &&&&& 0 \\ & \ddots &&&& \\ && -q_m^{(3)}r_m^{(3)} &&& \\ &&& 0 && \\ &&&& \ddots & \\ 0 &&&&& 0 \end{matrix} &
\begin{matrix} 1-\left[r_m^{(3)}\right]^2 &&&&& 0 \\ & \ddots &&&& \\ && 1-\left[r_m^{(3)}\right]^2 &&& \\ &&& 0 && \\ &&&& \ddots & \\ 0 &&&&& 0 \end{matrix}
\end{array}\right]$$

Unfortunately, the orthogonality relations are of no help in in our quest to simplify $\hat{\mathbf{E}}_2$. In fact, there is nothing we can do to improve its elegance. For compactness, we write it in block notation

$$\hat{\mathbf{E}}_2 = \varepsilon^{-1} \begin{bmatrix} \mathbf{P}_1\hat{\mathbf{s}}(\mathbf{p}_1,\ldots,\mathbf{p}_m) + \mathbf{Q}_1\hat{\mathbf{s}}(\mathbf{q}_1,\ldots,\mathbf{q}_m) \\ \mathbf{0} \\ \mathbf{P}_2\hat{\mathbf{s}}(\mathbf{p}_1,\ldots,\mathbf{p}_m) + \mathbf{Q}_2\hat{\mathbf{s}}(\mathbf{q}_1,\ldots,\mathbf{q}_m) \\ \mathbf{0} \\ \mathbf{P}_3\hat{\mathbf{s}}(\mathbf{p}_1,\ldots,\mathbf{p}_m) + \mathbf{Q}_3\hat{\mathbf{s}}(\mathbf{q}_1,\ldots,\mathbf{q}_m) \\ \mathbf{0} \end{bmatrix}$$

but for clarity explicate a general nonzero block:

$$\mathbf{P}_i\hat{\mathbf{s}}(\mathbf{p}_1,\ldots,\mathbf{p}_m) + \mathbf{Q}_i\hat{\mathbf{s}}(\mathbf{q}_1,\ldots,\mathbf{q}_m) = \begin{bmatrix} \hat{s}(\mathbf{x}_1;\mathbf{p}_1)p_1^{(i)} + \hat{s}(\mathbf{x}_1;\mathbf{q}_1)q_1^{(i)} \\ \vdots \\ \hat{s}(\mathbf{x}_m;\mathbf{p}_m)p_m^{(i)} + \hat{s}(\mathbf{x}_m;\mathbf{q}_m)q_m^{(i)} \end{bmatrix}$$

3. Triple Enforcement

When the all three directions of motion are enforced, we have

$$\mathbf{P}_3 \triangleq \varepsilon^{-1} \left[\begin{array}{cc|cc|cc} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right] \qquad \mathbf{B}_3 \triangleq \varepsilon^{-1} \begin{bmatrix} \hat{\mathbf{s}}(\mathbf{p}_1,\ldots,\mathbf{p}_m) \\ \mathbf{0} \\ \hat{\mathbf{s}}(\mathbf{q}_1,\ldots,\mathbf{q}_m) \\ \mathbf{0} \\ \hat{\mathbf{s}}(\mathbf{r}_1,\ldots,\mathbf{r}_m) \\ \mathbf{0} \end{bmatrix}$$

The multiplication here is a straightforward but involved version of the one that produced $\hat{\mathbf{K}}_2$:

$$\hat{\mathbf{K}}_3 = \varepsilon^{-1}\left[\begin{array}{cc|cc|cc}
\mathbf{P}_1^2+\mathbf{P}_2^2+\mathbf{P}_3^2 & \mathbf{0} & \mathbf{P}_1\mathbf{Q}_1+\mathbf{P}_2\mathbf{Q}_2+\mathbf{P}_3\mathbf{Q}_3 & \mathbf{0} & \mathbf{P}_1\mathbf{R}_1+\mathbf{P}_2\mathbf{R}_2+\mathbf{P}_3\mathbf{R}_3 & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline
\mathbf{Q}_1\mathbf{P}_1+\mathbf{Q}_2\mathbf{P}_2+\mathbf{Q}_3\mathbf{P}_3 & \mathbf{0} & \mathbf{Q}_1^2+\mathbf{Q}_2^2+\mathbf{Q}_3^2 & \mathbf{0} & \mathbf{Q}_1\mathbf{R}_1+\mathbf{Q}_2\mathbf{R}_2+\mathbf{Q}_3\mathbf{R}_3 & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline
\mathbf{R}_1\mathbf{P}_1+\mathbf{R}_2\mathbf{P}_2+\mathbf{R}_3\mathbf{P}_3 & \mathbf{0} & \mathbf{R}_1\mathbf{Q}_1+\mathbf{R}_2\mathbf{Q}_2+\mathbf{R}_3\mathbf{Q}_3 & \mathbf{0} & \mathbf{R}_1^2+\mathbf{R}_2^2+\mathbf{R}_3^2 & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}
\end{array}\right]$$

Thanks to the orthogonality relations Eq. (46), the off-diagonal blocks vanish and the nonzero on-diagonal blocks become identity matrices. Triple enforcement in 3D therefore produces a correction matrix whose simplicity equals that of double enforcement in 2D. Regrettably, the parallels extend through the entirety of the derivation, as $\hat{\mathbf{E}}_3$ is just as cumbersome to express as the right-side correction in Section IV.2.

$$\hat{\mathbf{K}}_3 = \frac{1}{\varepsilon}\left[\begin{array}{c:c|c:c|c:c}
\begin{matrix}1&&\\&\ddots&\\&&1\end{matrix} & & \begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} & & \begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} & \\ \hdashline
 & \begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} & & \begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} & & \begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} \\ \hline
\begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} & & \begin{matrix}1&&\\&\ddots&\\&&1\end{matrix} & & \begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} & \\ \hdashline
 & \begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} & & \begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} & & \begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} \\ \hline
\begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} & & \begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} & & \begin{matrix}1&&\\&\ddots&\\&&1\end{matrix} & \\ \hdashline
 & \begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} & & \begin{matrix}0&&\\&\ddots&\\&&0\end{matrix} & & \begin{matrix}0&&\\&\ddots&\\&&0\end{matrix}
\end{array}\right]\begin{array}{l}\}\,m\\ \\ \}\,m\\ \\ \}\,m\\ \\ \end{array}
\qquad
\hat{\mathbf{E}}_3 = \frac{1}{\varepsilon}\left[\begin{array}{c}
\big\langle \mathbf{d}_{1,1},\vec{\mathbf{s}}_1\big\rangle \\ \vdots \\ \big\langle \mathbf{d}_{1,m},\vec{\mathbf{s}}_m\big\rangle \\
0 \\ \vdots \\ 0 \\
\big\langle \mathbf{d}_{2,1},\vec{\mathbf{s}}_1\big\rangle \\ \vdots \\ \big\langle \mathbf{d}_{2,m},\vec{\mathbf{s}}_m\big\rangle \\
0 \\ \vdots \\ 0 \\
\big\langle \mathbf{d}_{3,1},\vec{\mathbf{s}}_1\big\rangle \\ \vdots \\ \big\langle \mathbf{d}_{3,m},\vec{\mathbf{s}}_m\big\rangle \\
0 \\ \vdots \\ 0
\end{array}\right]\begin{array}{l} \\ \\ \\ \}\,N-m \\ \\ \\ \\ \}\,N-m \\ \\ \\ \\ \}\,N-m \end{array}$$

$$\underbrace{\qquad}_{m}\qquad\underbrace{\qquad}_{m}\qquad\underbrace{\qquad}_{m}$$

where we have introduced, for shorthand, the quantities

$$\mathbf{d}_{i,k} \triangleq \left[p_k^{(i)}, q_k^{(i)}, r_k^{(i)}\right]^T \qquad\qquad \vec{\mathbf{s}}_i \triangleq \left[\hat{s}(\mathbf{x}_i;\mathbf{p}_i), \hat{s}(\mathbf{x}_i;\mathbf{q}_i), \hat{s}(\mathbf{x}_i;\mathbf{r}_i)\right]^T$$

C. Determining the Directions

Section V.B assumes that all the directions $\mathbf{p}_k$, $\mathbf{q}_k$, and $\mathbf{r}_k$ and the corresponding $\hat{s}$ values are known *a priori*. Constructing the direction and recession pairs $\big(\mathbf{p}_k, \hat{s}(\mathbf{x}_k;\mathbf{p}_k)\big)$, $\big(\mathbf{q}_k, \hat{s}(\mathbf{x}_k;\mathbf{q}_k)\big)$, and $\big(\mathbf{r}_k, \hat{s}(\mathbf{x}_k;\mathbf{r}_k)\big)$ at each node $\mathbf{x}_k$ from Δs and $\boldsymbol{\nu}$ at quadrature points is non-trivial and is the focus of this section.

To illustrate the challenges we face, consider Figure 28 below. It depicts a parallelepiped with a single receding face (colored red); the faces adjacent to it are sliding faces and the remaining one (parallelogram in the back) is fixed. On physical grounds, we expect the parallelogram face to melt away and the element to kept its general shape through recession. Figure 29 above highlights the difficulty in choosing directions of recession that lead to the physically correct result. At edge corners (in blue), we require that the node remain on the sliding face; this preserves the parallelepiped's exoskeleton formed by the four corner nodes.

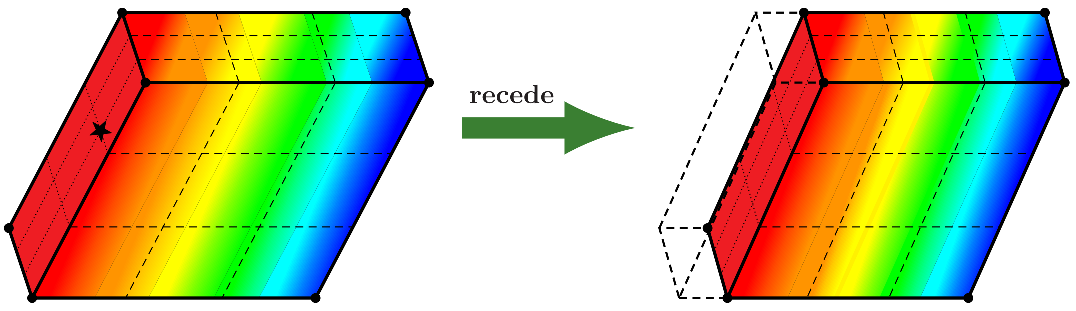


Figure 28. Physically Correct Recession. If the temperature over the parallelogram's surface (red) is uniform, the surface recedes parallel to its current plane of existence (right).

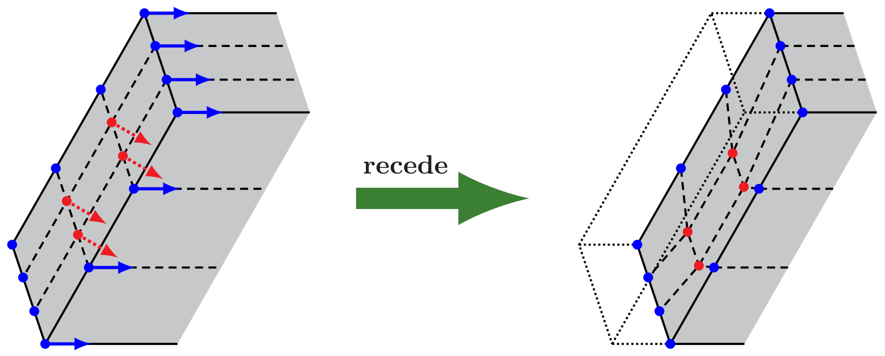


Figure 29. "Right" Direction Goes Wrong. Edge corners (blue nodes) should move in the sliding plane only for shape preservation. At the red inlaid nodes, the "natural" direction of motion is the normal vector. The result of this motion is on the left.

At the inalid nodes (in red), we have no information of the existence of a sliding face because all analysis is done on a per-element basis. Furthermore, even if we did know of the sliding side sets, there is one in each cardinal direction–which is the "right one"? As such, it seems that the natural direction for inlaid nodes is the normal vector. However, the right side of the figure reveals that such motion causes mesh quality to deteriorate. Continued motion along the normal sets up a future unavoidable collision with an edge.

1. Pure Corner Nodes

Perhaps contrary to expectations, the simplest case is the pure corner. A typical situation is depicted below:

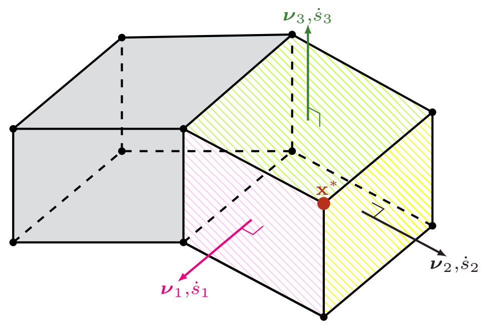

$\boldsymbol{\nu}_3, \dot{s}_3$
$\mathbf{x}^*$
$\boldsymbol{\nu}_2, \dot{s}_2$
$\boldsymbol{\nu}_1, \dot{s}_1$

Figure 30. Typical Pure Corner Case. The three different colors indicate the three side sets that contain the pure corner $\mathbf{x}^*$. The normal vectors for each side are indicated.

If any of the three side sets is of motion type "stationary," the node should remain fixed in place: set

$\mathbf{p} = \mathbf{e}_1$, $\mathbf{q} = \mathbf{e}_2$, $\mathbf{r} = \mathbf{e}_3$ and $\hat{s}(\mathbf{x}^*; \cdot) \equiv 0$ and move on. Only combinations sliding and receding side sets are considered from this point. Unlike in the coming edge corners, we can analyze all cases at once and need not be concerned about the recession nature of each side set involved.

It is tempting to choose the face normals as directions of motion (as in Figure 30). However, $\dot{s}$ is known only at quadrature points; choosing the value of $\dot{s}$ at the closest quadrature point is physically incorrect and likely to lead to poor mesh quality and skewed geometry. Instead, the solution is, in essence, to move entire side sets: we collect the face quadrature points from each face containing $\mathbf{x}^*$ in the side set, move each according to its known values of $\boldsymbol{\nu}$ and $\dot{s}$, and then compute the best-fit plane through the resultant points. We then find the intersection of these three planes, guaranteed to be unique except in degenerate cases.

The process is prescribed precisely in Algorithm 4 below:

Algorithm 4 (Pure Corner Motion).

1. Choose a side set S_j which contains $\mathbf{x}^*$. Let $\mathcal{F}_1, \ldots, \mathcal{F}_{m_j} \in S_j$ be the faces containing $\mathbf{x}^*$.
2. Let $\mathbf{z}_{k,1}, \ldots, \mathbf{z}_{k,q_k} \in \mathcal{F}_k$ be the quadrature points of the k^{th} face. Let $\boldsymbol{\nu}_{k,i}$ and $\Delta s_{k,i}$ be the normal and recession value, respectively, specified at the quadrature point $\mathbf{z}_{k,i}$.
3. Compute the centroid of the collection $\bigcup_{k=1}^{m_j} \left\{\mathbf{z}_{k,i} + \Delta s_{k,i}\boldsymbol{\nu}_{k,i}\right\}_{i=1}^{q_k}$:
$$\mathbf{c}_j \triangleq \left(\sum_{k=1}^{m_j} q_k\right)^{-1} \sum_{k=1}^{m_j}\sum_{i=1}^{q_k} \left[\mathbf{z}_{k,i} + \Delta s_{k,i}\boldsymbol{\nu}_{k,i}\right]$$
4. For each $k = 1, \ldots, m_j$, form the $3 \times q_k$ matrix
$$\mathbf{A}_k \triangleq \left[\begin{array}{c|c|c} \mathbf{z}_{k,1} + \Delta s_{k,1}\boldsymbol{\nu}_{k,1} - \mathbf{c}_j & \cdots & \mathbf{z}_{k,q_k} + \Delta s_{k,q_k}\boldsymbol{\nu}_{k,q_k} - \mathbf{c}_j \end{array}\right]$$
and glue all of these together to form the $3 \times Q_j$ matrix, where $Q_j \triangleq \prod_{k=1}^{m_j} q_k$
$$\mathbf{A} = \left[\begin{array}{ccc} \mathbf{A}_1 & \cdots & \mathbf{A}_{m_j} \end{array}\right]$$
5. Compute the singular value decomposition of $\mathbf{A}$:
$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$$
where $\mathbf{U} \in \mathbb{R}^{3\times 3}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{3\times Q_j}$, and $\mathbf{V} \in \mathbb{R}^{Q_j \times Q_j}$. Let $\mathbf{d}_1 \triangleq \mathbf{u}_3$, the 3rd column of $\mathbf{U}$.
6. Repeat Steps 1-5 for the remaining two side sets to obtain $\mathbf{c}_2$, $\mathbf{d}_2$, $\mathbf{c}_3$, and $\mathbf{d}_3$.
7. Solve the linear system
$$\begin{bmatrix} \mathbf{d}_1^T \\ \mathbf{d}_2^T \\ \mathbf{d}_3^T \end{bmatrix} \mathbf{x}_{\text{new}} = \begin{bmatrix} \langle \mathbf{d}_1, \mathbf{c}_1 \rangle \\ \langle \mathbf{d}_2, \mathbf{c}_2 \rangle \\ \langle \mathbf{d}_3, \mathbf{c}_3 \rangle \end{bmatrix} \tag{47}$$
8. Set $\mathbf{a} \triangleq \mathbf{x}^*_{\text{new}} - \mathbf{x}^*$ and $\hat{s} \triangleq |\mathbf{a}|$. Compute the singular value decomposition
$$\hat{\mathbf{a}} = \hat{\mathbf{U}}\hat{\boldsymbol{\Sigma}}\hat{\mathbf{V}}^T$$
9. Set $\mathbf{p} \triangleq (\operatorname{sgn}\langle \mathbf{a}, \hat{\mathbf{u}}_1 \rangle)\, \hat{\mathbf{u}}_1$, $\mathbf{q} \triangleq \hat{\mathbf{u}}_2$, $\mathbf{r} \triangleq \hat{\mathbf{u}}_3$, where $\hat{\mathbf{u}}_i$ is the i^{th} column of $\hat{\mathbf{U}}$.

(continued).

10. Follow Section V.B with the direction and recession pairs $(\mathbf{p}, \hat{s})$, $(\mathbf{q}, 0)$, $(\mathbf{r}, 0)$, all enforced.

Steps 1 and 2 are clear: collect all faces containing $\mathbf{x}^*$ along with the quadrature points and recession values and normals at those points. From the collection $\bigcup_{k=1}^{m_j}\{\mathbf{z}_{k,i} + \Delta s_{k,i}\boldsymbol{\nu}_{k,i}\}_{i=1}^{q_k}$, we extract a "spanning" single plane in Steps 3–6. As we did in the 2D corner case (Algorithm 1), we compute the best-fit plane with help from the SVD and Lemma 2. The best-fit plane is spanned by the first two columns of $\mathbf{U}$; the third column therefore gives the normal to this plane. We know a point on this plane thanks to Lemma 1 so the normal completes the set of information sufficient to characterize the plane completely.

Each plane generated in Step 5 is described by the equation $\langle \mathbf{x} - \mathbf{c}_i, \mathbf{d}_i \rangle = 0$. Writing out these three dot products and casting into matrix form leads to Eq. (47), the solvability of which is aided by the following:

Lemma 3.

The normal vectors $\boldsymbol{\nu}_1$, $\boldsymbol{\nu}_2$, and $\boldsymbol{\nu}_3$ of adjacent faces of a tetrahedron span $\mathbb{R}^3$.

Proof.

Let $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ be the edge vectors at the vertex formed by the intersection of the faces having normals $\boldsymbol{\nu}_1$, $\boldsymbol{\nu}_2$, and $\boldsymbol{\nu}_3$, arranged such that $\boldsymbol{\nu}_1 = \mathbf{a}\times\mathbf{b}$, $\boldsymbol{\nu}_2 = \mathbf{b}\times\mathbf{c}$, and $\boldsymbol{\nu}_3 = \mathbf{a}\times\mathbf{c}$. Because all vectors lie in $\mathbb{R}^3$, we have that

$$\det\left[\ \boldsymbol{\nu}_1 \ \middle|\ \boldsymbol{\nu}_2 \ \middle|\ \boldsymbol{\nu}_3\ \right] = \boldsymbol{\nu}_1 \cdot (\boldsymbol{\nu}_2 \times \boldsymbol{\nu}_3) \tag{48}$$

Expanding $\boldsymbol{\nu}_2$ and $\boldsymbol{\nu}_3$ in terms of $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$, and applying the identity for the the vector triple product $\mathbf{a}' \times (\mathbf{b}' \times \mathbf{c}') = \mathbf{b}'(\mathbf{a}'\cdot\mathbf{c}') - \mathbf{c}'(\mathbf{a}'\cdot\mathbf{b}')$, we have

$$\boldsymbol{\nu}_2 \times \boldsymbol{\nu}_3 = \big(\mathbf{a}\cdot(\mathbf{b}\times\mathbf{c})\big)\mathbf{c}$$

Hence,

$$\boldsymbol{\nu}_1 \cdot (\boldsymbol{\nu}_2 \times \boldsymbol{\nu}_3) = \big[(\mathbf{a}\times\mathbf{b})\cdot\mathbf{c}\big]\big[(\mathbf{b}\times\mathbf{c})\cdot\mathbf{a}\big] = \big[(\mathbf{a}\times\mathbf{b})\cdot\mathbf{c}\big]^2$$

by the nature of the scalar triple product. The volume of the tetrahedron is $\frac{1}{6}\left|(\mathbf{a}\times\mathbf{b})\cdot\mathbf{c}\right|$, so if the lefthand side is zero, then the tetrahedron is vacuous—a contradiction. Accordingly, it follows that $\boldsymbol{\nu}_1$, $\boldsymbol{\nu}_2$, and $\boldsymbol{\nu}_3$ are linearly independent and hence span $\mathbb{R}^3$. ■

For hexahedra and pyramids, we can construct a tetrahedron contained in the original polygon that overlaps with the original adjacent faces in question, thus extending the above result to these polygons. The lemma itself, of course, is not directly applicable to Eq. (47). The vectors that determine Eq. (47) are normal vectors from planes not known *a priori* to form a polygon. However, since the new plane is a best-fit of perturbed points from the original and the determinant is a continuous map that measures volume, we can reasonably expect Eq. (48) to remain nonzero. The contrary requires motion of one face so dramatic that it becomes parallel to another, highly unlikely unless this were nearly true originally (e.g., a highly obtuse corner).

The intersection point determined in Step 7 is the location of the new node. Step 8 computes an artificial direction of motion $\mathbf{a}$, directed toward the new location from the old. The recession value $\hat{s}$ is the length of this vector; SVD provides both a normalized version of $\mathbf{a}$ and a basis for $\{\mathbf{a}\}^\perp$. We must adjust for sign, since $\hat{\mathbf{u}}_1 = \pm\left|\mathbf{a}\right|^{-1}\mathbf{a}$. It is crucial that the direction be pointed toward $\mathbf{x}_{\text{new}}$ to be consistent with Δs.

A consequence of the algorithm is that if all three side sets are sliding, the node will remain fixed in place because there is no motion created by any of the faces containing this. This conveniently creates "anchor points" and removes burden to declare at least one stationary side set from the user.

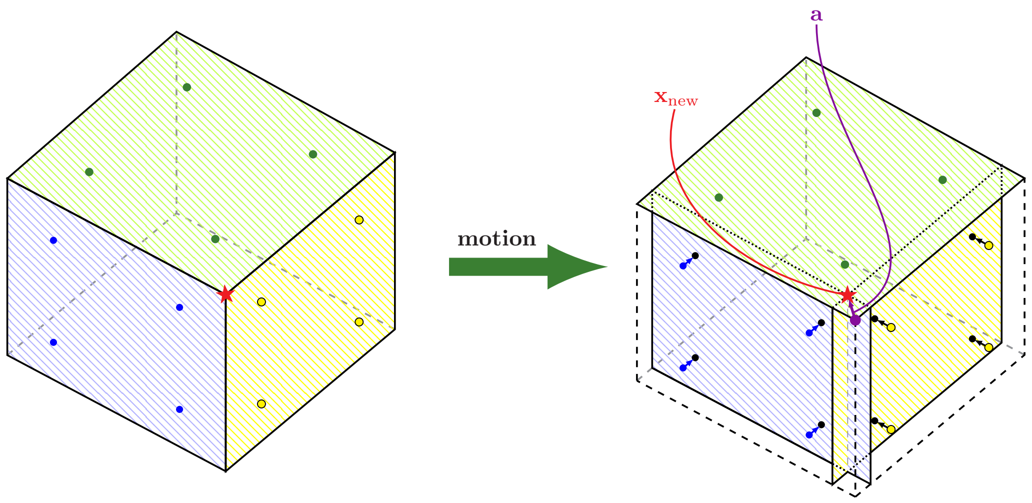


Figure 31. Illustration of Algorithm 4. This figure depicts Algorithm 4 in action on a simple example. The geometry before motion is shown on the left. The blue (left) and yellow (right) faces are receding, with uniform recession within each face. The green (top) face is sliding. The left diagram shows the motion of the node: it moves from the purple circle to the red star $\mathbf{x}_{\text{new}}$. The direction (but not magnitude) $\mathbf{a}$ is indicated. The length of this vector gives $\hat{s}$.

2. Edge Corner Nodes

Recall that an edge corner node is one that is contained in two different side sets. Geometry can differ wildly among instances of edge corners (two simple examples are depicted in Figure 27). We design for "typical" cases such as those in the figure and place some burden on the user to have designed a reasonable mesh.

We identify three possible configurations: when both side sets are receding, when one is sliding and the other receding, and when both are sliding. When one set is stationary, we set all motion to zero as we did before.

3. Edge Corner: Receder/Receder

We begin with the most complex of the cases: double receders. This configuration causes the most issues for simple mesh motion schemes; success in handling this case is therefore a hallmark of the tangle-free mesh motion scheme. The algorithm is similar to the pure corner case. As before, we collect faces containing $\mathbf{x}^*$ within each side set, apply recession at the quadrature points, and compute the best-fit plane. To replace the missing third side, we find the internal (non-boundary) faces containing $\mathbf{x}^*$ and compute intersections of the two best-fit planes with the each plane coinciding with these faces. The new node is the centroid of all such intersection points. Algorithm 5 below details this procedure mathematically.

Algorithm 5 (Edge Corner Motion: Receder/Receder).

1. Let $\mathcal{F}_1, \ldots, \mathcal{F}_m$ be the boundary faces containing $\mathbf{x}^*$ in the first (receding) side set; $\mathcal{F}'_1, \ldots, \mathcal{F}'_{m'}$, in the second.
2. Let $\mathbf{z}_{k,1}, \ldots, \mathbf{z}_{k,q_k} \in \mathcal{F}_k$ be the quadrature points of the kth face. Let $\boldsymbol{\nu}_{k,i}$ and $\Delta s_{k,i}$ be the normal and recession value, respectively, specified at $\mathbf{z}_{k,i}$.
3. Compute the centroid of the collection $\bigcup_{k=1}^{m} \left\{\mathbf{z}_{k,i} + \Delta s_{k,i} \boldsymbol{\nu}_{k,i}\right\}_{i=1}^{q_k}$:

$$\mathbf{c} \triangleq \left(\sum_{k=1}^{m} q_k\right)^{-1} \sum_{k=1}^{m} \sum_{i=1}^{q_k} \left[\mathbf{z}_{k,i} + \Delta s_{k,i} \boldsymbol{\nu}_{k,i}\right]$$

(continued).

4. For each $k = 1, \dots, m$, form the $3 \times q_k$ matrix

$$\mathbf{A}_k \triangleq \left[\ \mathbf{z}_{k,1} + \Delta s_{k,1} \boldsymbol{\nu}_{k,1} - \mathbf{c}\ \middle|\ \cdots\ \middle|\ \mathbf{z}_{k,q_k} + \Delta s_{k,q_k} \boldsymbol{\nu}_{k,q_k} - \mathbf{c}\ \right]$$

and glue all of these together to form the $3 \times Q$ matrix, where $Q \triangleq \prod_{k=1}^{m} q_k$,

$$\mathbf{A} = \left[\ \mathbf{A}_1 \quad \cdots \quad \mathbf{A}_m\ \right]$$

5. Compute the singular value decomposition of $\mathbf{A}$:

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$$

where $\mathbf{U} \in \mathbb{R}^{3\times 3}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{3\times Q}$, and $\mathbf{V} \in \mathbb{R}^{Q\times Q}$. Let $\mathcal{P}$ be the plane with normal vector $\boldsymbol{\nu} \triangleq \mathbf{u}_3$, the 3rd column of $\mathbf{U}$, passing through the point $\mathbf{c}$.

6. Repeat steps 2-5 with $\mathbf{z}_{k,i} \mapsto \mathbf{z}'_{k,i}$, $\Delta s_{k,i} \mapsto \Delta s'_{k,i}$, and $\boldsymbol{\nu}_{k,i} \mapsto \boldsymbol{\nu}'_{k,i}$ (the corresponding quantities in the second side set) to obtain the best-fit plane $\mathcal{P}'$ with normal $\boldsymbol{\nu}'$ passing through $\mathbf{c}'$.

7. Let $\mathcal{F}''_1, \dots, \mathcal{F}''_b$ be the non-boundary (internal) faces containing $\mathbf{x}^*$. Let $\boldsymbol{\nu}''_k$ and $\mathbf{c}''_k$ be the normal and centroid, respectively, of $\mathcal{F}''_k$.

8. For each $k = 1, \dots, b$, compute the intersection of $\mathcal{P}$, $\mathcal{P}'$, and $\mathcal{F}''_k$ by solving

$$\left[\ \boldsymbol{\nu}\ \middle|\ \boldsymbol{\nu}'\ \middle|\ \boldsymbol{\nu}''_k\ \right]^T \mathbf{x}^{(k)}_{\text{new}} = \begin{bmatrix} \langle \boldsymbol{\nu}, \mathbf{c} \rangle \\ \langle \boldsymbol{\nu}', \mathbf{c}' \rangle \\ \langle \boldsymbol{\nu}''_k, \mathbf{c}''_k \rangle \end{bmatrix} \tag{49}$$

9. Compute the centroid of $\left\{\mathbf{x}^{(k)}_{\text{new}}\right\}_{k=1}^{b}$. Call this point $\mathbf{x}^*_{\text{new}}$.

10. Set $\mathbf{a} \triangleq \mathbf{x}^*_{\text{new}} - \mathbf{x}^*$ and $\hat{s} \triangleq |\mathbf{a}|$. Compute the singular value decomposition

$$\hat{\mathbf{a}} = \hat{\mathbf{U}}\hat{\boldsymbol{\Sigma}}\hat{\mathbf{V}}^T$$

11. Set $\mathbf{p} \triangleq (\text{sgn}\langle \mathbf{a}, \hat{\mathbf{u}}_1 \rangle)\, \hat{\mathbf{u}}_1$, $\mathbf{q} \triangleq \hat{\mathbf{u}}_2$, $\mathbf{r} \triangleq \hat{\mathbf{u}}_3$, where $\hat{\mathbf{u}}_i$ is the i^{th} column of $\hat{\mathbf{U}}$.

12. Follow Section V.B with the direction and recession pairs $(\mathbf{p}, \hat{s})$, $(\mathbf{q}, 0)$, $(\mathbf{r}, 0)$, all enforced.

Step 2-5 are identical to those of Algorithm 4 for pure corner motion. Step 8 corresponds to Step 7 of the pure corner algorithm, except that we must repeat the solving process several times. In the pure corner case, a single plane encapsulates recession local to $\mathbf{x}^*$ within each side set, yielding three planes whose intersection point gives $\mathbf{x}^*_{\text{new}}$. For a double receder, we have two such planes from the receding sets. We complete the set of three necessary for a unique intersection point by locating internal boundary faces not otherwise involved in mesh motion–and it is possible to have several such faces, as shown below in Figure 32.

A hallmark of tangle-free mesh motion is its development of directions based on geometric clues provided by the mesh *interior*. By forcing nodes to remain on interior faces and edges, the boundary surfaces move in harmony with the internal structure of the mesh, avoiding the tangle inevitable of schemes that stick strictly with normal vectors. In 2D inlaid receders, relevant interior edges occur only within the same element as the receding edge so that the one-to-one correspondence of internal to boundary edges resolves any conflict easily. In 3D, however, such a clean and simple mapping does not exist and we must compute intersections of the two post-recession planes with each internal face, realized in Step 8.

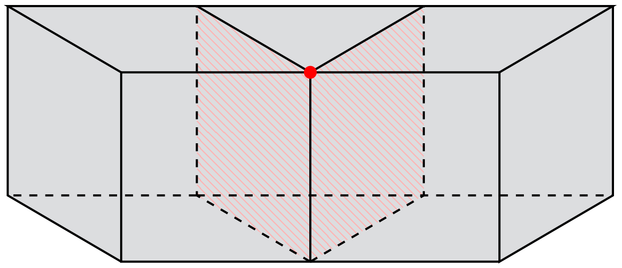


Figure 32. Multiple Internal Faces. The presence of tetrahedrons or other mixed elements makes it possible for a boundary node to be contained in multiple internal faces. Here, a triangular prism wedged between two hexahedra results in the red node having two internal faces. The two shared with elements involved in recession are shaded red.

The remainder of the algorithm is familiar: compute the centroid and movement vector $\hat{\mathbf{a}}$. The procedure concludes by determining an orthogonal set of directions coinciding with $\mathbf{a}$ in Steps 10 and 11 and enforcing three directions of motion. An illustration of the algorithm's operation is shown in Figure 33.

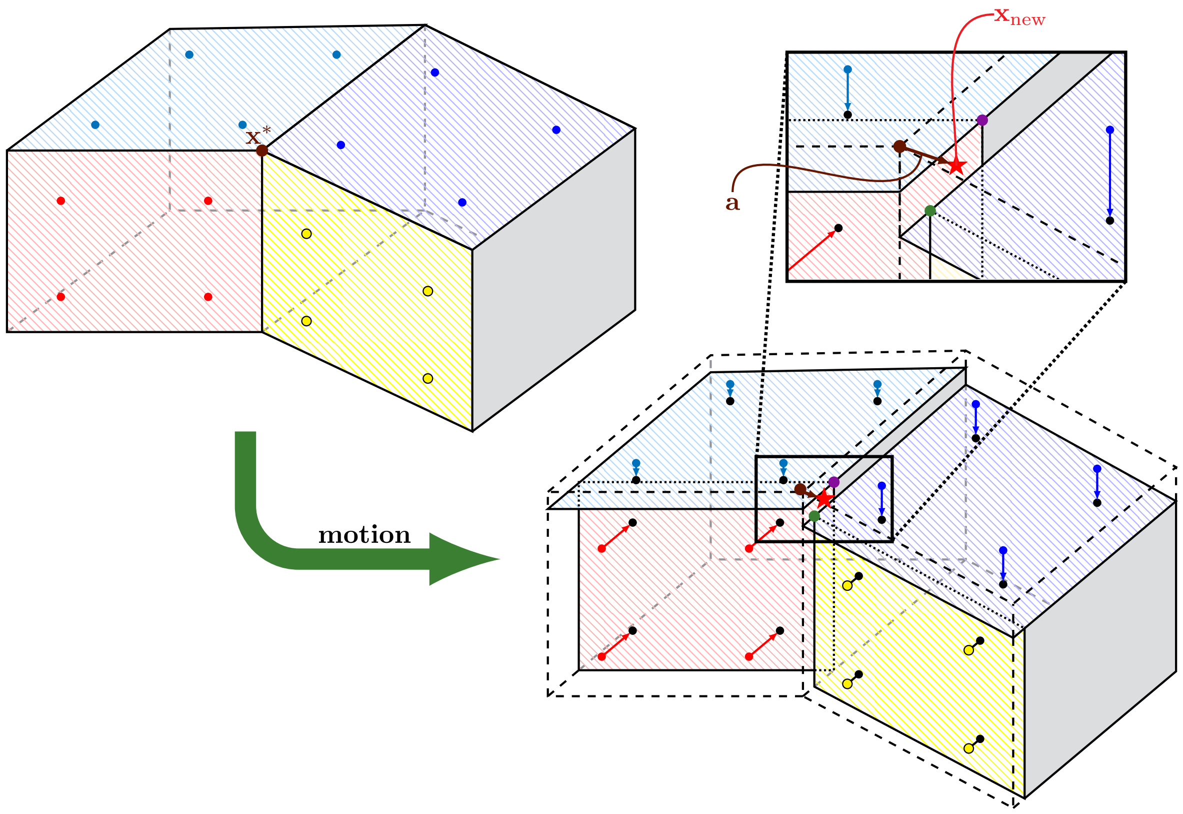


Figure 33. Illustration of Algorithm 5. ***(Upper left)*** **In this ideal case for Algorithm 5, $\mathbf{x}^*$, marked with brown circle, is an edge corner node. The two blue faces are in one side set; the red and yellow together reside in another.** ***(Lower Right)*** **The algorithm applies recession to each of the four involved faces. The diagram shows the new faces in color and the original ones in dashed outline. The original quadrature points $\mathbf{z}_{k,i}$ and normals $\boldsymbol{\nu}_{k,i}$ are shown in color; the black dots indicte the $\mathbf{z}_{k,i}$ post-recession and are used to compute the best fit-plane.** ***(Inset)*** **The green point marks the intersection of the planes containing the dark blue, yellow, and gray faces after recession; the purple point, of the light blue, red, and gray faces. The red star, the centroid of the green and purple points, is the new node location.**

4. Edge Corner Nodes: Receder/Slider

The procedure for resolving motion of edge corners between sliding and receding side sets has only one major difference from that of double receders: one of the side sets has no motion of its own. Nevertheless, we must still compute a single best-fit plane for the faces containing $\mathbf{x}^*$ in the sliding set because there is no well-defined normal at any node. Algorithm 6 below details the process. The first five steps are verbatim from Algorithm 5. Steps 6-9 are Steps 2-5 of the double receder algorithm specialized for sliding sets, in which $\Delta s \equiv 0$. From there, the algorithm concludes in a manner identical to that of Algorithm 5.

Algorithm 6 (Edge Corner Motion: Receder/Slider).

1. Let $\mathcal{F}_1, \ldots, \mathcal{F}_m$ be the boundary faces in the receding side set; $\mathcal{F}'_1, \ldots, \mathcal{F}'_{m'}$, in the sliding.
2. *(Receding side set.)* Let $\mathbf{x}_{k,1}, \ldots, \mathbf{x}_{k,q_k} \in \mathcal{F}_k$ be the quadrature points of the k^{th} receding face. Let $\boldsymbol{\nu}_{k,i}$ and $\Delta s_{k,i}$ be the normal and recession value, respectively, specified at $\mathbf{x}_{k,i}$.
3. Compute the centroid of the collection $\bigcup_{k=1}^m \left\{\mathbf{x}_{k,i} + \Delta s_{k,i}\boldsymbol{\nu}_{k,i}\right\}_{i=1}^{q_k}$:
$$\mathbf{c} \triangleq \left(\sum_{k=1}^m q_k\right)^{-1} \sum_{k=1}^m \sum_{i=1}^{q_k} \left[\mathbf{x}_{k,i} + \Delta s_{k,i}\boldsymbol{\nu}_{k,i}\right]$$
4. For each $k = 1, \ldots, m$, form the $3 \times q_k$ matrix
$$\mathbf{A}_k \triangleq \left[\begin{array}{c|c|c} \mathbf{x}_{k,1} + \Delta s_{k,1}\boldsymbol{\nu}_{k,1} - \mathbf{c} & \cdots & \mathbf{x}_{k,q_k} + \Delta s_{k,q_k}\boldsymbol{\nu}_{k,q_k} - \mathbf{c} \end{array}\right]$$
and glue all of these together to form the $3 \times Q$ matrix, where $Q \triangleq \prod_{k=1}^m q_k$,
$$\mathbf{A} = \left[\begin{array}{ccc} \mathbf{A}_1 & \cdots & \mathbf{A}_m \end{array}\right]$$
5. Compute the singular value decomposition of $\mathbf{A}$:
$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$$
where $\mathbf{U} \in \mathbb{R}^{3\times3}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{3\times Q}$, and $\mathbf{V} \in \mathbb{R}^{Q\times Q}$. Let $\mathcal{P}$ be the plane with normal vector $\boldsymbol{\nu} \triangleq \mathbf{u}_3$, the 3rd column of $\mathbf{U}$, passing through the point $\mathbf{c}$.
6. *(Sliding side set.)* Let $\mathbf{x}'_{k,1}, \ldots, \mathbf{x}'_{k,q'_k} \in \mathcal{F}'_k$ be the quadrature points of the k^{th} sliding face.
7. Compute the centroid of the collection $\bigcup_{k=1}^{m'} \left\{\mathbf{x}'_{k,i}\right\}_{i=1}^{q'_k}$:
$$\mathbf{c}' \triangleq \left(\sum_{k=1}^m q'_k\right)^{-1} \sum_{k=1}^{m'} \sum_{i=1}^{q'_k} \mathbf{x}'_{k,i}$$
8. Form the $3 \times Q'$ matrix, where $Q' \triangleq \prod_{k=1}^{m'} q'_k$,
$$\mathbf{A}' \triangleq \left[\begin{array}{c|c|c|c|c|c|c} \mathbf{x}'_{1,1} - \mathbf{c}' & \cdots & \mathbf{x}'_{1,q_1} - \mathbf{c}' & \cdots & \mathbf{x}'_{m',q'_{m'}} - \mathbf{c}' & \cdots & \mathbf{x}'_{m',q'_{m'}} - \mathbf{c}' \end{array}\right]$$
9. Compute the singular value decomposition of $\mathbf{A}'$:
$$\mathbf{A} = \mathbf{U}'\boldsymbol{\Sigma}'\mathbf{V}'^T$$
where $\mathbf{U}' \in \mathbb{R}^{3\times3}$, $\boldsymbol{\Sigma}' \in \mathbb{R}^{3\times Q'}$, and $\mathbf{V}' \in \mathbb{R}^{Q'\times Q'}$. Let $\mathcal{P}'$ be the plane with normal vector $\boldsymbol{\nu}' \triangleq \mathbf{u}'_3$, the 3rd column of $\mathbf{U}'$, passing through the point $\mathbf{c}'$.
10. Let $\mathcal{F}''_1, \ldots, \mathcal{F}''_b$ be the non-boundary (internal) faces containing $\mathbf{x}^*$. Let $\boldsymbol{\nu}''_k$ and $\mathbf{c}''_k$ be the normal and centroid, respectively, of $\mathcal{F}''_k$.

(continued).

11. For each $k = 1, \ldots, b$, compute the intersection of $\mathcal{P}$, $\mathcal{P}'$, and $\mathcal{F}''_k$ by solving

$$\left[\ \boldsymbol{\nu}\ \middle|\ \boldsymbol{\nu}'\ \middle|\ \boldsymbol{\nu}''_k\ \right]^T \mathbf{x}_{\text{new}}^{(k)} = \begin{bmatrix} \langle \boldsymbol{\nu}, \mathbf{c} \rangle \\ \langle \boldsymbol{\nu}', \mathbf{c}' \rangle \\ \langle \boldsymbol{\nu}''_k, \mathbf{c}''_k \rangle \end{bmatrix} \tag{50}$$

12. Compute the centroid of $\left\{ \mathbf{x}_{\text{new}}^{(k)} \right\}_{k=1}^{b}$. Call this point $\mathbf{x}^*_{\text{new}}$.
13. Set $\mathbf{a} \triangleq \mathbf{x}^*_{\text{new}} - \mathbf{x}^*$ and $\hat{s} \triangleq |\mathbf{a}|$. Compute the singular value decomposition

$$\hat{\mathbf{a}} = \hat{\mathbf{U}} \hat{\mathbf{\Sigma}} \hat{\mathbf{V}}^T$$

14. Set $\mathbf{p} \triangleq (\text{sgn}\langle \mathbf{a}, \hat{\mathbf{u}}_1 \rangle)\, \hat{\mathbf{u}}_1$, $\mathbf{q} \triangleq \hat{\mathbf{u}}_2$, $\mathbf{r} \triangleq \hat{\mathbf{u}}_3$, where $\hat{\mathbf{u}}_i$ is the i^{th} column of $\hat{\mathbf{U}}$.
15. Follow Section V.B with the direction and recession pairs $(\mathbf{p}, \hat{s})$, $(\mathbf{q}, 0)$, $(\mathbf{r}, 0)$, all enforced.

A figure, similar to Figure 33, below illustrates Algorithm 6 in action on a simple example.

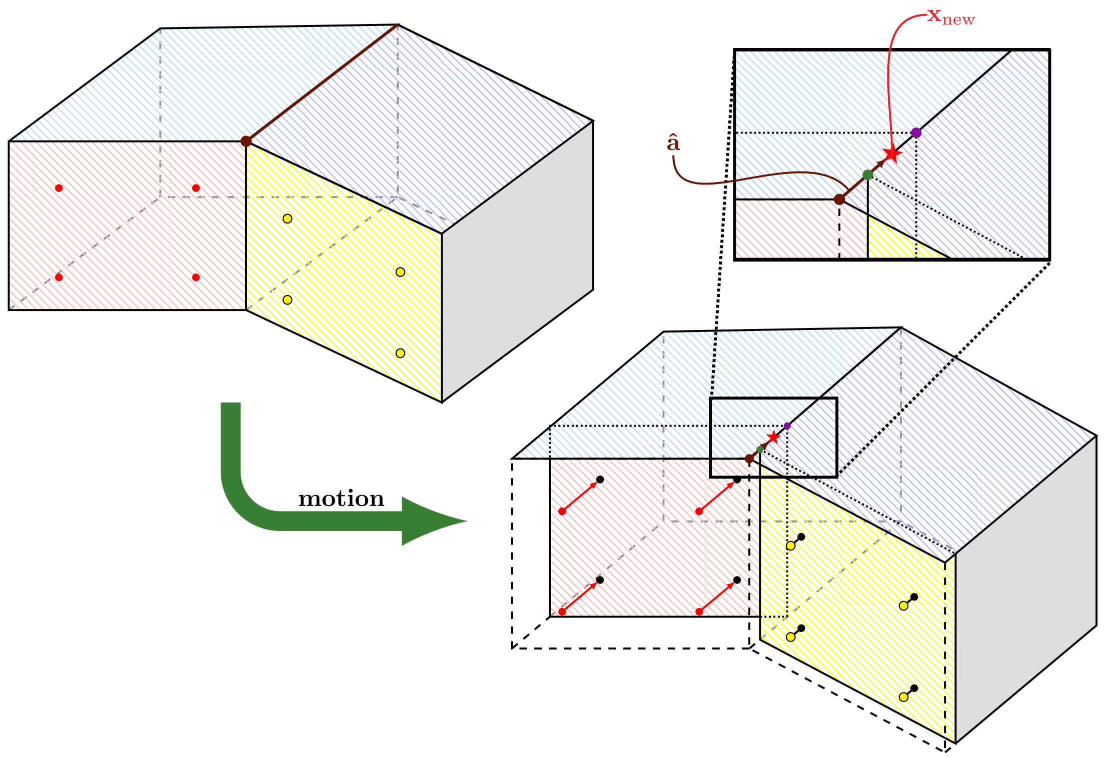


Figure 34. Illustration of Algorithm 6. ***(Upper left)*** **The starting scenario is as it was in Figure 33, except that the blue-colored faces are members of a sliding side set this time. Note the brown edge between the sliding faces.** ***(Lower right)*** **The red and yellow faces are post-recession. The blue faces have no movement of their own because they are sliding faces.** ***(Inset)*** **The green circle indicates the point of intersection of the light blue face, red face, and lone internal face (not colored); the purple circle, of the dark blue face, yellow face, and internal face. The red star is the centroid of these two points and is the new node location.**

In Figure 34, the final node location lies on the edge between the sliding faces (marked in blue in the upper left subdiagram). This will always occur when there is only one internal face. In fact, finding the intersection of faces is equivalent to identifying the edges in the sliding set, computing the intersection with receding faces, and extracting a best-fit vector of the results. An earlier version of the algorithm did precisely this. However, repetition of the logic of Algorithm 5 greatly simplifies implementation, allowing a single code path to cover both slider/receder and receder/receder cases after detection of a single receding side set.

5. Edge Corner Nodes: Slider/Slider

The presence of at least one receding side set in each of the two previous edge corner cases provides a source of movement. In this manner, the various intersection points of post-recession planes each make a non-null contribution to the final node location. A node sitting on the edge between two sliding side sets has no inherent source of motion; the task is to determine in which directions it is *allowed* to move.

One approach would be to determine a "normal" (more accurately, forbidden direction) for each of the two sliding side sets and then determine a vector $\mathbf{u}$ which completes the basis for $\mathbb{R}^3$ and allow the system dynamics to determine the motion in the direction of $\mathbf{u}$. Except for locally flat side sets—each edge corner node is contained in elements that have approximately parallel normals—this approach will lead to significant shape deformation as imprecision in approximating normals does not quash motion as required.

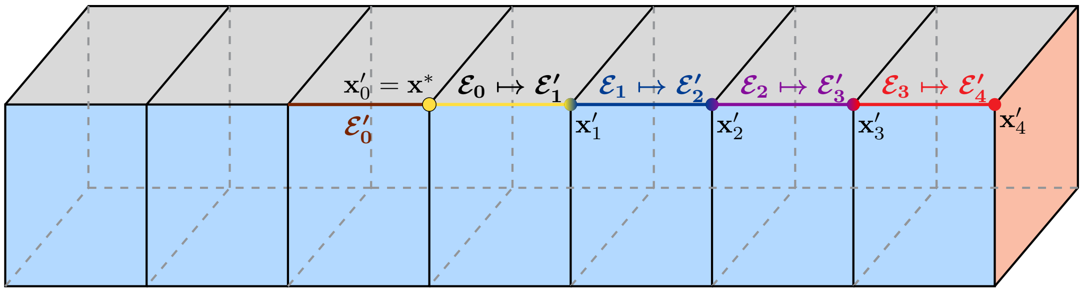
Suppression of motion in the normal direction is merely the mathematical interpretation of a sliding side set; more intrinsically, a sliding node should remain in its current plane (or planes) of existence, moving only in response to motion from sources. Consequently, rather than determine forbidden directions, it equivalent to determine the allowable ones. For the slider/slider edge corner, we expect that each side set should contribute one forbidden direction; juxtaposing the two means we should seek one permissible direction.

Geometry provides an excellent set of candidates for the allowable direction of motion: the edges between the side sets (i.e., contained in both). No matter how many elements an edge corner node $\mathbf{x}^*$ is contained in, there are only two such edge segments on which it can be found. This greatly simplifies matters—no best-fit planes to calculate or no intersections to find. It does, however, raise a dilemma: which of the two to choose? It is natural that nodes retreat away from heat sources; consequently, we execute this algorithm:

Algorithm 7 (Slider/Slider Edge Selection).

1. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be the side sets and $\mathcal{E}_1, \mathcal{E}_2 \subset \mathcal{S}_1, \mathcal{S}_2$ be the edges that contain $\mathbf{x}^*$.
2. Set $\mathcal{E} = \mathcal{E}_1$, $\mathcal{E}' = \mathcal{E}_2$, and $\mathbf{x}' = \mathbf{x}^*$. If $\mathcal{E}_1 \parallel \mathcal{E}_2$ (as vectors), set $I = 1$ and go to Step 6.
3. $\mathbf{x}'$ is one node on $\mathcal{E}$. Let $\mathbf{x}''$ be the other. If $\mathbf{x}'' = \mathbf{x}^*$, set $I = 1$ and go to Step 6.
4. Let $\mathcal{S}''_1, \dots, \mathcal{S}''_n$ be the side sets of which $\mathbf{x}''$ is a member. If $n = 3$ and any $\mathcal{S}''_i$ is receding, set $I = 2$; otherwise, set $I = 1$. If I was set, procede to Step 6.
5. Locate the edges $\hat{\mathcal{E}}$ and $\hat{\mathcal{E}}'$ containing $\mathbf{x}''$ with $\mathbf{x}' \notin \hat{\mathcal{E}}$ (i.e., $\mathbf{x}' \in \hat{\mathcal{E}}'$). Update $\mathcal{E} \leftarrow \hat{\mathcal{E}}$, $\mathcal{E}' \leftarrow \hat{\mathcal{E}}'$, and $\mathbf{x}' \leftarrow \mathbf{x}''$. Return to Step 3.
6. The selected edge is $\mathcal{E}_I$.

Algorithm 7 marches along the topological edge between the side sets containing $\mathbf{x}^*$, starting in the direction of $\mathcal{E}_1$ until it either loops back to $\mathbf{x}^*$ (a bad situation) or comes to a pure corner. In the event of the second, if one of the side sets comprising that corner is receding, then $\mathcal{E}_2$ is selected as the edge along which to move. The process is illustrated in Figure 35. In the figure, subscripts denote the quantity for an iteration. Because it is unnecessary to store this information from step to step, Algorithm 7 overwrites what is no longer needed; the diagram merely notes what belongs to each for the purposes of demonstration.



$\mathbf{x}'_0 = \mathbf{x}^*$
$\mathcal{E}_0 \mapsto \mathcal{E}'_1$
$\mathcal{E}_1 \mapsto \mathcal{E}'_2$
$\mathcal{E}_2 \mapsto \mathcal{E}'_3$
$\mathcal{E}_3 \mapsto \mathcal{E}'_4$
$\mathcal{E}'_0$
$\mathbf{x}'_1$
$\mathbf{x}'_2$
$\mathbf{x}'_3$
$\mathbf{x}'_4$

Figure 35. Illustration of Algorithm 7. We begin at the yellow node $\mathbf{x}^*$. We identify the two edges $\mathcal{E}_0$ and $\mathcal{E}'_0$ (these names are assigned randomly). The algorithm proceeds along the topological edge to $\mathbf{x}'_1$ to $\mathbf{x}'_2$ to $\mathbf{x}'_3$. At $\mathbf{x}'_4$, it discovers three containing side sets. If the red (rightmost) face is a receding side set, $\mathcal{E}'_0$ will be the selected edge. Otherwise, it will be $\mathcal{E}_0$.

Algorithm 7 handles most of the heavy-lifting for this case. With it in place, the motion algorithm is simple:

Algorithm 8 (Edge Corner Motion: Slider/Slider).

1. Execute Algorithm 7. Let $\mathbf{a}$ be the unit vector parallel to $\mathcal{E}_I$ and pointing away from $\mathbf{x}^*$.
2. Compute the singular value decomposition
$$\hat{\mathbf{a}} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T$$
3. Set $\mathbf{r} \triangleq (\text{sgn}\langle \mathbf{a}, \hat{\mathbf{u}}_1\rangle)\, \hat{\mathbf{u}}_1$, $\mathbf{p} \triangleq \hat{\mathbf{u}}_2$, $\mathbf{q} \triangleq \hat{\mathbf{u}}_3$, where $\hat{\mathbf{u}}_i$ is the i^{th} column of $\hat{\mathbf{U}}$.
4. Follow Section V.B with the direction and recession pairs $(\mathbf{p}, 0)$, $(\mathbf{q}, 0)$, and $(\mathbf{r}, \cdot)$, only the first two of which should be enforced.

The cases up to this point have all featured triple enforcement. Reduction to double enforcement of the sliding/sliding edge corner allows the system dynamics to determine the sliding of $\mathbf{x}^*$ along the detemermined direction, as we do not know this *a priori*. Figure 36 below illustrates Algorithm 8 simply.

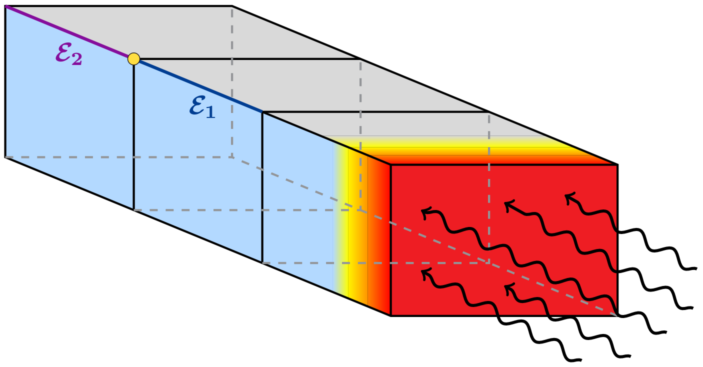


Figure 36. Illustration of Algorithm 8. The blue and gray-colored faces are each members of different sliding side sets. The yellow circle marks an edge corner node $\mathbf{x}^*$. The two edges between the sliding side sets, $\mathcal{E}_1$ and $\mathcal{E}_2$, are marked in dark blue and purple, respectively. Heating is applied to the front face, indicated in red. Because $\mathcal{E}_1$ is closer to this source, we choose $\mathcal{E}_2$ to be the direction of allowed motion.

6. Inlaid Nodes

Because an inlaid node is one for which all containing faces share a single commmon side set, there are only three subcases. The first of these, the stationary one, we have encountered before and resolve it in the same manner: set the directions to coordinate axes and all motion values to zero. We thus focus on the receding and sliding subcases.

7. Inlaid Nodes: Receders

By now, the general *modus operandi* of processing receding faces should be clear: apply recession at all quadrature points of faces containing $\mathbf{x}^*$, compute best-fit planes for some collection of these points, and then determine the intersection of these best-fit planes with some geometric feature within the interior of the mesh. The story is no different here, except that instead of using an internal face, we instead use internal *edges* containing $\mathbf{x}^*$. For the various corners, such an edge may not exist and, in fact, the absense is more desirable for the mesh motion algorithm. For inlaid nodes, however, at least one such edge must exist. Algorithm 9 details the process, many steps of which bear similiarity to those of the previous algorithms.

Algorithm 9 (Inlaid Motion: Receder).

1. Let $\mathcal{F}_1, \ldots, \mathcal{F}_m$ be the boundary faces containing $\mathbf{x}^*$.
2. Let $\mathbf{z}_{k,1}, \ldots, \mathbf{z}_{k,q_k} \in \mathcal{F}_k$ be the quadrature points of the k^{th} face. Let $\boldsymbol{\nu}_{k,i}$ and $\Delta s_{k,i}$ be the normal and recession value, respectively, specified at the quadrature point $\mathbf{z}_{k,i}$.
3. Compute the centroid of the points $\left\{\mathbf{z}_{k,i} + \Delta s_{k,i}\boldsymbol{\nu}_{k,i}\right\}_{i=1}^{q_k}$:

$$\mathbf{c}_k \triangleq \frac{1}{q_k}\sum_{i=1}^{q_k}\left[\mathbf{z}_{k,i} + \Delta s_{k,i}\boldsymbol{\nu}_{k,i}\right]$$

4. Form the $3 \times q_k$ matrix

$$\mathbf{A}_k \triangleq \left[\begin{array}{c|c|c} \mathbf{z}_{k,1} + \Delta s_{k,1}\boldsymbol{\nu}_{k,1} - \mathbf{c}_k & \cdots & \mathbf{z}_{k,q_k} + \Delta s_{k,q_k}\boldsymbol{\nu}_{k,q_k} - \mathbf{c}_k \end{array}\right]$$

5. Compute the singular value decomposition of $\mathbf{A}_k$:

$$\mathbf{A}_k = \mathbf{U}_k \boldsymbol{\Sigma}_k \mathbf{V}_k^T$$

   where $\mathbf{U}_k \in \mathbb{R}^{3\times 3}$, $\boldsymbol{\Sigma}_k \in \mathbb{R}^{3\times q_k}$, and $\mathbf{V}_k \in \mathbb{R}^{q_k \times q_k}$. Let $\hat{\boldsymbol{\nu}}_k \triangleq \mathbf{u}_3$, the 3rd column of $\mathbf{U}_k$.
6. Repeat Steps 2-5 for each $k \in \{1, \ldots, m\}$ to obtain $\mathbf{c}_k$ and $\hat{\boldsymbol{\nu}}_k$ for each face $\mathcal{F}_k$.
7. Locate each edge containing $\mathbf{x}^*$ that is not contained on $\mathcal{F}_k$. This will yield the collection $\left\{\bar{\mathbf{e}}_{k,i}\right\}_{i=1}^{n_k}$, each of which should be first normalized.
8. For each $i = 1, \ldots, n_k$, compute the intersection of the line $\ell_i(t) = \mathbf{x}^* + t\,\bar{\mathbf{e}}_{k,i}$ with the plane with normal $\hat{\boldsymbol{\nu}}_k$ passing through the point $\mathbf{c}_k$ (k^{th} best fit plane). First, compute

$$\alpha_i \triangleq \frac{\left\langle \mathbf{c}_k - \mathbf{x}^*, \hat{\boldsymbol{\nu}}_k \right\rangle}{\left\langle \bar{\mathbf{e}}_{k,i}, \hat{\boldsymbol{\nu}}_k \right\rangle}$$

   The intersection point is then given by

$$\mathbf{y}_{k,i} \triangleq \mathbf{x}^* + \alpha_i \bar{\mathbf{e}}_{k,i}$$

9. Repeat Steps 7 and 8 for each $k \in \{1, \ldots, m\}$.

(continued).

10. Compute the centroid of the collection $\bigcup_{k=1}^{m} \{\mathbf{y}_{k,i}\}_{i=1}^{n_k}$:

$$\mathbf{x}^*_{\text{new}} \triangleq \left(\sum_{k=1}^{m} n_k\right)^{-1} \sum_{k=1}^{m}\sum_{i=1}^{n_k} \mathbf{y}_{k,i}$$

11. Set $\mathbf{a} \triangleq \mathbf{x}^*_{\text{new}} - \mathbf{x}^*$ and $\hat{s} \triangleq |\mathbf{a}|$. Compute the singular value decomposition

$$\mathbf{a} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T$$

12. Set $\mathbf{p} \triangleq (\operatorname{sgn}\langle \mathbf{a}, \hat{\mathbf{u}}_1\rangle)\, \hat{\mathbf{u}}_1$, $\mathbf{q} \triangleq \hat{\mathbf{u}}_2$, $\mathbf{r} \triangleq \hat{\mathbf{u}}_3$, where $\hat{\mathbf{u}}_i$ is the i^{th} column of $\hat{\mathbf{U}}$.
13. Follow Section V.B with the direction and recession pairs $(\mathbf{p}, \hat{s})$, $(\mathbf{q}, 0)$, $(\mathbf{r}, 0)$.

The steps, with exception of Step 8, which recalls Algorithm 2 for 2D internal recession, should be familiar from the previous 3D motion algorithms. Figure 37 below diagrams the process.

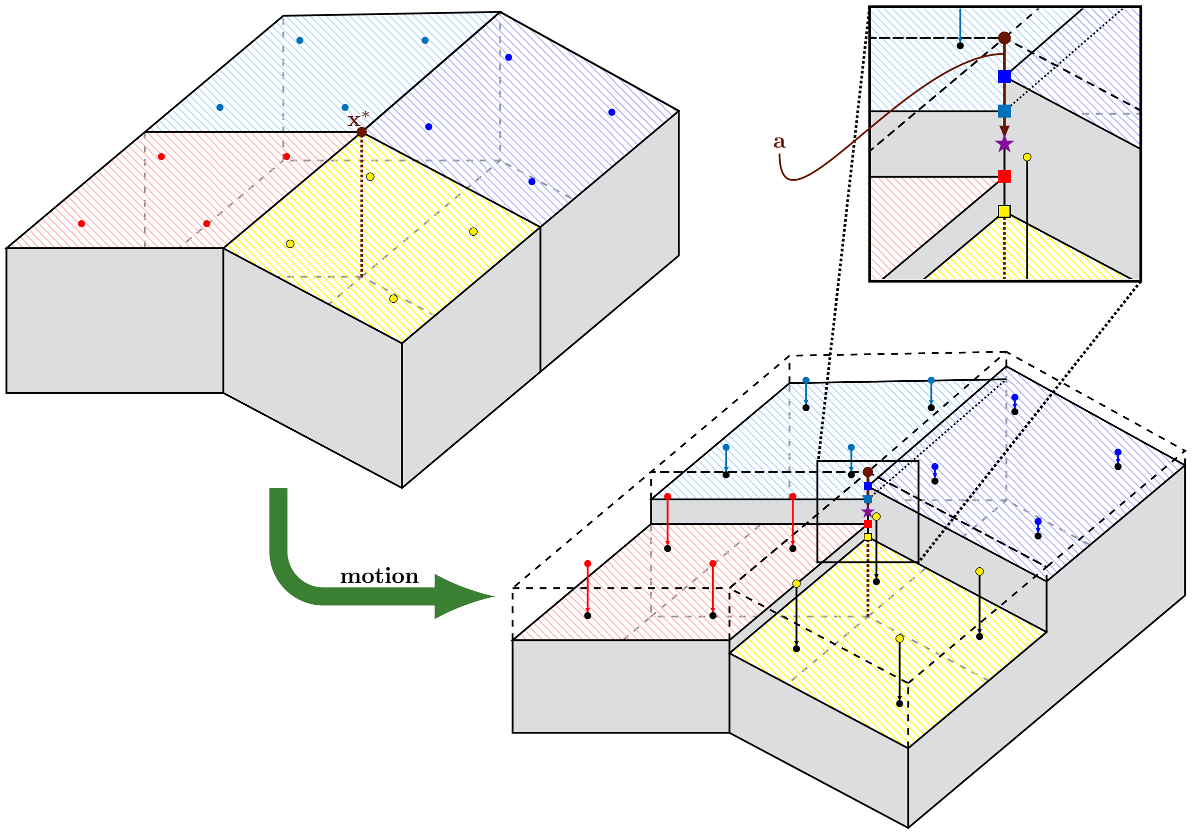


Figure 37. Illustration of Algorithm 9. *(Upper left)* $\mathbf{x}^*$ is marked with a brown circle; the lone internal edge is the brown dashed line. *(Lower right)* In contrast to previous algorithms, which computed a single best-fit plane for each side set, Algorithm 9 computes one for each face containing $\mathbf{x}^*$. *(Inset)* The intersection of each post-recession plane with the internal edge is marked with squares matching the color of the corresponding face. The purple star, the new node location, marks the centroid of these points.

Of the times we've encountered sliding side sets up to this point, only two of them have not included a receding side set: the pure corner with three sliders and the slider/slider edge corner. Algorithm 4 resolves the triple slider without special consideration. Because the normals of the faces meeting at $\mathbf{x}^*$ span $\mathbb{R}^3$ per Lemma 3, and zero motion conditions are imposed in each of these normals, a triple slider is equivalent to a stationary point. In the slider/slider edge corner, rather than attempt to determine a set of forbidden directions from the sliding surfaces, we chose an edge along which motion is permitted.

In the introduction, we defined a sliding node to be one whose motion is quashed in the surface normal direction, or, equivalently, one that moves only in its current local plane of existence. The fundamental goal of a sliding condition is to preserve the general shape of surface in the wake of downstream deformations; for small displacements, the tangent plane, the orthogonal complement of the normal vector, is an excellent approximation of the surface and hence restricting motion to it accomplishes that objective quite well.

With the exception of the pure corner, none of the scenarios analyzed so far have made any use of surface normals for sliding nodes, instead using geometry to craft directions. Since the normal at a node is an elusive quantity, this has been by design. In this manner, we have been able to remain true to the purpose of sliding conditions—surface preservation—without making approximations. One may hope that a similar approach for inlaid sliders, perhaps using edges as we did for inlaid receders, could be effective, but this faith is ultimately misplaced. None of the the internal mesh structure has any relation to the above-lying surface; in all but the most basic of geometries, taking clues from it will only lead us astray.

It is therefore with a sigh of agony that we accept that we must find a way to obtain the normal vector, as our ability to maintain the surface is only as good as this approximation. If a surface $\mathcal{S}$ can be defined implicitly by the equation $S(\mathbf{x}) = 0$, then the gradient ∇S gives the surface normal. If we can find an analytic interpolant $\hat{S}$ to S at the points $\mathbf{x}_1, \ldots, \mathbf{x}_m$, then $\nabla \hat{S}$ is as good an approximation to the true normal as we can hope to obtain. No matter which type of interpolant we choose, attempting to solve $\hat{S}(\mathbf{x}_i) = 0$ leads to $\hat{S} \equiv 0$. To obtain a nontrivial solution, we must add some some nonzero values to the collection.

For δ sufficiently small, the perturbed surface $\tilde{S}(\mathbf{x}) = S(\mathbf{x}) + \delta$ is a reasonable approximation of the original, complete with the same normal. By selecting a subset of points $\{\mathbf{x}_i\}_{i\in I}$ and requiring $S(\mathbf{x}_i) = \delta$ for $i \in I$, we can obtain a nonzero interpolant. Because we know the normal at quadrature points, we can improve on this; rather than replace points and values from the original collection, we can agument it:

$$\begin{aligned} S(\mathbf{x} + \delta\boldsymbol{\nu}) &= \delta \\ S(\mathbf{x} - \delta\boldsymbol{\nu}) &= -\delta \end{aligned}$$

We add points both above and below the surface per recommendation of [5]. The interpolant of choice is the radial basis function [3, 4], or RBF for short. An RBF is exactly as it sounds: it is a function whose value at a point depends only on the distance of that point from some origin $\mathbf{x}$, or $\zeta(\mathbf{y}) = \zeta\big(d(\mathbf{x}, \mathbf{y})\big)$, where $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^+$ is a distance metric. Here, we use the usual Euclidean distance $d(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}|$. There are many choices for ζ; commonly, Gaussian functions of the form $\zeta(r) = e^{-\varepsilon r^2}$ are used. For our application, we employ the RBF $\zeta_{n,k}$ developed by Wendland [18], which are compactly supported on $\mathbb{R}^n$ and are of class $C^{2k}(\mathbb{R})$, perfect for our purposes. On $\mathbb{R}^3$, $k = 1$ is sufficient; the function has the form

$$\zeta(r) \triangleq \zeta_{3,1}(r) = \begin{cases} (1+r)^4(4r+1) & r \le 1 \\ 0 & r > 1 \end{cases}$$

With the functions set, we populate the quadature points $\mathbf{z}_1, \ldots, \mathbf{z}_n$ across all faces on the surface. The $3n$ interpolation nodes are then

$$\mathbf{x}_i = \begin{cases} \mathbf{z}_i & 1 \le i \le n \\ \mathbf{z}_{i-n} + \delta\boldsymbol{\nu}(\mathbf{z}_{i-n}) & n+1 \le i \le 2n \\ \mathbf{z}_{i-2n} - \delta\boldsymbol{\nu}(\mathbf{z}_{i-2n}) & 2n+1 \le i \le 3n \end{cases}$$

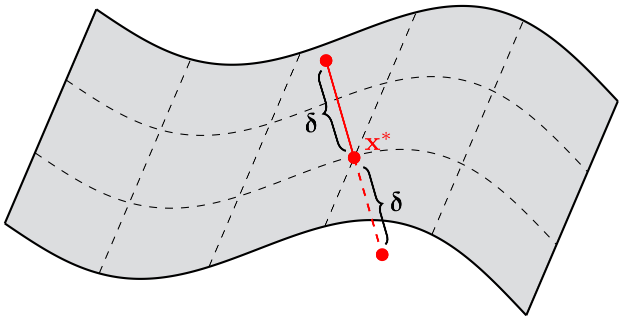


Figure 38. Points for Surface Interpolation. If the true surface normal $\boldsymbol{\nu}$ is known at a point $\mathbf{x}$, then the pairs $(\mathbf{x}, 0)$, $(\mathbf{x} + \delta\boldsymbol{\nu}, \delta)$, and $(\mathbf{x} - \delta\boldsymbol{\nu}, -\delta)$ can be used for the interpolation problem.

We seek an interpolant based on the RBF network:

$$\hat{S}(\mathbf{x}) = \sum_{i=1}^{3n} w_i \zeta(\mathbf{x} - \mathbf{x}_i) \tag{51}$$

The interpolation problem can be cast into matrix form:

$$\mathbf{A}\mathbf{w} = \mathbf{b} \tag{52}$$

where the matrix $\mathbf{A} \in \mathbb{R}^{3n \times 3n}$ and vector $\mathbf{b} \in \mathbb{R}^{3n}$ have entries

$$A_{ij} = \zeta(\mathbf{x}_i - \mathbf{x}_j) \qquad\qquad b_i = \begin{cases} \delta & 1 \le i \le n \\ 0 & n+1 \le i \le 2n \\ -\delta & 2n+1 \le i \le 3n \end{cases}$$

To ensure that ζ centered at $\mathbf{x}_i$ is nonzero across the entire domain and thus interpolates each point, we scale the input argument. Let

$$\mathbf{x}_{\max} \triangleq \left[\max_{\mathbf{x}\in\mathcal{S}} x^{(0)}, \max_{\mathbf{x}\in\mathcal{S}} x^{(1)}, \max_{\mathbf{x}\in\mathcal{S}} x^{(2)}\right] \qquad\qquad \mathbf{x}_{\min} \triangleq \left[\min_{\mathbf{x}\in\mathcal{S}} x^{(0)}, \min_{\mathbf{x}\in\mathcal{S}} x^{(1)}, \min_{\mathbf{x}\in\mathcal{S}} x^{(2)}\right]$$

so that $\mathbf{x}_{\max}$ and $\mathbf{x}_{\min}$ represent the "upper right back" and "lower left front" coordinates of the smallest bounding box for $\mathcal{S}$. Define $r_d \triangleq |\mathbf{x}_{\max} - \mathbf{x}_{\min}|$ and set $\zeta(r) \hookleftarrow \zeta(r/r_d)$ so that ζ is supported on $[0, r_d]$ instead of $[0, 1]$. With this scaling, $A_{ij} > 0$ and $\mathbf{A}$ becomes a dense matrix.

As the number of points n on the surface increases, Eq. (52) becomes increasingly difficult to solve. To remedy this issue, we make use of a novel preconditioning technique designed to make iterative solvers such as GMRES feasible, detailed for RBF in [20]. The technique is called the restricted additive Schwarz method (RASM) and involves decomposing the physical domain into a number of disjoint and overlapping subdomains. An example on a small, two-dimensional domain is shown below in Figure 39.

For each overlapping domain Ω_i and disjoint domain $\tilde{\Omega}_i$, the restriction operators $\mathbf{R}_i \in \mathbb{R}^{|\Omega_i| \times 3n}$ and $\tilde{\mathbf{R}}_i \in \mathbb{R}^{|\tilde{\Omega}_i| \times 3n}$, respectively, take a vector on the whole domain and extract only the portion contained in Ω_i or $\tilde{\Omega}_i$. The transpose of these matrices are the projection operators and take the subdomains to the whole domain. By extracting rows and columns of $\mathbf{A}$ corresponding to the nodes in the overlapping domains, we form the submatrices $\mathbf{A}_i$. From there, the RASM preconditioner is then given by

$$\mathbf{P}^{-1} = \sum_i \tilde{\mathbf{R}}_i^T \mathbf{A}_i^{-1} \mathbf{R}_i$$

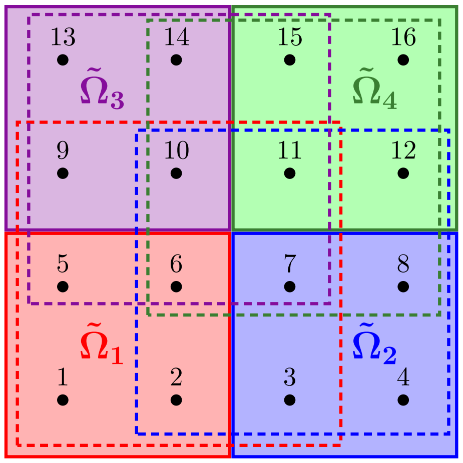


Figure 39. Simple Example of RASM Domain Decomposition. The disjoint domains $\tilde{\Omega}_1$ through $\tilde{\Omega}_4$ are outlined in solid line and shaded in solid color. Each contains four points. The corresponding overlapping domains Ω_i are depicted in dashed line of the same color. Each contains nine points.

Algorithms for RASM have been robustly implemented in freely-available linear algebra software packages such as `PETSc`. `PETSc` can handle the decomposition as well as construction of the preconditioner.

Once the weights w_i have been determined by solving Eq. (52) with help from RASM, we return to Eq. (51) and differentiate to obtain the needed normal vector:

$$\boldsymbol{\nu}(\mathbf{x}) = \nabla \hat{S} = \sum_{i=1}^{3n} w_k \frac{\partial \zeta}{\partial r}(\mathbf{x} - \mathbf{x}_i) \cdot \frac{\mathbf{x} - \mathbf{x}_i}{|\mathbf{x} - \mathbf{x}_i|} \tag{53}$$

Obtaining the normal vector is the most difficult step in processing inlaid sliders. With that process outlined, the algorithm is the shortest and simplest of all of the six cases we've examined:

Algorithm 10 (Inlaid Motion: Slider).

1. Let $\mathcal{S}$ be a sliding side set. Enumerate the quadrature points from all faces contained in $\mathcal{S}$.
2. Form and solve Eq. (52) to obtain the weights for the RBF interpolation of $\mathcal{S}$.
3. Set $\mathbf{a} = \boldsymbol{\nu}(\mathbf{x}^*)$ from Eq. (51). Compute the singular value decomposition

$$\hat{\mathbf{a}} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T$$

4. Set $\mathbf{p} \triangleq (\operatorname{sgn}\langle \mathbf{a}, \hat{\mathbf{u}}_1 \rangle)\, \hat{\mathbf{u}}_1$, $\mathbf{q} \triangleq \hat{\mathbf{u}}_2$, $\mathbf{r} \triangleq \hat{\mathbf{u}}_3$, where $\hat{\mathbf{u}}_i$ is the i^{th} column of $\hat{\mathbf{U}}$.
5. Follow Section V.B with the direction and recession pairs $(\mathbf{p}, 0)$, $(\mathbf{q}, \cdot)$, $(\mathbf{r}, \cdot)$, with only the first of these enforced.

The first two steps of Algorithm 10, which are quite expensive, should be viewed as preprocesing step and should be executed prior to looping over nodes.

D. Demonstration

1. Wedge with Constant Heat Flux

The first demonstration problem is a three-dimensional wedge, a slice of the shoulder of an iso-q:

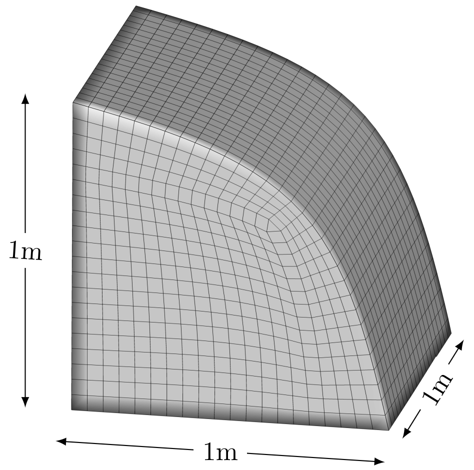
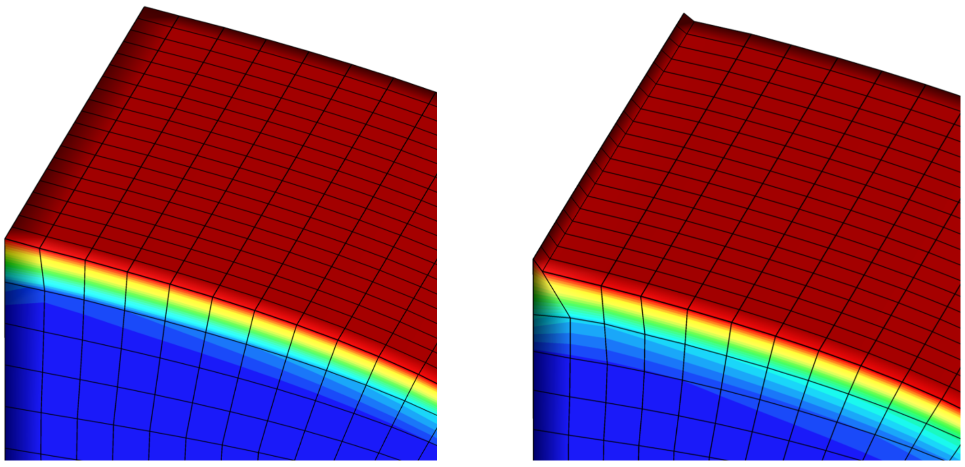


Figure 40. 3D Wedge Geometry. This is merely a slice from the shoulder of an iso-q, not entirely one. The curved portion of a true iso-q (cf. Figure 15) bends sharply enough that it meets the flat back and bottom nearly orthogonally. This is not the case here, as the curved portion does not flatten out as it meets the other sides. Furthermore, the back edge slopes upward slightly with a 5% grade.

We place a simple constant heat-flux boundary condition with $\dot{q} = 10^7\ {}^{\mathrm{W}}\!/_{\mathrm{m}^2}$ on the curved portion of the figure, which is set as a *receding* side set. The other five topological sides are each declared as sliding sets. The solid is a melter with the same properties as before, except with a slightly lower melt temperature:

Solid density	ρ	8960 $^{\mathrm{kg}}\!/_{\mathrm{m}^3}$	Melt temperature	T_{melt}	700 K
Thermal conductivity	k	394 $^{\mathrm{W}}\!/_{\mathrm{m\text{-}K}}$	Latent heat of fusion	ΔH_f^0	$2.05 \times 10^5\ {}^{\mathrm{J}}\!/_{\mathrm{kg}}$
Specific heat	c	383 $^{\mathrm{J}}\!/_{\mathrm{kg\text{-}K}}$			



(a) $t \approx 12$ **(b) $t \approx 40$**

Figure 41. Trouble for Simple Scheme. Initially, recession is satisfactory; in (a), after 12s, there are no indications of coming issues. As the simulation progresses, the behavior at flat upper and lower edges of the curved portion is incorrect. Fails occurs shortly after (b).

The simple normal scheme, which takes the direction of motion to be the normal without exception, is not capable of running this problem, although the point of default is not what might be expected. On the 2D iso-q, we saw tangling on the curved portion after a long simulation run. Simple normal doesn't run long enough for this to become an issue; instead, the non-orthgonality of the junction between the curved portion and the flat back and bottom fail to slide correctly and triggers the failure after about 42s.

The simulation used adaptive time stepping with $(\Delta t)_{\min} = 0.01$ and $(\Delta t)_{\max} = 0.05$. The initial time is $t_0 = 0$ and completed at the requested final time $t_f = 200$. Figure 17 below shows tangle-free's `CHAR` results.

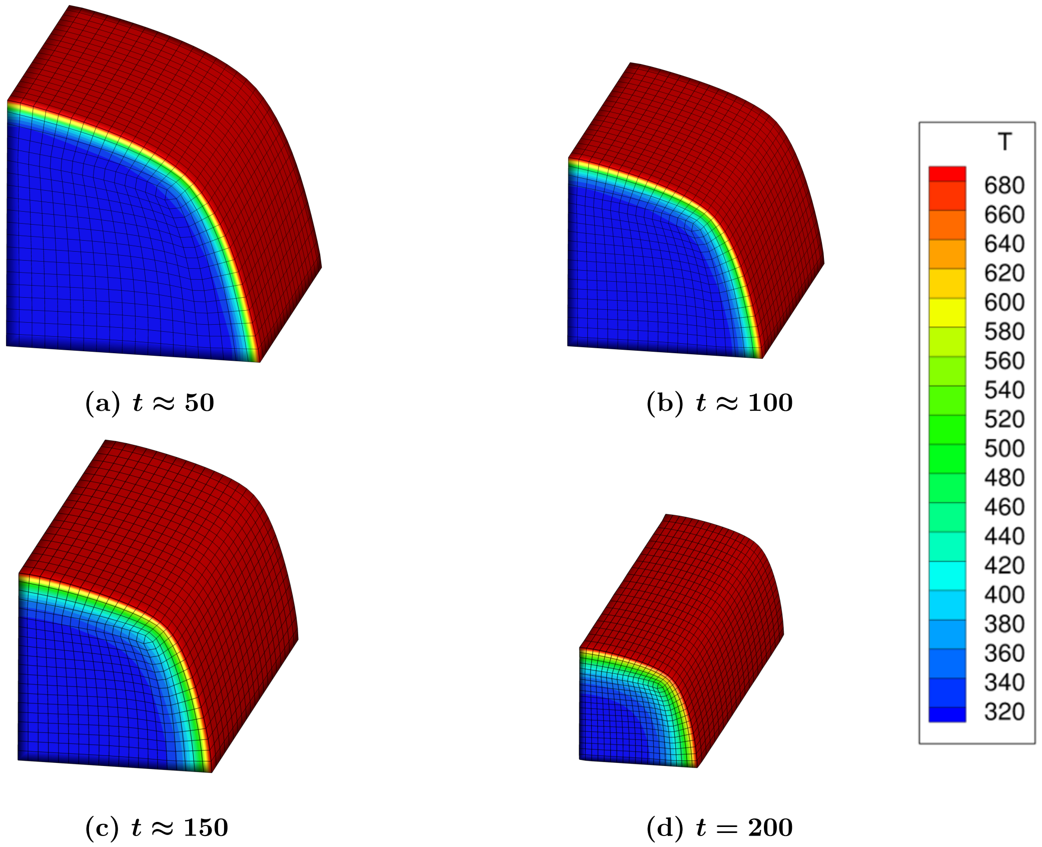


Figure 42. Wedge `CHAR` Simulation Results. All figures shown have the same view settings and scaling. Recession proceeds smoothly along the curved manifold without tangle.

The tangle-free scheme lives up to its name by handling the curve of this example flawlessly. Curiously, the simple scheme fails before it can tangle, which we would expect to occur after significant recession (cf. the 2D iso-q). Either way, it is another tally mark in the victory column of the tangle-free algorithm.

2. Parallelepiped

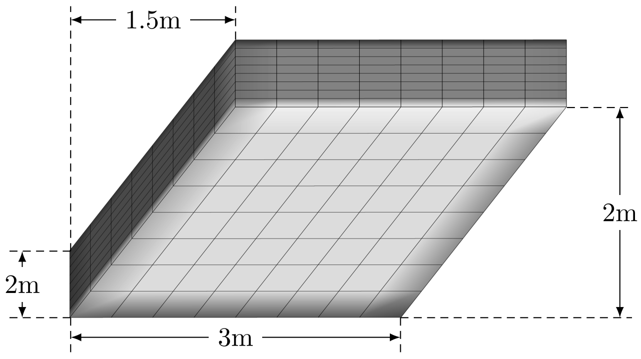
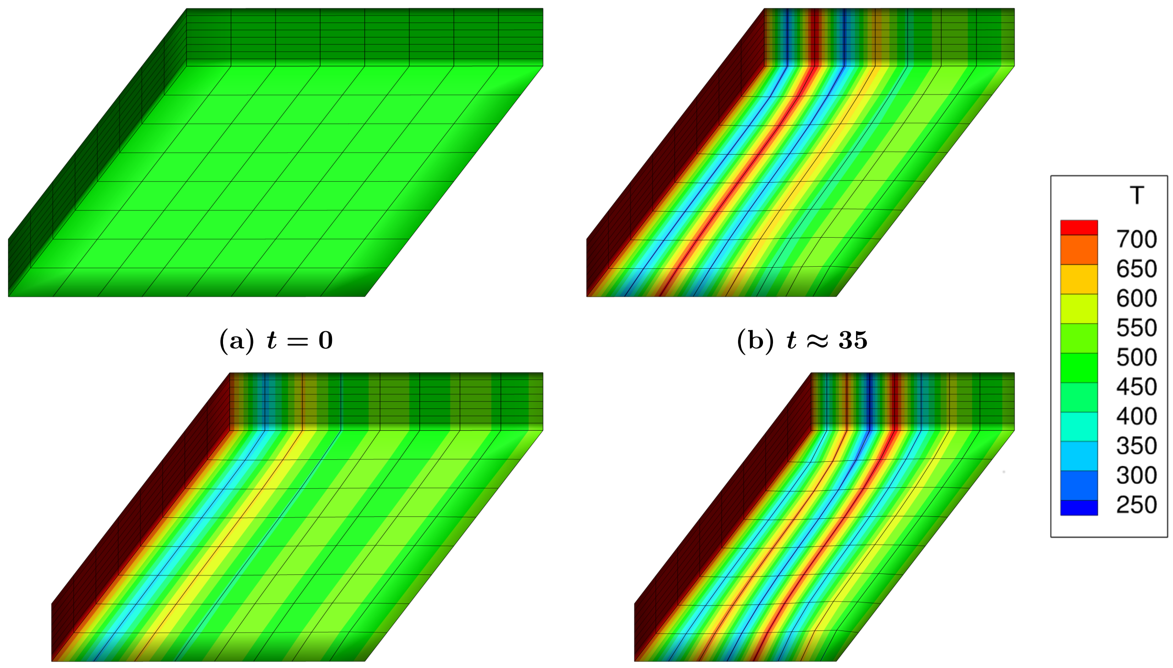


Figure 43. Parallelepiped Geometry. An axis-aligned parallelepiped services this example.

The second demonstration problem is an innoculous-looking parallelepiped, depicted above in Figure 43. The material properties are the same as in the previous example. The time steps were again adaptive with $(\Delta t)_{\min} = 0.05$ and $(\Delta t)_{\max} = 0.10$; the simulation was run from $t_0 = 0$ until $t_f = 50$ and without failure. A simple constant heat-flux boundary condition with $\dot{q} = 10^7\ {}^{\text{W}}/_{\text{m}^2}$ was imposed on the leftmost, sloping face. Initial temperature was 500K. The figure below depictes the results for four times between t_0 and t_f.

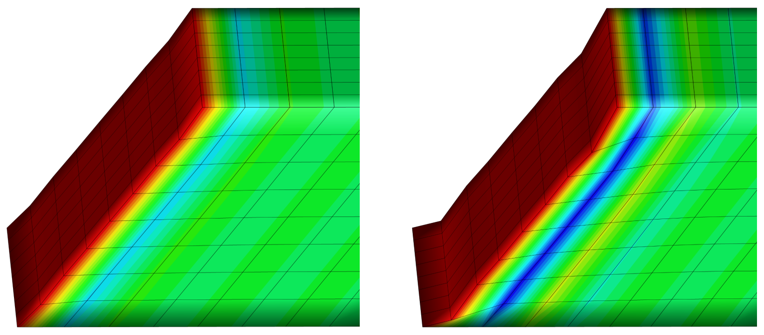


(a) $t = 0$ **(b) $t \approx 35$**

(c) $t \approx 15$ **(d) $t = 50$**

Figure 44. Parallelepiped CHAR Simulation Results. All figures shown have the same view and scaling. Recession proceeds smoothly; the bit of internal drift does not damage the shape.

While we observe some planar drift of nodes on the frontmost side, the structure and frame shape are maintained. The color banding shown is an artifact of insufficient grid and causes inaccuracies in the thermal response results. For this demonstration, these errors are of no major consequence. A recurring trend throughout the examples, the simple scheme fares poorly on this problem geometry, displaying significantly shape deformation early and failing shortly. The figure below depicts severity of failure.



(a) **$t \approx 3.5$** (b) **$t = 9.6684$**

Figure 45. Unsimple Problem for Simple Scheme. The simple mesh motion scheme is incapable of handling this seemingly simple geometry. Shape and mesh corruption, especially at the bottom edge, appears quickly in (a). (b) displays the final time step before failure, where the parallelepiped has deformed significantly and displays internal tangling on the front side.

VI. Conclusion

The tangle-free scheme for mesh motion is a highly powerful and flexible method that expands the possible problem geometries for ablation modeling. In 2D, it does so at a marginal increase in computational cost over a simple scheme that always uses the normal as the direction of motion. In 3D, the use of radial basis functions induces a significant spike in computational cost compared to the simple scheme; however, the increases in the number of allowable geometries more than justify the price that must be paid.

Above all, the greatest strength of the tangle-free method is its adaptability. At its core, it is nothing more than a collection of recipes for intelligently selecting directions of motion. While developed here for a linear elastic system, no particulars of the finite element formulation play the slightest role in the method's algorithms. As a result, the tangle-free scheme can be adapted for use with other mesh redistribution techniques that require a direction and magnitude of motion, including interpolation methods.

We have demonstrated tangle-free's capabilities for a number of test geometries. However, our testing procedure can improve and it is necessary to test more cases and "real world" problems. It is likely that we can make minor tweaks to the algorithms presented here to make them even more robust. In particular, the handling of double slider edge corners may have difficulties with some geometries. Future work is to make use of radial basis functions to obtain a more suitable direction of motion for this case.

VII. Acknowledgements

The author wishes to acknowledge the gracious support of his JSC Applied Aeroscience and CFD Branch colleagues and brother developers of `CHAR`:

- Adam Amar, who suggested the name "tangle-free" and contributed greatly to the design of the 2D verification scheme. Fruitful discussions with him during development inspired key innovations.
- Giovanni Salazar, whose input and aid greatly accelerated adaptation of the analytic sphere verification problem for use in `CHAR` from the notes of Mr. Amar and Dr. Ben Blackwell
- Dr. Benjamin Kirk, who introduced the author to this mesh motion problem, suggested an approach rooted in change of basis, and alerted the author to the existing work on radial basis functions.
- Brandon Oliver, who provided valuable input on likely usage scenarios and desirable features.

References

[1]A. Amar, J. Droba, B. Kirk, B. Oliver, and G. Salazar. Overview of the CHarring Ablator Response CHAR Code. In *AIAA Aviation and Aeronautics Forum and Exposition*, Washington, D.C., June 13-17, 2016 *(*submitted for publication).

[2]W. Bangerth, R. Hartmann, and G. Kanschat. `deal,ii` - A General-purpose Object-oriented Finite-element Library. *ACM Transactions in Mathematical Software*, 33(4), 2007.

[3]D. S. Broomhead and D. Lowe. Multivariable Functional Interpolation and Adaptive Networks. *Complex Systems*, 2:321–355, 1988.

[4]D. S. Broomhead and D. Lowe. Radial Basis Functions, Multi-variable Functional Interpolation, and Adaptive Networks. Technical Report RSRE-4148, Royals Signals and Radar Establishment, July 1988.

[5]J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and Representation of 3D Objects with Radial Basis Functions. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 67–76, New York, NY, USA, 2001. ACM.

[6]A. de Boer, M. S. van der Schoot, and H. Bijl. Mesh Deformation Based on Radial Basis Function Interpolation. *Computational Structures*, 85(11-14):784–795, June 2007.

[7]L. C. Evans. *Partial Differential Equations*, volume 19 of *Graduate Texts in Mathematics.* American Mathematical Society, 1st edition, 1998.

[8]G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, 3rd edition, 1996.

[9]B. T. Helenbrook. Mesh Deformation Using the Biharmonic Operator. *International Journal for Numerical Methods in Engineering*, 56(7):1007–1021, 2003.

[10]W. Huang and R. D. Russell. *Adaptive Moving Mesh Methods*, volume 174 of *Applied Mathematical Sciences.* Springer, 2011.

[11]C. Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method.* Dover, 2009.

[12]B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. `libMesh`: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. *Engineering with Computers*, 22(3–4):237–254, 2006.

[13]J. R. Lachaud, A. Martin, I. Cozmuta, and B. Laub. Ablation Test-case Series 1. In *4th AFSOR/SNL/NASA Ablation Workshop*, Albuquerque, NM, March 1-3, 2010.

[14]J. Landry, A. Soula'imani, E. Luke, and A.B. Haj Ali. Robust Moving-mesh Algorithms for Hybrid Stretched Meshes: Application to Moving Boundaries Problems. In *AIAA SciTech: 54th AIAA Aerospace Sciences Meeting*, San Diego, CA, January 4-8, 2016.

[15]D. Lynch and K. O'Neill. Elastic Grid Deformation for Moving Boundary Problems in Two Space Dimensions. *Finite Elements in Water Resources*, 2:7.111–7.120, 1980.

[16]G. Salazar, J. Droba, A. Amar, and B. Oliver. Development, Verification, and Validation of Enclosure Radiation Capabilities in the CHarring Ablator Response CHAR Code. In *AIAA Aviation and Aeronautics Forum and Exposition*, Washington, D.C., June 13-17, 2016.

[17]M. Utku and G. F. Carey. Boundary Penalty Techniques], journal = Computer Methods in Applied Mechanics and Engineering, volume = 30, pages = 103–118, year = 1982.

[18]H. Wendland. Piecewise Polynomial, Positive Definite and Compactly Supported Radial Functions of Minimal Degree. *Advances in Computational Mathematics*, 4(1):389–396, 1995.

[19]J. E. Wiebenga and I. D. Boyd. Computation of Multi-Dimensional Material Response Coupled to Hypersonic Flow. In *42nd AIAA Thermophysics Conference*, number AIAA 2012-2873, New Orleans, NO, June 25-28, 2012.

[20]R. Yokota, L. A. Barba, and M. G. Knepley. PetRBF — A Parallel $\mathcal{O}(n)$ Algorithm for Radial Basis Function Interpolation with Gaussians. *Computer Methods in Applied Mechanics and Engineering*, 199(25–28):1793 – 1804, 2010.