Mesh Deformation Boundary Conditions for Three-Dimensional Ablation Solvers

Adam J. Amar, Justin Cooper, A. Brandon Oliver
NASA Lyndon B. Johnson Space Center
2101 NASA Parkway, Houston, TX, 77058

Giovanni Salazar, Lucas Agricola
Corvid Technologies, Mooresville, NC, 28117

A method for determining boundary node displacements for three-dimensional ablation solvers is developed. The approach is applicable to a wide range of mesh motion algorithms used to determine internal node displacements. The method leverages radial basis functions to analytically define the geometry to which the boundary nodes should be constrained. Constrained optimization, elasticity, and spring analogy solvers are utilized to redistribute the nodes on the evolved geometry surface while maintaining mesh quality. The method is implemented in the *CH*arring *A*blator *R*esponse (*CHAR*) code, and a realistic demonstration case is presented on the Boundary Layer Transition (BOLT) flight experiment configuration. The method is shown to be robust for highly-curved geometries with large domain deformations.

Nomenclature

$\boldsymbol{F}$	Force vector (N)
$\boldsymbol{u}$	Displacement vector with Cartesian components u, v, w (m)
$\boldsymbol{v}$	Velocity $(\mathrm{m}/\mathrm{sec})$
$\boldsymbol{x}$	Position vector with Cartesian components x, y, z (m)
$\dot{\boldsymbol{s}}$	Surface recession velocity $(\mathrm{m}/\mathrm{sec})$
$\dot{m}$	Ablation (surface mass loss) flux $(\mathrm{kg}/\mathrm{m}^2\cdot\mathrm{sec})$
$\dot{Q}$	Volumetric energy source $(\mathrm{W}/\mathrm{m}^3)$
$\dot{q}_{cond_s}$	Conductive heat flux into solid $(\mathrm{W}/\mathrm{m}^2)$
$\dot{q}_{rad}$	Radiative heat flux $(\mathrm{W}/\mathrm{m}^2)$
$\dot{s}$	Surface recession rate $(\mathrm{m}/\mathrm{sec})$
$\hat{\boldsymbol{n}}$	Unit normal vector
$\tilde{\boldsymbol{k}}$	Thermal conductivity tensor $(\mathrm{W}/\mathrm{m}\cdot\mathrm{K})$
c, λ	Radial basis function coefficients
c_{ij}	Material elasticity constants (Pa)
E	Young's modulus (Pa) or Number of elements containing a node
e_o	Total internal energy (J/kg)
f	Distance from radial basis function center to surface (m)
G	Shear modulus (Pa)
h	Enthalpy (J/kg)
k	Spring constant $(\mathrm{kg}/\mathrm{sec}^2)$
L	Distance between adjacent nodes on domain edge (m)
P	Pressure (Pa)
R	Residual
r	Distance from node to radial basis function center (m)
s	Distance function (m)
T	Temperature (K)

t	Time (sec)
u', v'	Displacements in tangential directions τ_1 and τ_2 respectively (m)

Symbols

α	Absorptivity
Δt	Time step (sec)
$\dot{\omega}$	Mass source $(\mathrm{kg/m^3 \cdot sec})$
$\hat{\boldsymbol{\tau}}$	Unit tangent vector
Λ	Lagrange multiplier
$\mathcal{L}$	Lagrangian function
μ	Dynamic viscosity $(\mathrm{Pa \cdot sec})$
ν	Test function or Poisson's ratio
ϕ	Porosity or Basis function
ψ	Basis function for finite element method
ρ	Density $(\mathrm{kg/m^3})$
$\rho_e u_e C_H$	Film coefficient $(\mathrm{kg/m^2 \cdot sec})$
σ	Stefan-Boltzmann constant $(\mathrm{W/m^2 \cdot K^4})$
τ_i	Tangential direction (m)
$\tilde{\boldsymbol{\kappa}}$	Permeability tensor $(\mathrm{m^2})$
ε	Emissivity

Subscripts and Superscripts

∞	Farfield quantity
g	Quantity of the gas
m	Quantity of the mesh
n	Time level index
$node$	Quantity of a node
o	Total/Stagnation quantity or Initial condition
r	Recovery quantity
s	Quantity of the solid
w	Quantity at the wall/surface

Acronyms

AD	Automatic Differentiation
ALE	Arbitrary Lagrangian-Eulerian
AMR	Adaptive Mesh Refinement
BOLT	Boundary Layer Transition flight experiment
LU	Lower-Upper decomposition
PDE	Partial Differential Equation
RBF	Radial Basis Function

I. Introduction

For many disciplines, deformation of discretized computational domains (e.g. meshes) is a common aspect of solving partial differential equations (PDEs) with numerical methods. These include, but are not limited to, structural mechanics, aeroelasticity, store separation, and multi-phase interface modeling. Each discipline's physics comes with its own nuances and challenges that influence the approach for domain deformation. Ablation modeling is a discipline that has leveraged mesh deformation from its infancy.[1] Herein, mesh deformation refers to the temporal evolution of the mesh to represent the changing geometry of the system. In general, mesh deformation involves some combination of cell/element/node translation, contraction, elimination, cutting, and re-meshing. Mesh deformation does not refer to adaptive refinement of an otherwise static mesh geometry.

For ablation modeling, the mesh deformation problem in one dimension is relatively simple, yet it has been solved in multiple ways. Moyer and Rindal[1] implemented a translating and node-dropping scheme in the Charring Materials Ablation (CMA) code where the first $N-1$ mesh elements translated, without volume reduction, at the surface recession rate. The last, N^{th}, element contracts until reaching a critical minimum volume, at which time it is eliminated

from the domain, thereby reducing the total element and node counts by one. Since the development of CMA, the more common approach has been to treat the one-dimensional domain as a linear elastic solid.[2–4] With this approach the elements contract at the same rate as the domain. Consequently, each node translates at a fraction of the surface recession rate inversely proportional to the node's depth from the surface. With this method, relative element spacing and overall element count is maintained.

Beyond one dimension, TITAN[5] (two-dimensional) and 3dFIAT[6] (three-dimensional) rely on an initial structured mesh definition to provide recession paths along which the grid can move. Other two- and three-dimensional ablation codes have extended the concept of treating the domain as an elastic solid.[7–9] While the elasticity solution in one dimension is closed-form, the extension to higher dimensions introduces the need for PDE solvers. Other PDE-based approaches include the use of Laplacian smoothing as exemplified by PATO[10] through the use of the mesh motion capabilities in OpenFOAM.[11] Another physics analog approach is implemented in the KATS code,[12] where neighboring nodes are assumed to be connected by elastic springs, and the equilibrium solution gives the nodal displacements. Other codes have implemented algebraic methods to determine the mesh displacements. These are typically based on interpolation such as the radial basis function (RBF) approach in Icarus.[13] In general, these methods that redistribute the nodes throughout the ablating domain, such as the PDE and algebraic methods, will be referred to as "mesh motion methods" in this work. Finally, an element cutting and near-surface re-meshing approach has been implemented in the Hero code.[14] Unlike the PDE-based and algebraic methods, the element cutting approach does not move and redistribute nodes throughout the domain and is not characterized as one of the "mesh motion methods" discussed in this work. However, it has been shown to be a valid domain deformation approach for ablation problems.

All of the methods come with their advantages and disadvantages for a number of metrics including mesh quality, robustness to topological changes, speed, and memory. In general, each has the ability to produce high-quality meshes over a range of geometric scenarios. A good review of the mesh motion methods is provided by Selim and Koomullil.[15] Each of these methods requires boundary displacement information to drive the solution for internal displacements. The focus of this paper is the development of the boundary displacement data recognizing that the subsequent determination of internal node displacements will follow via one of the aforementioned mesh motion methods. For PDE-based solvers, the boundary displacements will serve as Dirichlet boundary conditions on nodal displacements. Alternatively, for interpolation-based algebraic methods, the boundary displacements serve as the known data points from which the internal node displacements are interpolated. The method developed herein is applicable to any problem with mesh motion, but many of the nuanced implementation details are specifically designed to overcome challenges associated with ablation problems. The *CH*arring *A*blator *R*esponse (*CHAR*) code[8,16] is used as a testbed for the algorithm, and demonstration cases will be presented.

II. Background

CHAR is a one-, two-, and three-dimensional Galerkin finite-element code that solves the heat and mass transport equations in a pyrolyzing (charring), ablating, porous medium. The governing equations are:

$$\begin{aligned} \text{Energy:}\quad & \left.\frac{\partial(\rho e_o)}{\partial t}\right|_{node} - \boldsymbol{\nabla}\cdot\left(\tilde{\boldsymbol{k}}\boldsymbol{\nabla}T\right) + \boldsymbol{\nabla}\cdot\left(\phi\rho_g h_{o_g}\boldsymbol{v}_g\right) - \boldsymbol{v}_m\cdot\boldsymbol{\nabla}(\rho e_o) - \dot{Q} = 0 \\ \text{Gas Mass:}\quad & \left.\frac{\partial(\phi\rho_g)}{\partial t}\right|_{node} - \dot{\omega}_g + \boldsymbol{\nabla}\cdot(\phi\rho_g\boldsymbol{v}_g) - \boldsymbol{v}_m\cdot\boldsymbol{\nabla}(\phi\rho_g) = 0 \end{aligned} \tag{1}$$

where the $node$ subscript denotes conservation with respect to a constant location in the moving coordinate frame (such as a node in the moving mesh) and Darcy's law defines the gas velocity field by

$$\boldsymbol{v}_g = -\frac{\tilde{\boldsymbol{\kappa}}}{\phi\mu}\boldsymbol{\nabla}P \tag{2}$$

Additionally, the solid mass conservation equation, given by

$$\frac{\partial\rho_s}{\partial t} = \dot{\omega}_s = -\dot{\omega}_g \tag{3}$$

is solved in a stationary reference frame and treated as a constitutive relation when defining the temporal derivative in the energy equation.

Of particular interest to the current work are the Arbitrary Lagrangian-Eulerian (ALE) terms with the mesh velocity, $\boldsymbol{v}_m$, that arise from solving the PDEs on a moving mesh. In *CHAR*, the nodal velocities are calculated using a

finite-difference approximation between the old and new meshes

$$\boldsymbol{v}_{m_i}^{n+1} = \frac{\boldsymbol{x}_i^{n+1} - \boldsymbol{x}_i^n}{\Delta t} \tag{4}$$

where $\boldsymbol{x}$ is a position vector, i is the node index, n is the time index, and Δt is the time step. The nodal displacement between time steps,

$$\boldsymbol{u}_i = \begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} = \boldsymbol{x}_i^{n+1} - \boldsymbol{x}_i^n \tag{5}$$

is the solution of a linear elastic solver as described by Amar et al.[8] It is important to note that the nodal mesh velocities internal to the mesh are arbitrary in that a given node is not attached to a specific point of mass. The nodes are free to move in space and time internal to the domain as long as mesh quality is maintained and the velocities are appropriately accounted for in the conservation equations. The boundary nodes have the added constraint that they must maintain an accurate representation of the shape as the geometry evolves. The remainder of this work is focused on determining the boundary node displacements, which will be the boundary conditions for the linear elasticity solver in *CHAR*.

The physics solution from the ablation problem results in known ablation (or mass loss) fluxes, $\dot{m}$, at discrete points on the boundary, such as nodes, cell face centers, or quadrature points. For *CHAR*, the ablation fluxes are known at quadrature points, which could be anywhere on an element boundary face, depending on the quadrature rule and order. Knowing the material density, ρ, and outward surface unit normal, $\hat{\boldsymbol{n}}$, the surface recession velocity at these discrete points can be determined

$$\dot{\boldsymbol{s}} = -\dot{s}\hat{\boldsymbol{n}} = -\frac{\dot{m}}{\rho_s}\hat{\boldsymbol{n}} \tag{6}$$

Previous versions of *CHAR* have utilized the discrete recession velocities to enforce Dirichlet conditions on the boundary node displacements

$$\boldsymbol{u} = \dot{\boldsymbol{s}}\Delta t \tag{7}$$

This approach works fine for many simple topologies. Once corners, highly-curved geometries, and non-uniform recession are introduced, enforcement of all three components of the displacement vector on a receding surface can become overly constraining and lead to poor quality or tangled meshes. Instead, a better approach would be to allow a given node to recede only under the constraints that it ends up in a location that lies on the physical surface and maintains mesh quality.

Ablation problems often require the boundary of a domain to slide. This implies the boundary does not recede, but the nodes are required to move along the boundary, maintaining its shape, while an adjacent boundary recedes. Figure 1 shows a simplified two-dimensional representation of a typical arcjet test specimen. The top curved surface has prescribed displacements using the normal displacement method in Eq. (7), and the sides slide to maintain shape as the surface ablates. This can be accomplished by setting the normal component of displacement to zero, and letting the second component come out of the linear elasticity solution. This is a straightforward boundary condition when the surface normal is aligned with one of the Cartesian displacement components u, v, or w. Complexity is increased if the surface is not Cartesian aligned. Finally, the boundary condition fails and will not maintain shape if the sliding boundary were curved. Consequently, a more generally applicable sliding condition would have the same constraint as previously discussed for receding boundaries, which is to allow a given node to move only under the constraints that it ends up in a location that lies on the physical surface and maintains mesh quality.

This work seeks to develop a methodology to define boundary node displacements as inputs to the mesh motion solver. The primary considerations when developing the approach, given in priority order, are

1. Maintain accurate physics-based representation of geometry
2. Maintain high-quality meshes under large deformations
3. Flexible, robust, and automatic implementation able to handle a wide variety of geometric scenarios without over-burdening the user with onerous input parameters
4. Fast and scalable
5. Low memory footprint

While some work has gone into priorities 4 and 5, there are still substantial gains to be had with already identified forward work to be discussed.

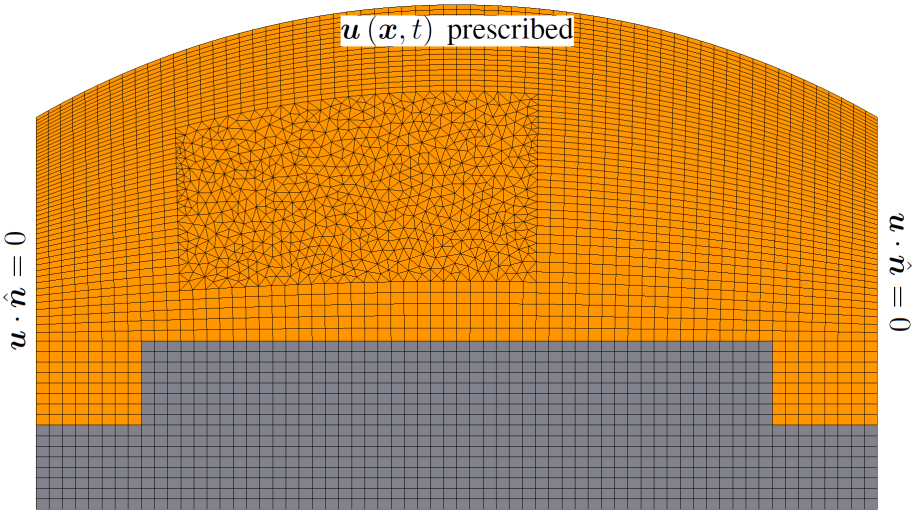


Figure 1. Arcjet specimen from *CHAR*'s mesh motion demonstration case.[8]

III. Nomenclature

To establish proper nomenclature, it is helpful to utilize a geometric configuration that is simple to explain yet complex enough to demonstrate the method. Figure 2 shows a discretized, twisted rectangular prism. The six gray surfaces are called "domain boundaries." Domain boundaries are a set of contiguous element faces that collectively represent one topological side of a domain and have the same mobility or motion assumption. The three possible mobility or motion assumptions for domain boundaries are:

- *Fixed*: No nodes on this domain boundary will move for the duration of the simulation.
- *Sliding*: Nodes can move during the simulation, but they are constrained to the domain boundary's geometric definition given in the initial mesh.
- *Receding*: Nodes can recede and/or slide constrained to the instantaneous definition of the domain boundary.

The blue segments in Figure 2 are called "domain edges." Domain edges are the intersection of two, and only two, domain boundaries. Domain edges are a set of contiguous element edges. A domain edge's mobility is inherited from the two domain boundaries that intersect to define it. The possible mobility or motion assumptions for domain edges are:

- *Fixed-Fixed*, *Sliding-Fixed*, and *Receding-Fixed*: No nodes on this domain edge will move for the duration of the simulation.
- *Sliding-Sliding* and *Receding-Sliding*: Nodes can move during the simulation, but they are constrained to the domain edge's geometric definition given in the initial mesh.
- *Receding-Receding*: Nodes can recede and/or slide constrained to the instantaneous definitions of the two receding domain boundaries.

The red spheres in Figure 2 are called "domain corners." Domain corners are the intersection of three or more domain boundaries. A domain corner is a special node that, in general, could be part of more than one element. A domain corner's mobility is inherited from the three (or more) domain boundaries that intersect to define it. The possible mobility or motion assumptions for domain corners are:

- *Sliding-Sliding-Sliding* and *<any boundary>-<any boundary>-Fixed*: Domain corner will not move for the duration of the simulation.
- *Receding-Sliding-Sliding*, *Receding-Receding-Sliding*, and *Receding-Receding-Receding*: Nodes can move during the simulation, but they are constrained to the instantaneous intersection point of the three domain boundaries that define it.

A domain edge is inclusive of the two domain corners that define its end points, and a domain boundary is inclusive of the domain edges that define its border. Consequently, corner and edge nodes are each associated with multiple domain boundaries.

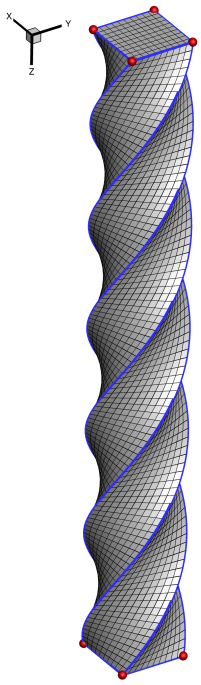


Figure 2. Twisted prism mesh showing domain boundaries (gray), edges (blue), and corners (red).

IV. Approach

The goal of the method is to define every displacement component, u, v, and w, for every node on every domain boundary. These data are then provided to the mesh motion algorithm as boundary conditions. The high-level steps to the approach are:

1. *Geometry Data Collection*: Survey mesh to detect edge and corner nodes and their respective mobility based on user-defined boundary conditions.

2. *Sliding Boundary Definition*: At simulation initialization prior to time integration, generate an analytical definition for each sliding domain boundary. These data will be static and stored in memory for the duration of the simulation.

3. *Surface Recession Rate Solution*: During each time step, determine recession rates, $\dot{s}$, at discrete locations from the physics solution (typically a surface energy balance). This will be at quadrature points for the *CHAR* implementation.

4. *Receding Boundary Definition*: Utilize the surface normals and discrete recession rate data to construct an analytical definition for each receding domain boundary.

5. *Domain Corner Node Displacements*: Determine domain corner node displacements from intersection of associated domain boundaries.

6. *Domain Edge Node Displacements*: Determine domain edge node displacements from the known domain corner displacements and the geometric constraints from the two associated domain boundaries.

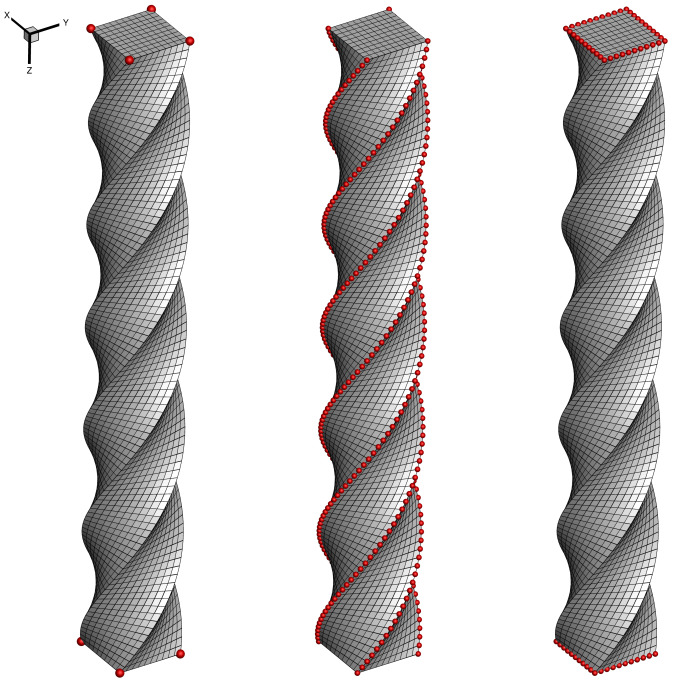
7. *Domain Boundary Node Displacements*: Determine the domain boundary node displacements from the known domain edge displacements and the geometric constraint.

8. *Determine Domain Interior Node Displacements*: Determine the interior node displacements from the known domain boundary displacements. This can be done through a variety of approaches as outlined in Section I.

The twisted prism configuration in Figure 2 will be used as an example to step through the algorithm. The top and bottom ($-z$ and $+z$ respectively) square domain boundaries will be receding, and the four twisted side domain boundaries will be sliding. The following sections walk through the algorithmic details of the relevant steps above. Steps 3 and 8 are not the focus of this work and will not be discussed in further detail.

IV.A. Geometry Data Collection

For *CHAR*, the user needs to specify if a domain boundary is receding, sliding, or fixed. Upon execution, *CHAR* will detect and store all of the pertinent geometric and mobility details. For the twisted prism problem, the domain corner and edge nodes are identified as shown in Figure 3, and the domain boundaries with which they are associated are stored.



(a) Receding-Sliding-Sliding Corner Nodes (b) Sliding-Sliding Edge Nodes (c) Receding-Sliding Edge Nodes

Figure 3. Node geometry data detected and saved at initialization.

IV.B. Sliding Boundary Definition

RBFs are used to generate analytical definitions for sliding domain boundaries. This can be done once, at the beginning of a simulation, and stored for repeated use. The work of Carr et al. was heavily leveraged for this work.[17] The concept is to define a function, $s(\boldsymbol{x})$, where s is the signed distance of a point, $\boldsymbol{x}$, away from the surface. Consequently, $s(\boldsymbol{x}) = 0$ defines the surface. Per Carr, s is of the form

$$s(\boldsymbol{x}) = c_1 + c_2 x + c_3 y + c_4 z + \sum_{i=1}^{N} \lambda_i \phi(r_i) \tag{8}$$

where $\phi(r_i)$ is the basis function with $r_i = |\boldsymbol{x} - \boldsymbol{x}_i|$. The points, $\boldsymbol{x}_i$, of which there are N, are known as the centers of the RBF. The centers are the data points that are provided for surface fitting, and the distance from the surface, $s(\boldsymbol{x}_i) = f_i$, is known for each of the centers. Typically, best surface fits are achieved when both on-body ($f_i = 0$)

and off-body ($f_i \neq 0$) centers are provided. In this work, the centers are chosen to be each of the nodes on the domain boundary of interest. Additionally, two centers are provided for each element face centroid, one on each side, offset a small distance in the outward and inward normal directions respectively. This gives $N = N_{nodes} + 2N_{elems}$ centers.

For this work, the biharmonic basis function, $\phi(r_i) = r_i$ is used which gives

$$s(\boldsymbol{x}) = c_1 + c_2 x + c_3 y + c_4 z + \sum_{i=1}^{N} \lambda_i |\boldsymbol{x} - \boldsymbol{x}_i| \tag{9}$$

As detailed by Carr,[17] the c and λ coefficients in (9) are the solution to the linear system

$$\begin{bmatrix} \tilde{\boldsymbol{A}} & \tilde{\boldsymbol{P}} \\ \tilde{\boldsymbol{P}}^T & \tilde{\boldsymbol{0}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{c} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix} \tag{10}$$

where

$$A_{i,j} = |\boldsymbol{x}_i - \boldsymbol{x}_j| \text{ for } i, j = 1, ..., N$$

$\tilde{\boldsymbol{P}}$ is the matrix with i^{th} row $(1, x_i, y_i, z_i)$, $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_N)^T$, $\boldsymbol{c} = (c_1, c_2, c_3, c_4)^T$, and $\boldsymbol{f} = (f_1, ..., f_N)^T$. The resulting matrix is symmetric and dense. The current implementation in *CHAR* uses the Pliris dense linear system solver package,[18] which is an object-oriented interface to a parallelized LU solver. Pliris is distributed as part of the Trilinos library.[19] The four c and N λ coefficients resulting from the linear system solution are stored in memory for use later when the RBF needs to be evaluated, which will be discussed in subsequent sections. Figure 4 shows a rendering of the RBF definition for one of the four sliding domain boundaries. It is important to recognize that the RBFs are defined over all space. It is only the zero-isosurface, defined over the space covered by the provided centers, that defines the surface. Consequently, the RBF's character is unpredictable outside of the provided centers, but this region of the RBF will not be utilized for sliding domain boundaries since the evolving geometry will always be bounded by the initial geometry. However, this may not be the case for expanding domains resulting from material swelling, which is not handled with the curent approach.

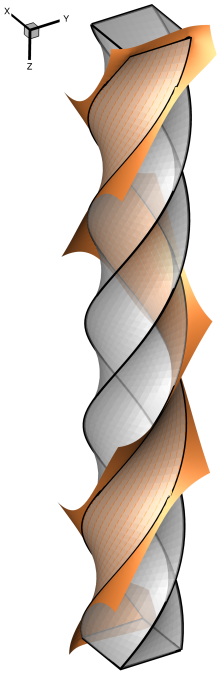


Figure 4. RBF rendering for one of four sliding domain boundaries wrapped around translucent domain.

A possible opportunity for speed-up in the RBF fitting process is the utilization of basis functions, other than the biharmonic, which would introduce compact support. This would make the linear system sparse, which could be solved more rapidly. The biharmonic basis function with non-compact support was implemented during initial

development due to the known surface-fitting quality characteristics discussed by Carr.[17] Other potential speed-up opportunities include the implementation of the RBF fast-fitting algorithm described by Beatson et al.[20] and the center reduction (greedy) algorithm described by Carr.[17]

IV.C. Receding Boundary Definition

With the geometric definitions of the sliding domain boundaries stored in the form of RBF coefficient data, time integration commences and continues until the onset of surface recession. Once this occurs, the physics solution provides recession rate data at the quadrature points (step 3 in the overall process from Section IV). These discrete recession rate data must be processed to provide an analytical definition of the recessed surface. RBFs will once again be leveraged for this analytical definition, but first a discrete representation of the new surface must be reconstructed from the surface recession rate data. To accomplish this, the algorithm interpolates nearby quadrature point recession rate data to the nodes using RBF interpolation. The specific steps, which are performed on a node-by-node basis, are outlined as follows:

1. Find the set of E element faces on the given domain boundary that contain the node of interest.

2. Use Wendland RBFs[21] to interpolate recession rates *from* the collection of quadrature points on the E element faces *to* the node. Currently, Wendland RBFs are being used since that is what is readily available in libMesh,[22] which is the foundational finite-element library with which *CHAR* was developed. Future work will include exploring the use of RBFs with non-compact support, which could improve interpolation accuracy, since only very nearby quadrature points are being used as source data for the RBF.

3. To prevent interpolation overshoots that could impact grid quality, limit the recession rate to the maximum and minimum observed in the collection of quadrature points used to construct the RBF.

4. With the interpolated recession rate value, $\dot{s}_{node}$, determine the nodal displacement by averaging the nodal displacements from all E element faces according to

$$\boldsymbol{u} = -\dot{s}_{node}\Delta t \frac{\sum_{e=1}^{E} \hat{\boldsymbol{n}}_e}{\left|\left|\sum_{e=1}^{E} \hat{\boldsymbol{n}}_e\right|\right|} \tag{11}$$

   where $\hat{\boldsymbol{n}}_e$ is the outward unit normal at the node for element e. This step is necessary because each element has a different definition for the normal direction at the node.

It is important to note that these average nodal displacements are *not* the final displacements to be used by the mesh motion solver. Rather, they are temporary displacements only utilized to generate the RBF representation of the receded surface. As an alternative to the above approach, the face offsetting method of Jiao[23] has also been implemented in *CHAR* to define the receded surface, but those details are omitted here since no results with this method are shown. Through testing, Jiao's method has been shown to produce comparable results to the method described herein.

Once the temporary nodal displacements are determined, an RBF representation of the receding surface can be generated in the same manner described in Section IV.B. The only difference is that on- and off-body "ghost" centers are added around the edges to ensure intersection with neighboring domain boundary RBFs. The locations of the ghost centers are determined with a zeroth order extrapolation. The RBF coefficients for a given receding domain boundary are stored, potentially for multiple time steps, until the surface recedes again thus requiring regeneration of the RBF. Figure 5 shows a rendering of the receding domain surface RBFs after one time step. For this problem, the receding surfaces are flat, and the recession is uniform. A subsequent demonstration problem will exhibit some more complex receding surface behavior.

IV.D. Domain Corner Node Displacements

Each sliding and receding boundary now has an analytical representation through the RBFs. Now the final displacements of the domain corner nodes can be determined. Each domain corner is simply the intersection of the three domain boundary RBFs associated with it. Finding the intersection can be posed as a constrained optimization problem that can be solved with the method of Lagrange multipliers. The constrained optimization problem can be stated as:

$$\text{Maximize } -\left|\boldsymbol{x}^{n+1} - \boldsymbol{x}^{n}\right| \text{ subject to } s_k\left(\boldsymbol{x}^{n+1}\right) = 0 \text{ for } k = 1, ..., K \tag{12}$$

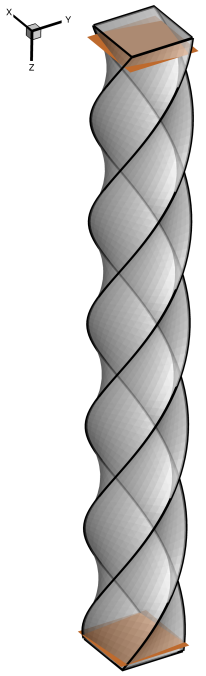


Figure 5. RBF rendering for both receding domain boundaries after first time step.

where $\boldsymbol{x}^{n+1}$ is the new/displaced position of the corner node, $\boldsymbol{x}^n$ is the old position of the corner node, and $K = 3$ for a corner since it is the intersection of three RBFs. This gives the Lagrangian function

$$\mathcal{L} = -\left(x^{n+1} - x^n\right)^2 - \left(y^{n+1} - y^n\right)^2 - \left(z^{n+1} - z^n\right)^2 - \sum_{k=1}^{K} \Lambda_k s_k \tag{13}$$

where Λ_k are the Lagrange multipliers. Taking the derivatives of Eq. (13) with respect to x, y, z, and Λ_k gives the $3 + K$ residual functions whose solution gives the displaced corner node location, $\boldsymbol{x}$.

$$R_i = -\frac{\partial \mathcal{L}}{\partial x_i} = 2\left(x_i^{n+1} - x_i^n\right) + \sum_{k=1}^{K} \Lambda_k \frac{\partial s_k}{\partial x_i} \text{ for } i = 1, 2, 3 \tag{14}$$

$$R_{3+k} = -\frac{\partial \mathcal{L}}{\partial \Lambda_k} = s_k \text{ for } k = 1, ..., K \tag{15}$$

where the derivatives of the RBF distance function, s, can be found in Appendix A. The nonlinear system of equations is solved fully-implicit using Newton's method with analytical Jacobians shown in Appendix B. It is important to note that the corner node displacement problem, like the edge and boundary node displacement problems to follow, requires repeated evaluations of the RBFs. Due to the non-compact support of the biharmonic basis function, evaluation of the RBFs requires looping over every center in the RBF every time s or one of its derivatives is evaluated. An opportunity to speed-up the RBF evaluation process is to explore the use of basis functions with compact support. Other options are to implement 1) a fast RBF evaluation method based on Fast Multipole Methods as described by Beatson[24] and/or 2) a center reduction (greedy) algorithm as proposed by Carr.[17]

IV.E. Domain Edge Node Displacements

Once the domain corner node displacements are known, these can serve as boundary conditions to determine the edge node displacements since each domain edge terminates with a domain corner node on each of its two ends. To determine the non-corner edge node displacements, two methods have been implemented: 1) a one-dimensional constrained spring analogy and 2) a uniformly contracting grid scheme. The code allows the user to select which methodology to employ.

IV.E.1. 1D Constrained Spring Analogy

This approach utilizes the concept of nodes connected by springs and constrained to the two RBFs for the domain boundaries associated with the edge. The constraint implies that the node's only degree of freedom is tangentially along the edge. The equilibrium solution (i.e. the sum of the tangential forces is zero) provides the new nodal locations. The spring analogy algorithm developed in this work heavily leverages the work of Blom[25] but adds in the shape constraints and reduces the degrees of freedom. The sum of the tangential forces exerted on a node, i, by the springs connected to its J neighbors is

$$\boldsymbol{F}_i \cdot \hat{\boldsymbol{\tau}}_i = \hat{\boldsymbol{\tau}}_i \cdot \sum_{j=1}^{J} k_{ij}^{n+1} \left(\boldsymbol{x}_i^{n+1} - \boldsymbol{x}_i^n - \boldsymbol{x}_j^{n+1} + \boldsymbol{x}_j^n \right) = 0 \tag{16}$$

where $J = 2$. The superscript n denotes the time level. $\hat{\boldsymbol{\tau}}_i$ is the local unit tangent vector to the edge, and the spring constant is given by

$$k_{ij}^{n+1} = - \left[\left(\boldsymbol{x}_i^{n+1} - \boldsymbol{x}_j^{n+1} \right) \cdot \left(\boldsymbol{x}_i^{n+1} - \boldsymbol{x}_j^{n+1} \right) \right]^{-a} \tag{17}$$

For this work, $a = 1$. a controls the linearity of the spring stiffness where higher values make shorter springs stiffer. Eq. (16) is one equation with three unknowns, x^{n+1}, y^{n+1}, and z^{n+1}. The two additional equations for each node are the shape constraints from the two domain boundaries.

$$s_b \left(\boldsymbol{x}_i^{n+1} \right) = 0, \text{ for } b = 1, 2 \tag{18}$$

The unit tangent vector, $\hat{\boldsymbol{\tau}}_i$, is determined by taking the cross product of the normal vectors of the two RBFs for the domain boundaries associated with the edge and scaling the result to unit length. A unit normal vector, $\hat{\boldsymbol{n}}_i$, for each RBF is determined from

$$\hat{\boldsymbol{n}}_i = \frac{\boldsymbol{\nabla} s \left(\boldsymbol{x}_i^{n+1} \right)}{\left|\left| \boldsymbol{\nabla} s \left(\boldsymbol{x}_i^{n+1} \right) \right|\right|} \tag{19}$$

The derivatives of s for defining the gradient can be found in Appendix A. It is important to note that the initial guess for the nodal locations and subsequent iterates may not lie on the domain edge until the system is converged. Since the RBFs are defined in all space, this allows for meaningful calculations of the normal and tangent vectors even when the nodes do not satisfy the shape constraints during the iterative solution process prior to convergence.

The nonlinear system of equations (Eqs. (16) and (18)) is solved fully-implicit with Newton's method using the known domain corner node displacements as Dirichlet boundary conditions. The implementation in *CHAR* uses Automatic Differentiation (AD) to provide the Jacobians. The AD functionality is provided by Sacado,[26] which is distributed as part of the Trilinos library.[19]

IV.E.2. Uniformly Contracting Grid

The contracting grid scheme described by Blackwell and Hogan[27] is extended to apply to domain edges in three dimensions. The concept is that the relative grid spacing along the edge is maintained as it deforms through the duration of the simulation. Alternatively, it can be stated that each edge element undergoes the same percent contraction (or extension), which is equivalent to the overall percent contraction of the entire edge. For each i^{th} node along the domain edge, the residual equation to enforce this behavior is

$$R_i = 0 = \left(\frac{L}{L_o} \right)_{i \to i-1} - \left(\frac{L}{L_o} \right)_{i \to i+1} \tag{20}$$

where the distance between adjacent nodes is

$$L_{i \to j} = \sqrt{\left(x_i^{n+1} - x_j^{n+1} \right)^2 + \left(y_i^{n+1} - y_j^{n+1} \right)^2 + \left(z_i^{n+1} - z_j^{n+1} \right)^2} \tag{21}$$

and L_o denotes the original distance between adjacent nodes on the edge. Eq. (20) is one equation with three unknowns, x^{n+1}, y^{n+1}, and z^{n+1}. The two additional equations for each node are the shape constraints from the two domain boundaries given by Eq. (18).

IV.F. Domain Boundary Node Displacements

With the corner and edge node displacements now known, the remainder of the boundary node displacements can be determined. The nodal displacements for fixed domain boundaries are trivially zero, and the nodal displacements for sliding and receding domain boundaries are described in this section. Again, there are two methods implemented: 1) a two-dimensional constrained spring analogy and 2) a two-dimensional manifold elasticity method.

IV.F.1. 2D Constrained Spring Analogy

The concept is similar to the spring analogy developed for the edge node displacements in Section IV.E. For domain boundaries, the nodes are only constrained to the one RBF associated with the domain boundary, and a degree of freedom is added to account for the second tangential direction. The force summation expressions for the two tangential directions are

$$\boldsymbol{F}_i \cdot \hat{\boldsymbol{\tau}}_i^m = \hat{\boldsymbol{\tau}}_i^m \cdot \sum_{j=1}^{J} k_{ij}^{n+1} \left(\boldsymbol{x}_i^{n+1} - \boldsymbol{x}_i^n - \boldsymbol{x}_j^{n+1} + \boldsymbol{x}_j^n \right) = 0, \text{ for } m = 1, 2 \tag{22}$$

and the single shape constraint is

$$s\left(\boldsymbol{x}_i^{n+1}\right) = 0 \tag{23}$$

The mesh on the boundary is, in general, a collection of quadrilateral and triangular element faces. For this algorithm, the set of J node neighbors includes all nodes on element faces that contain the i^{th} node, which includes diagonal nodes on quadrilateral faces.

The choice of the tangent vectors is somewhat arbitrary as long as the two tangent vectors are orthogonal to each other and both are orthogonal to the normal. In the current work, the three-dimensional Householder vector orthogonalization algorithm from Lopes et al. is used.[28] This method results in smoothly varying tangent vectors over the RBF surface definition.

The nonlinear system of equations (Eqs. (22) and (23)) is solved fully-implicit with Newton's method using the known domain edge displacements as Dirichlet boundary conditions. Like the edge algorithm in Section IV.E, *CHAR* uses AD to provide the Jacobians.

Beyond the basic algorithm described above, additional features were tested to see if boundary mesh quality could improve. First, the ortho-torsional spring terms described by Markou et al. were implemented.[29] However, there was not much noticeable improvement in mesh quality for the problems on which the algorithm was tested. The method could prove more valuable as a broader class of problems is attempted.

Next, a relative area preserving modification was implemented. This attempted to preserve the percentage area that an element face occupied on the boundary over all time, thereby attempting to preserve the mesh spacing on the boundary. This was accomplished by adding springs emanating from the centroid of a face to all of the nodes on the face. The springs would push if the area needed to grow and pull if it needed to shrink. This did lead to improved mesh quality under certain circumstances and has been preserved as an option in the code.

Finally, a monolithic solver was implemented, which simultaneously solved for all domain edge and domain surface displacements at once. This allowed the edge node displacements to be influenced by the boundary nodes instead of being independent. In general, the meshes were of poorer quality for the problems on which the algorithm was tested. However, future work will entail investigating potential improvements to this approach.

IV.F.2. 2D Manifold Elasticity

As an alternative to the spring method, a manifold elasticity solver can be used to determine the domain boundary displacements. From Reddy,[30] the steady-state two-dimensional elasticity equations with no body forces are

$$\begin{aligned} \frac{\partial}{\partial x}\left(c_{11}\frac{\partial u}{\partial x} + c_{12}\frac{\partial v}{\partial y}\right) + \frac{\partial}{\partial y}\left[c_{66}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] = 0 \\ \frac{\partial}{\partial x}\left[c_{66}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] + \frac{\partial}{\partial y}\left(c_{12}\frac{\partial u}{\partial x} + c_{22}\frac{\partial v}{\partial y}\right) = 0 \end{aligned} \tag{24}$$

where the c coefficients are the material's elastic properties. For this work, the plain strain assumption is adopted, which gives

$$\begin{aligned} c_{11} &= \frac{E_1\left(1-\nu_{12}\right)}{1-\nu_{12}-2\nu_{12}\nu_{21}} \\ c_{12} &= \frac{\nu_{12}E_2}{1-\nu_{12}-2\nu_{12}\nu_{21}} \\ c_{22} &= \frac{E_2\left(1-\nu_{12}\nu_{21}\right)}{\left(1+\nu_{12}\right)\left(1-\nu_{12}-2\nu_{12}\nu_{21}\right)} \\ c_{66} &= G_{12} \end{aligned} \tag{25}$$

Next, an isotropic assumption is employed where

$$\begin{aligned} E_1 = E_2 = E &= 1\ \mathrm{Pa} \\ \nu_{12} = \nu_{21} = \nu &= 0 \\ G_{12} = G &= \frac{E}{2\left(\nu+1\right)} = \frac{1}{2}\ \mathrm{Pa} \end{aligned} \tag{26}$$

Converting Eq. 24 to a locally-tangential coordinate system to more generally describe a manifold gives

$$\begin{aligned} \frac{\partial}{\partial\tau_1}\left(c_{11}\frac{\partial u'}{\partial\tau_1}+c_{12}\frac{\partial v'}{\partial\tau_2}\right)+\frac{\partial}{\partial\tau_2}\left[c_{66}\left(\frac{\partial u'}{\partial\tau_2}+\frac{\partial v'}{\partial\tau_1}\right)\right] &= 0 \\ \frac{\partial}{\partial\tau_1}\left[c_{66}\left(\frac{\partial u'}{\partial\tau_2}+\frac{\partial v'}{\partial\tau_1}\right)\right]+\frac{\partial}{\partial\tau_2}\left(c_{12}\frac{\partial u'}{\partial\tau_1}+c_{22}\frac{\partial v'}{\partial\tau_2}\right) &= 0 \end{aligned} \tag{27}$$

In *CHAR*, these equations are solved with the finite element method, which requires the weak form of the governing equations given by

$$\begin{aligned} 0 &= \int_\Omega\left[\frac{\partial\nu_1}{\partial\tau_1}\left(c_{11}\frac{\partial u'}{\partial\tau_1}+c_{12}\frac{\partial v'}{\partial\tau_2}\right)+\frac{\partial\nu_1}{\partial\tau_2}c_{66}\left(\frac{\partial u'}{\partial\tau_2}+\frac{\partial v'}{\partial\tau_1}\right)\right]d\Omega \\ 0 &= \int_\Omega\left[\frac{\partial\nu_2}{\partial\tau_1}c_{66}\left(\frac{\partial u'}{\partial\tau_2}+\frac{\partial v'}{\partial\tau_1}\right)+\frac{\partial\nu_2}{\partial\tau_2}\left(c_{12}\frac{\partial u'}{\partial\tau_1}+c_{22}\frac{\partial v'}{\partial\tau_2}\right)\right]d\Omega \end{aligned} \tag{28}$$

where ν_1 and ν_2 represent the test functions. Eq. 28 is expressed with natural boundary conditions, which is acceptable for this method since the domain edge displacements are known for every node on the boundary of the manifold. Consequently, Dirichlet boundary conditions will be enforced.

Eq. 28 can be discretized by expanding the independent variables and test functions in terms of a finite-dimensional basis

$$\begin{aligned} \frac{\partial u'}{\partial\tau_1} &= \sum_{j=1}^{N_e} u'_j\frac{\partial\psi_j}{\partial\tau_1}, & \frac{\partial u'}{\partial\tau_2} &= \sum_{j=1}^{N_e} u'_j\frac{\partial\psi_j}{\partial\tau_2} \\ \frac{\partial v'}{\partial\tau_1} &= \sum_{j=1}^{N_e} v'_j\frac{\partial\psi_j}{\partial\tau_1}, & \frac{\partial v'}{\partial\tau_2} &= \sum_{j=1}^{N_e} v'_j\frac{\partial\psi_j}{\partial\tau_2} \\ \frac{\partial\nu_1}{\partial\tau_1} &= \sum_{i=1}^{N_e} \nu_{1_i}\frac{\partial\psi_i}{\partial\tau_1}, & \frac{\partial\nu_1}{\partial\tau_2} &= \sum_{i=1}^{N_e} \nu_{1_i}\frac{\partial\psi_i}{\partial\tau_2} \\ \frac{\partial\nu_2}{\partial\tau_1} &= \sum_{i=1}^{N_e} \nu_{2_i}\frac{\partial\psi_i}{\partial\tau_1}, & \frac{\partial\nu_2}{\partial\tau_2} &= \sum_{i=1}^{N_e} \nu_{2_i}\frac{\partial\psi_i}{\partial\tau_2} \end{aligned} \tag{29}$$

where i and j are nodal indices for the test and independent variable shape functions respectively, and N_e is the number of nodes on the element containing the point of interest (i.e. a quadrature point). The necessary transformations between the Cartesian and locally-tangential coordinate systems gives

$$\begin{aligned} u'_j &= \boldsymbol{u}_j\cdot\hat{\boldsymbol{\tau}}_{\boldsymbol{1}}\big|_j \\ v'_j &= \boldsymbol{u}_j\cdot\hat{\boldsymbol{\tau}}_{\boldsymbol{2}}\big|_j \end{aligned} \tag{30}$$

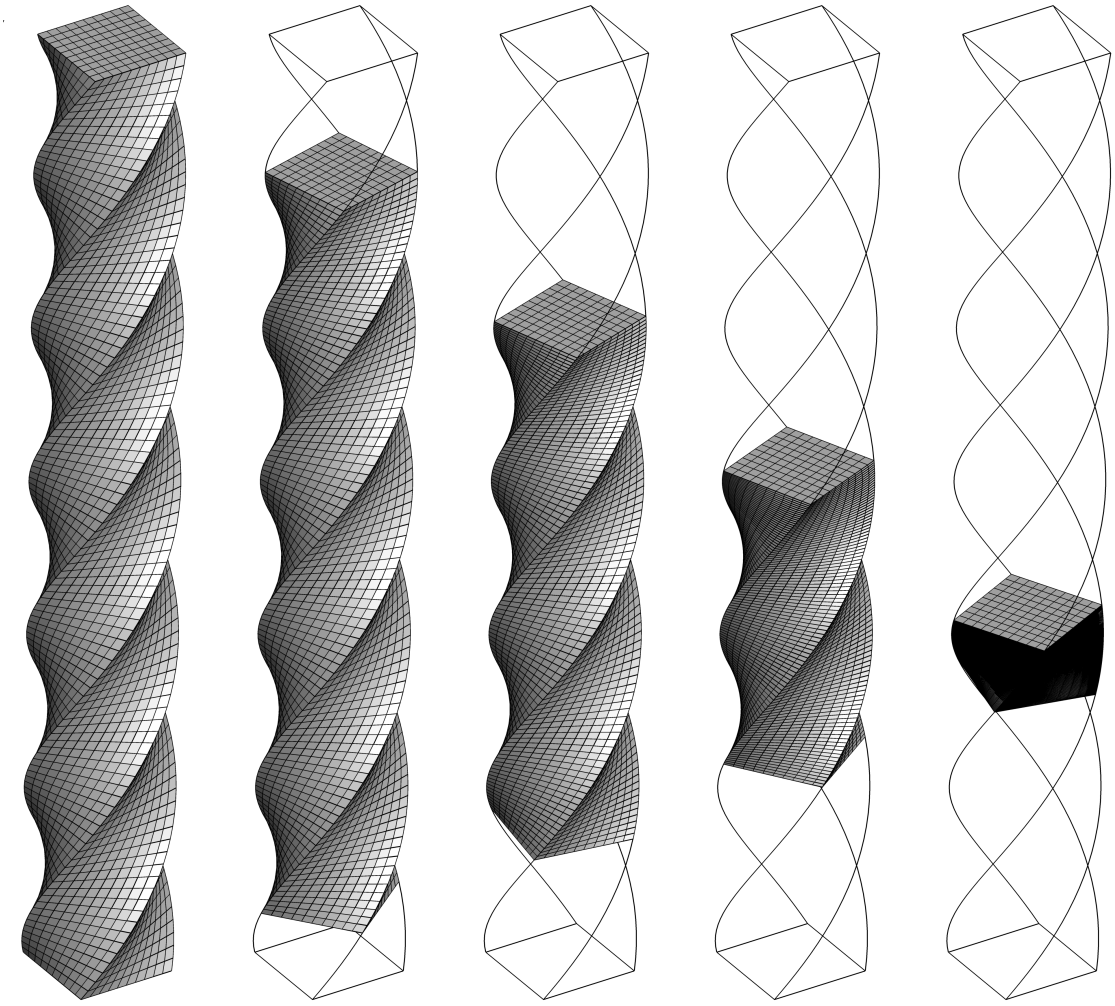
and

$$\begin{aligned} \frac{\partial \psi_i}{\partial \tau_1} &= \hat{\boldsymbol{\tau}}_{\mathbf{1}}(\boldsymbol{x}) \cdot \boldsymbol{\nabla} \psi_i(\boldsymbol{x}) \\ \frac{\partial \psi_i}{\partial \tau_2} &= \hat{\boldsymbol{\tau}}_{\mathbf{2}}(\boldsymbol{x}) \cdot \boldsymbol{\nabla} \psi_i(\boldsymbol{x}) \end{aligned} \tag{31}$$

The nodal residual equations are developed through an appropriate selection of basis functions. In *CHAR*, the compactly supported first order Lagrangian basis functions are used for both the test and independent variable shape functions. Trapezoidal quadrature is used for the spatial integration.

The final result is two equations with three unknowns per node. The shape constraint equation, Eq. 23, provides the third equation to close the system. The nonlinear system of equations is solved with Newton's method, and the Jacobians are determined with AD, which is again accomplished by utilizing Sacado.

After determining the domain boundary node displacements, the displacements are known for every node on the exterior of the domain. This information is provided to the mesh motion algorithm, and the domain interior node displacements are determined (step 8 in the overall process from Section IV). Figure 6 shows the solution for the twisted prism problem at five points in the solution process.



(a) Initial, Time 1 (b) Time 2 (c) Time 3 (d) Time 4 (e) Final, Time 5

Figure 6. Twisted prism solution using spring methodologies in Sections IV.E.1 and IV.F.1.

V. Demonstration Case: Boundary Layer Transition (BOLT) Flight Experiment

The proposed mesh motion boundary condition algorithm was applied to the Boundary Layer Transition (BOLT) flight experiment vehicle.[31] BOLT was selected purely for its challenging geometric configuration. Besides the external geometry, all details of this case are fabricated for demonstration purposes. This includes the materials, material properties, internal configuration, and aerothermal environment assumptions. This is not an attempt to simulate a flight.

V.A. Geometry, Materials, and Mesh

Figure 7 shows the BOLT geometry and the assumed material layout. The nose is a solid piece of carbon-carbon with properties taken from Ohlhorst.[32] The flank is a carbon phenolic shell wrapped around an aluminum substructure. The carbon phenolic properties are derived from the fictitious TACOT material.[33] All properties are the same as TACOT except the density of the non-decomposing component, which was increased for the current simulation. Beneath the aluminum structure, the flank is void, and no back end is modeled.

The computational domain was one quadrant of the vehicle, denoted by the magenta borders in Figure 7. Symmetry was assumed across the pitch (xy) and yaw (xz) planes. Figure 8 shows the hybrid mesh. The surface mesh is composed of quadrilaterals and triangles. The volume is composed of hexahedrons, tetrahedrons, prisms, and pyramids. The mesh at the interface between the nose and the flank is not point-matched on either the exterior surface (Figure 8(b)) or the pitch plane (Figure 8(d)). The surface normal spacing is significantly tighter for the carbon phenolic in order to capture the near-surface pressure gradient. The mesh has approximately 332,000 nodes and 448,000 elements, and the problem was run on four cores. Since this simulation is purely a demonstration of the mesh motion algorithm, no effort was put into performing a mesh or timestep resolution study.

V.B. Boundary and Initial Conditions

A summary of the boundary conditions is shown in Figure 9. This section focuses on a more detailed explanation of the two most complex boundary conditions: contact interfaces and external heating.

V.B.1. Thermal Contact Boundaries

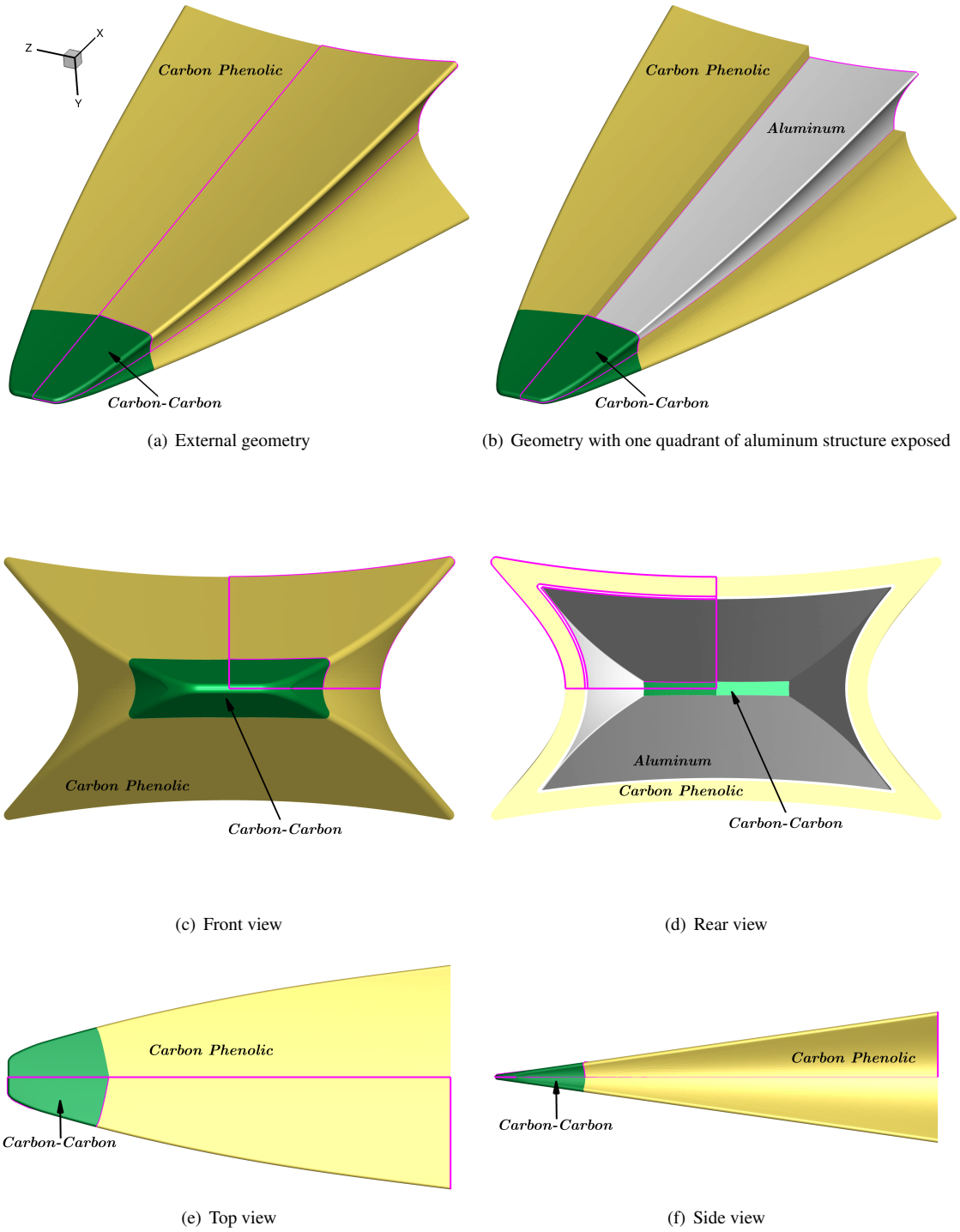
The contact interface between the nose and the flank utilized the perfect thermal contact model discussed by Amar et al.[8] and Salazar.[34] *CHAR* detects the thermal contact for every quadrature point on the contacting surfaces, and the implicit implementation of the method requires accommodation of additional degree-of-freedom dependencies. Since the boundary meshes on the contact interface are sliding throughout the simulation, the contact detection must be repeated every time the mesh moves. The element faces on opposing sides of the contact interface can vary greatly in size. This could either be due to the way the mesh was generated or due to the mesh deformation. Since the contact is only tracked at discrete locations (quadrature points), the element face size mismatch can result in missed contact detection between elements. To alleviate this, an adaptive quadrature scheme has been implemented. The idea is to refine the quadrature order for larger element faces on the contact surface to promote detection of contact with multiple smaller faces on the other side of the contact interface. The refinement algorithm attempts to match the number of quadrature points per unit area on each side of the contact interface.

V.B.2. External Heating Boundaries

The ablating external surfaces were subjected to a combined convective and radiative heating environment, and the surface was allowed to reradiate. The thermochemical ablation surface energy balance, discussed in more detail by Amar el al.,[8] is given by

$$\rho_e u_e C_H \left(h_r - h_w\right) + \alpha \dot{q}_{rad} - \sigma\varepsilon\left(T_w^4 - T_\infty^4\right) - \dot{m}_w h_w + \dot{m}_s h_s + \dot{m}_g h_g + \dot{q}_{cond_s} = 0 \tag{32}$$

where the red terms are provided by the user as part of the boundary condition definition. The recovery enthalpy, h_r, and the reradiation sink temperature, T_∞, are constant in space and time and are assumed to be $2 \times 10^7\,\mathrm{J/kg}$ and $300\,\mathrm{K}$ respectively. The initial film coefficient distribution is shown in Figure 10(a). The film coefficient decays exponentially in the axial direction and increases linearly along the span, which results in the peak heating point being off-centerline. In order to promote surface recession, the film coefficient has a minimum limit of $0.004\,\mathrm{kg/m^2 \cdot sec}$ and is increased linearly in time across the entire surface. Finally, the pressure distribution is also exponential in the axial



(a) External geometry

(b) Geometry with one quadrant of aluminum structure exposed

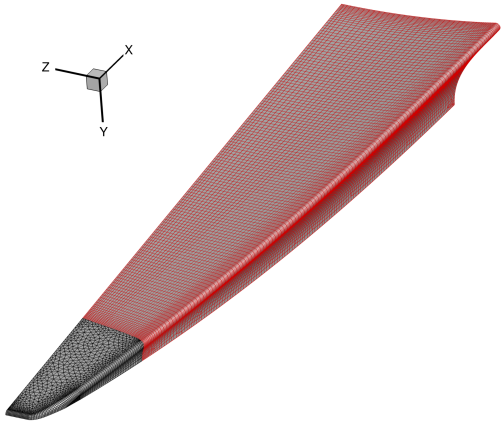
(c) Front view

(d) Rear view

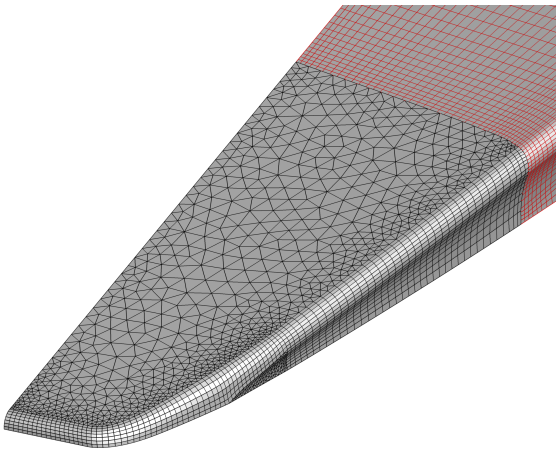
(e) Top view

(f) Side view

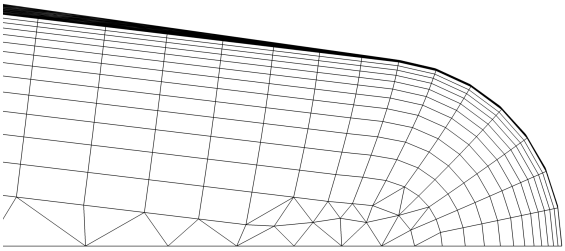
Figure 7. BOLT geometry and materials (magenta borders denote the computational domain).



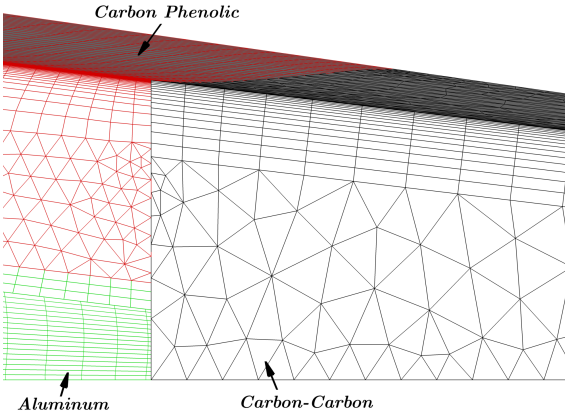
(a) External surface mesh



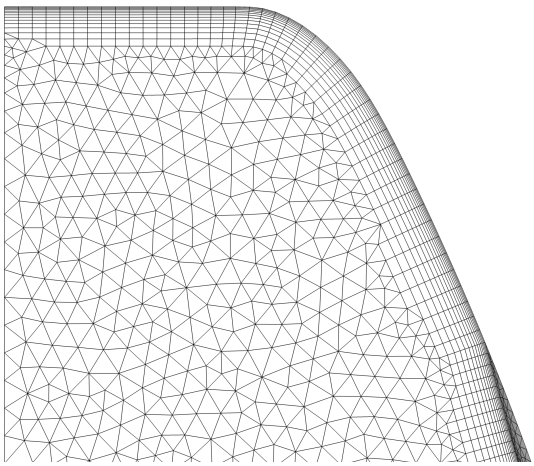
(b) External surface mesh on nose



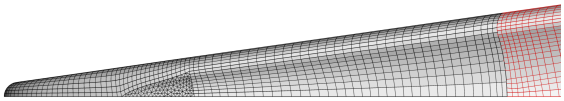
(c) Nosetip mesh on pitch (xy) plane



(d) Material interface mesh on pitch (xy) plane

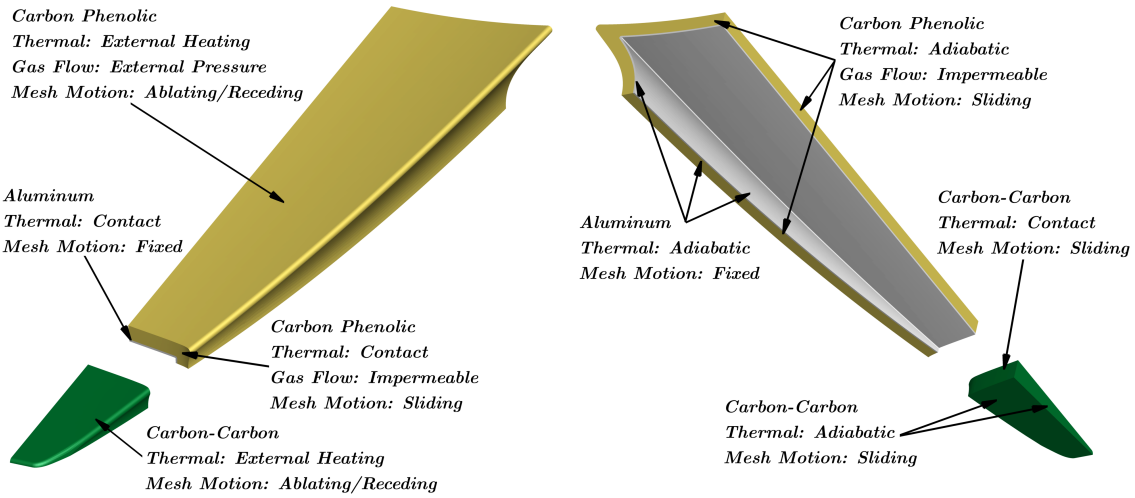


(e) Nosetip mesh on yaw (xz) plane



(f) Nosetip mesh on external surface

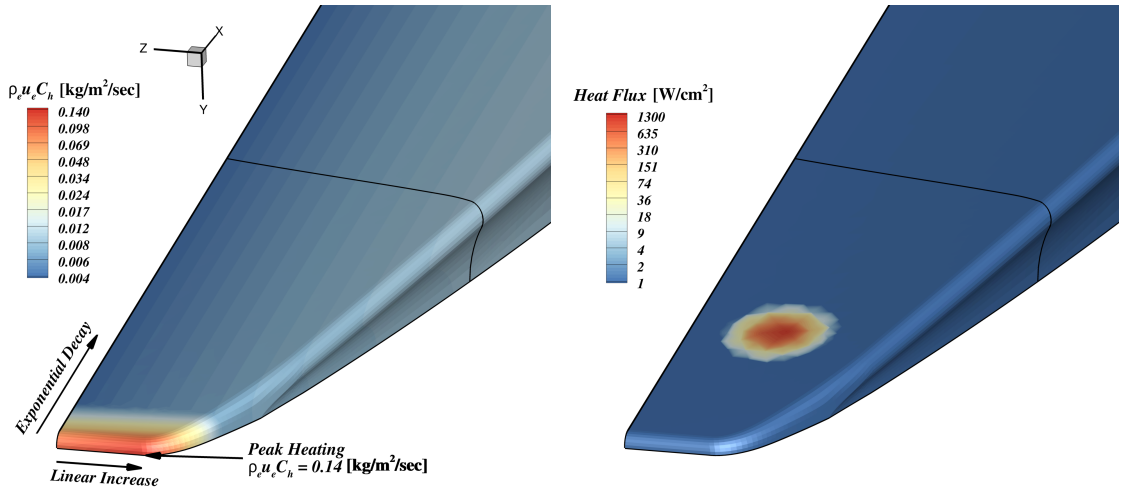
Figure 8. BOLT mesh (black: carbon-carbon, red: carbon phenolic, green: aluminum).



(a) Outer view (b) Inner view

Figure 9. BOLT boundary conditions (note: exploded view for clarity).

direction with a value of $100,000\,\mathrm{Pa}$ at the nose and a minimum limit of $1000\,\mathrm{Pa}$. The film coefficient and pressure distributions are defined relative to the instantaneous nose location. Therefore, the distribution shifts accordingly as the nosetip recedes.



(a) Initial film coefficient (b) Radiation

Figure 10. BOLT external heating distributions.

The radiation environment, shown in Figure 10(b), was designed to simulate a directed energy source. The pulse peaks at $1300\,\mathrm{W/cm^2}$ and has a Gaussian distribution in xz-space with one standard deviation of $0.5\,\mathrm{cm}$. The radiation pulse is fixed in space through all time and centered at $(x, z) = (0.078, -0.031)$. The mesh is not fine enough to resolve this distribution, but it is considered adequate for demonstration purposes.

V.B.3. Mesh Motion

All of the sliding boundaries noted in Figure 9 utilized the manifold elasticity method described in Section IV.F.2, and the receding boundaries utilized the spring analogy method described in Section IV.F.1. All moving edges utilized the uniformly contracting method described in Section IV.E.2.

V.B.4. Initial Conditions

All materials were initialized with a uniform temperature of $300\,\mathrm{K}$, and the carbon phenolic was initialized with a uniform pyrolysis gas pressure of $1000\,\mathrm{Pa}$.

V.C. Results

Figures 11-14 show the BOLT temperature field and geometry evolution at six snapshots in time throughout the solution process. Figures 12-14 introduce nodes marked with fiducial spheres to provide a reference for interpreting nodal motion. The fiducial spheres are consistent across the three figures.

Figure 11 shows a macro view of the solution. While both the carbon-carbon nose and carbon phenolic flank recess due to ablation, the most significant shape change occurs on the nose where the heating rate and pressure are the highest. By Time 6, nearly 50% of the length of the nose has ablated. It is evident where the directed energy radiation pulse creates a gouge in the surface, which is ultimately ablated away as the nosetip recedes.

Figure 12 focuses on the carbon-carbon nose. One feature of note is evidenced by the group of five cyan spheres that initially starts forward of the gouge. As the gouge develops and the nose ablates, the cyan spheres traverse into and through the gouge, and back onto the non-gouged surface aft of the directed energy pulse. This demonstrates how the method dissociates surface recession from lateral displacement. A second notable feature is the most forward and outboard line of white spheres. While they are initially near the corner of the nose, they end up being centered on the gouge on the leading edge of the nose. Of similar note is the patch of triangular faces that starts on the side of the nose then ends up wrapping around to the leading edge. Both of these features are a direct result of the uniformly contracting edge method described in Section IV.E.2. This method ensures that nodes will wrap around to the front face as the nose ablates by enforcing that the edge nodes maintain their initial relative spacing.

Figure 13 shows a cutaway view of the carbon-carbon nose that exposes the internal hybrid mesh in the vicinity of the gouge. Satisfactory mesh quality is maintained even under large deformations. Again, it is evident that the nodes traverse the gouge without shifting the position of the gouge itself. It should be noted that the calculation would benefit from additional mesh refinement for regions of the mesh that traverse the gouge. This would limit any geometric errors accumulated as the mesh deforms because significant mesh sliding along receding boundaries tends to smooth out edges. This is because the RBFs for receding boundaries are re-fitted at every time step based on a faceted surface defined by instantaneous locations of displaced nodes and cell centers. Consequently, poorly resolved curved surfaces and edges will tend to lose crispness through RBF re-fitting as the nodes slide along the surface. With the current algorithm, sliding boundaries do not have these smoothing errors since the RBFs derived from the initial geometry are always imposed as the shape constraints, thereby eliminating the need for RBF fitting at every time step.

Figure 14 provides a forward-looking view of the nose, which includes the contact boundary at the nose-to-flank material interface. It is evident that the two materials ablate at different rates thereby exposing some aft-facing carbon-carbon to the environment. In this simulation, the element-to-element contact was updated every time the mesh moved, and the exposed elements were treated as adiabatic. Further, the contact interface was curved as shown in Figure 7(e), and the mesh motion algorithm was able to maintain geometric integrity of the interface by appropriately sliding both sides.

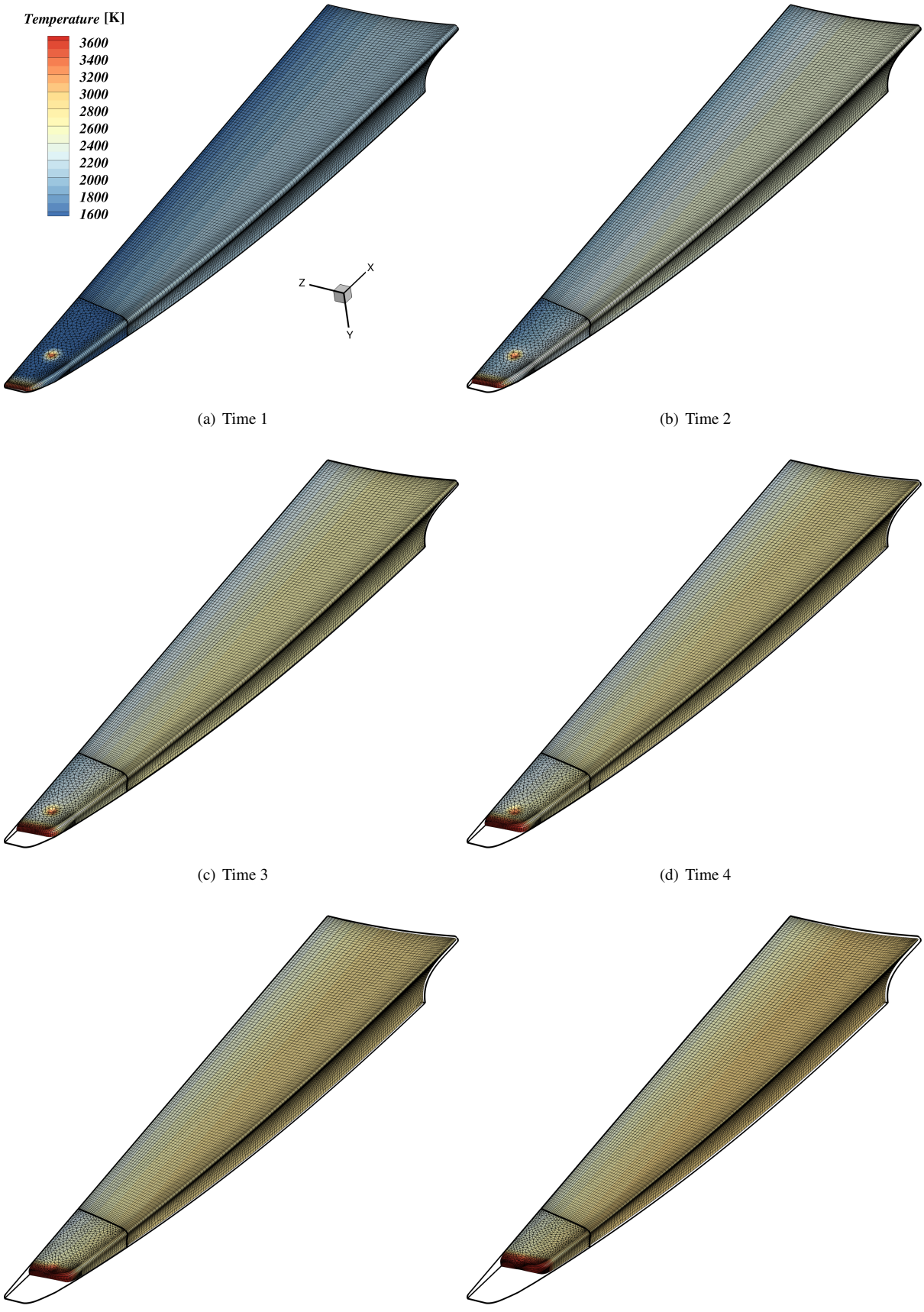
VI. Conclusions

A three-dimensional mesh motion boundary condition scheme has been developed that handles arbitrary sliding and receding surfaces through the use of RBFs to provide shape constraints. The method is independent of the scheme utilized to move the internal nodes of the mesh. The method has been demonstrated on the BOLT configuration, which showed satisfactory mesh quality could be maintained for large deformations.

The primary drawback of the current implementation is speed, but significant speed-up could be acheived by:

1. Using compactly supported basis functions for the RBFs, which will speed up both the fitting and evaluation processes
2. Implementing the RBF fast-fitting algorithm described by Beatson[20]
3. Implementing the RBF center reduction (greedy) algorithm described by Carr[17]

An analogous two-dimensional mesh motion boundary condition scheme has also been implemented in *CHAR*. The two-dimensional version follows the same concepts but is much simpler to implement since it does not require RBFs



(a) Time 1

(b) Time 2

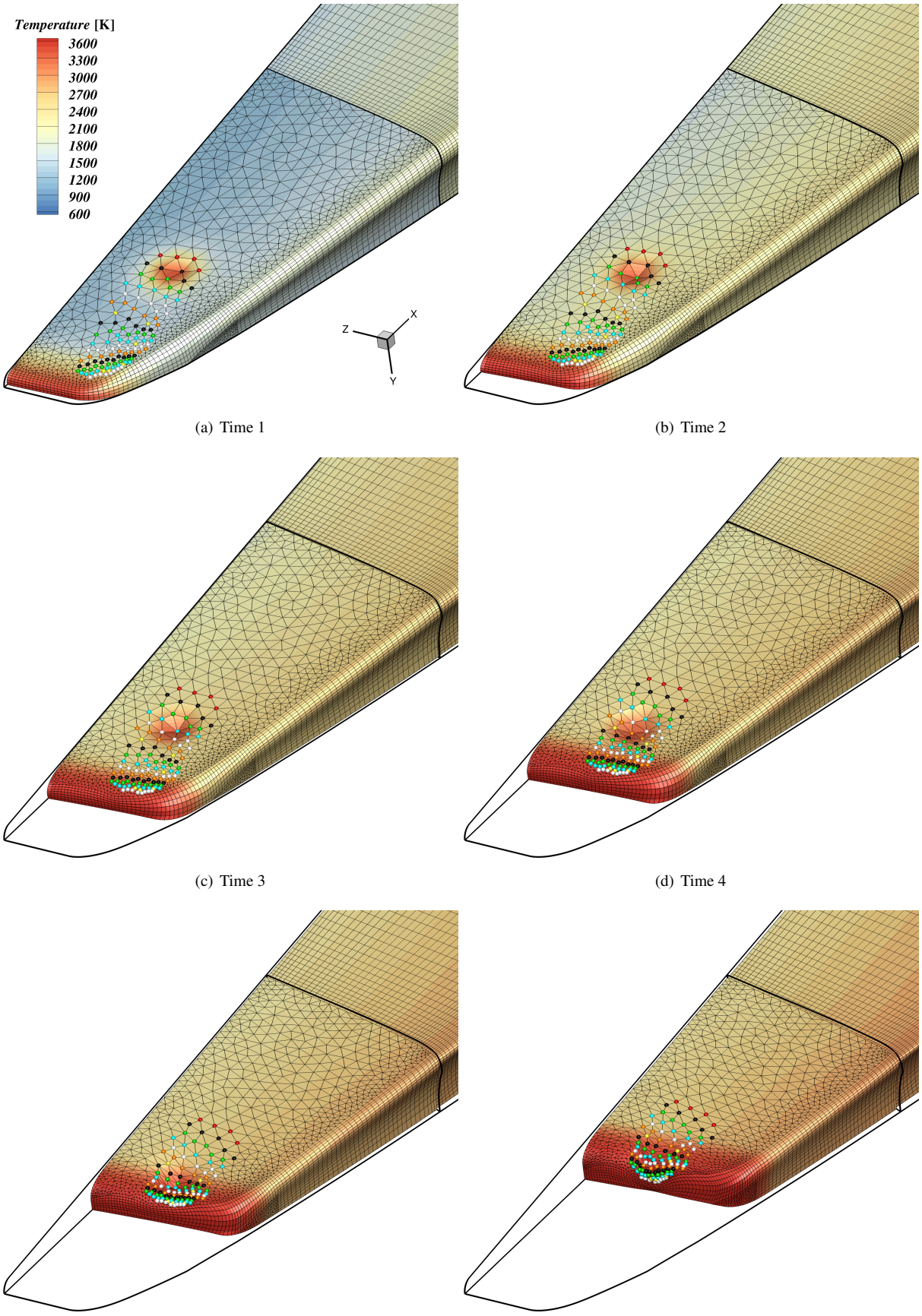
(c) Time 3

(d) Time 4

(e) Time 5

(f) Time 6

Figure 11. Macro view of BOLT solution (initial geometry defined by black outline).



(a) Time 1

(b) Time 2

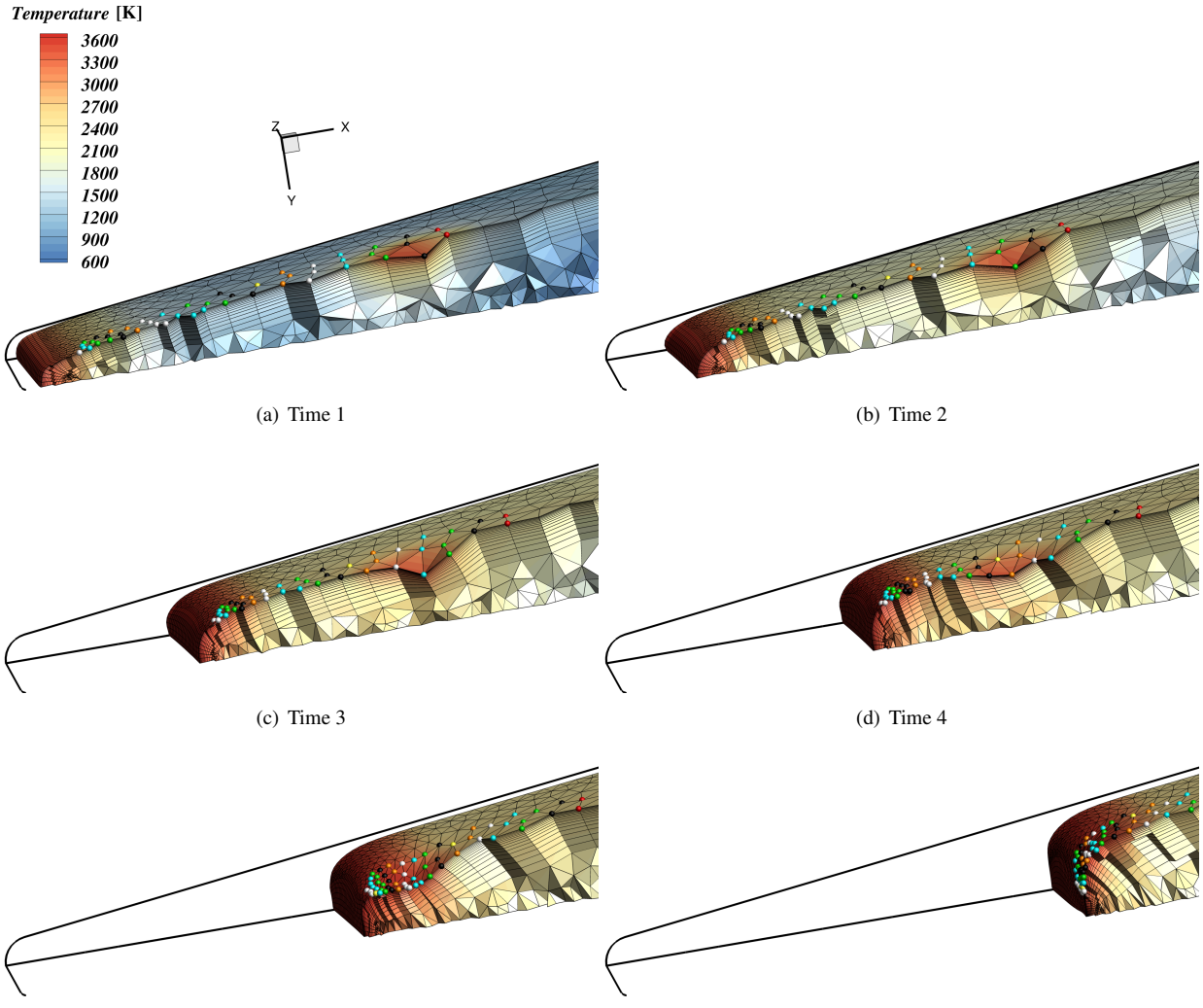
(c) Time 3

(d) Time 4

(e) Time 5

(f) Time 6

Figure 12. Nose view of BOLT solution (colored nodes introduced to visualize nodal motion).



(a) Time 1

(b) Time 2

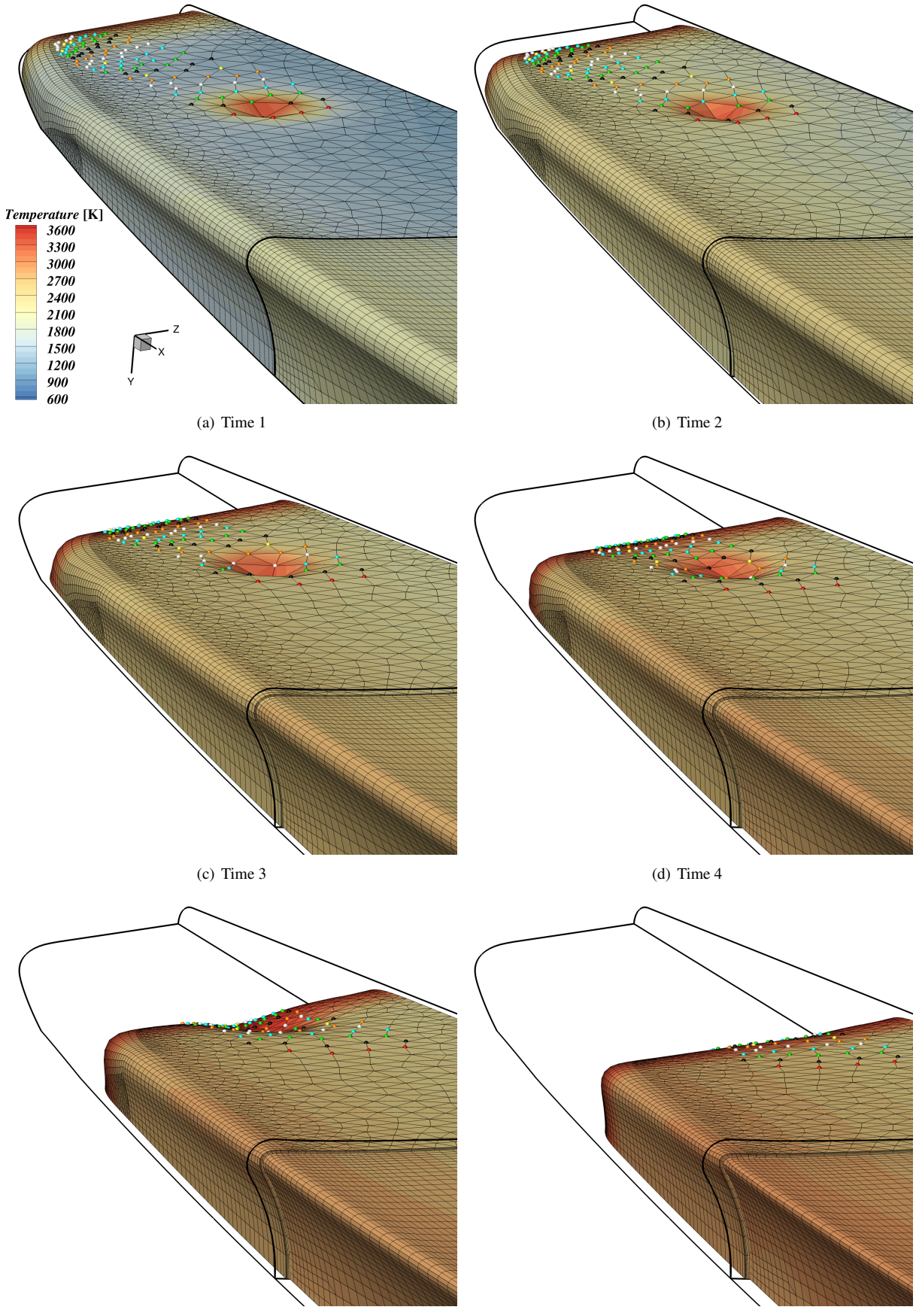
(c) Time 3

(d) Time 4

(e) Time 5

(f) Time 6

Figure 13. Cutaway nose view of BOLT solution.



(a) Time 1

(b) Time 2

(c) Time 3

(d) Time 4

(e) Time 5

(f) Time 6

Figure 14. Contact boundary view of BOLT solution.

to provide shape constraints. Instead the shape constraint is simply the piecewise linear definition of the boundary extracted directly from the mesh. Finally, while *CHAR* has adaptive mesh refinement (AMR) capability as discussed by Amar et al.,[8] the three-dimensional mesh motion boundary condition scheme discussed herein does not support AMR.

VII. Acknowledgments

The authors would like to thank:

- Dr. Sarah Popkin (Air Force Office of Scientific Research) for facilitating the use of the BOLT external surface geometry.
- Kyle Benalil (Corvid Technologies) for generating the BOLT CAD.
- Dr. Joseph Kotulski from Sandia National Laboratories (SNL) for the assistance with the Pliris package.
- Peter McCloud (ERC, Inc.) for the advice on mesh generation methods and quality improvements.
- Dr. Benjamin Kirk (NASA JSC) for the consulting and the development of the original mesh motion algorithms in *CHAR*.
- Dr. Alexandre Martin (University of Kentucky) for the courtesy review.
- Dr. Micah Howard (SNL), Dr. Derek Dinzl (SNL), Dr. Eric Stern (NASA ARC), and Dr. Alexandre Martin for the ongoing engagement in development of ablation models.

References

[1] Carl B. Moyer and Roald A. Rindal. An Analysis of the Coupled Chemically Reacting Boundary Layer and Charring Ablator, Part II: Finite Difference Solution for the In-Depth Response of Charring Materials Considering Surface Chemical and Energy Balances. Technical Report 66-7 Part II, Aerotherm, March 1967.

[2] B.F. Blackwell, R.W. Douglas, and H. Wolf. A User's Manual for the Sandia One-Dimensional Direct and Inverse Thermal (SODDIT) Code. Technical Report SAND 95-2478, Sandia National Laboratories, 1987.

[3] Y.-K. Chen Frank Milos. Ablation and thermal response program for spacecraft heatshield analysis. *Journal of Spacecraft and Rockets*, 36(3):475–483, 1999.

[4] Adam J. Amar, Bennie F. Blackwell, and Jack R. Edwards. One-Dimensional Ablation Using a Full Newton's Method and Finite Control Volume Procedure. *AIAA Journal of Thermophysics and Heat Transfer*, 22(71-82):1, 2006.

[5] Y.-K. Chen and F. S. Milos. Two-dimensional implicit thermal response and ablation program for charring materials. *Journal of Spacecraft and Rockets*, 38(4):473–481, 2001.

[6] Yih-Kanq Chen and Frank Milos. *Three-Dimensional Ablation and Thermal Response Simulation System*.

[7] R. Hogan, B. Blackwell, and R. Cochran. *Numerical solution of two-dimensional ablation problems using the finite control volume method with unstructured grids*.

[8] Adam J. Amar, Brandon Oliver, Benjamin Kirk, Giovanni Salazar, and Justin Droba. *Overview of the CHarring Ablator Response (CHAR) Code*. 2016.

[9] Fisher T. C. Hoemmen M. F. Dinzl D. J. Overfelt J. R. Bradley A. M. Kim K. Howard, M. Employing Multiple Levels of Parallelism for CFD at Large Scales on Next Generation High-Performance Computing Platforms. Technical Report SAND2018-6937C, Sandia National Laboratories, 2018.

[10] J. Lachaud and N. N. Mansour. Porous-material analysis toolbox. *Journal of Thermophysics and Heat Transfer*, 28(2):191–202, 2014.

[11] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in Physics*, 12(6):620–631, 1998.

[12] Justin M. Cooper, Olivia M. Schroeder, Haoyue Weng, and Alexandre Martin. *Implementation and Verification of a Surface Recession Module in a Finite Volume Ablation Solver*.

[13] Olivia M. Schroeder, Joseph Brock, Eric Stern, and Graham V. Candler. *A coupled ablation approach using Icarus and US3D*.

[14] Daron A. Isaac H. Heath Dewey Mark E. Ewing, David T. Walker. Heat Transfer and Erosion Analysis Program - Hero 4.1. Technical report, ATK, Alliant Techsystems, 2015.

[15] Mohamed Selim and Roy Koomullil. Mesh deformation approaches – a survey. *Journal of Physical Mathematics*, 7, 06 2016.
[16] Adam Amar, Nathan Calvert, and Benjamin Kirk. *Development and Verification of the Charring Ablating Thermal Protection Implicit System Solver*.
[17] J. Carr, Rick Beatson, J. Cherrie, T. Mitchell, W Fright, B. Mccallum, and T. Evans. Reconstruction and representation of 3d objects with radial basis functions. *ACM SIGGRAPH*, 09 2001.
[18] The Pliris Team. *Pliris Project Website* available at `https://trilinos.github.io/pliris.html`.
[19] The Trilinos Project Team. *The Trilinos Project Website* available at `https://trilinos.github.io`.
[20] R. Beatson, J. Cherrie, and C. T. Mouat. Fast fitting of radial basis functions: Methods based on preconditioned gmres iteration. *Advances in Computational Mathematics*, 11:253–270, 1999.
[21] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics*, 4:389–396, 1995.
[22] Benjamin S. Kirk, John W. Peterson, Roy H. Stogner, and Graham F. Carey. `libMesh`: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. *Engineering with Computers*, 22(3):237–254, 2006.
[23] Xiangmin Jiao. Face offsetting: A unified approach for explicit moving interfaces. *J. Comput. Phys*, 2006.
[24] R. K. Beatson, M. J. D. Powell, and A. M. Tan. Fast evaluation of polyharmonic splines in three dimensions. *IMA Journal of Numerical Analysis*, 27(3):427–450, 11 2006.
[25] Frederic J. Blom. Considerations on the spring analogy. *International Journal for Numerical Methods in Fluids*, 32(6):647–668, 2000.
[26] The Sacado Team. *Sacado Project Website* available at `https://trilinos.github.io/sacado.html`.
[27] B.F. Blackwell and R. E. Hogan. One-dimensional ablation using landau transformation and finite control volume procedure. *Journal of Thermophysics and Heat Transfer*, 8(2):282–287, 1994.
[28] D.S. Lopes, M.T. Silva, and J.A. Ambrósio. Tangent vectors to a 3-d surface normal: A geometric tool to find orthogonal vectors based on the householder transformation. *Computer-Aided Design*, 45(3):683–694, 2013.
[29] George Markou, Zacharias Mouroutis, Dimos Charmpis, and Papadrakakis Manolis. The orto-semi-torsional (ost) spring analogy method for 3d mesh moving boundary problems. *Computer Methods in Applied Mechanics and Engineering*, 196, 01 2007.
[30] J.N. Reddy. *Introduction to the Finite Element Method*. McGraw-Hill Education, 1993.
[31] Bradley M. Wheaton, Dennis C. Berridge, Thomas Wolf, Daniel Araya, Ryan Stevens, Brian E. McGrath, Brian Kemp, and David Adamczak. *Final Design of the Boundary Layer Transition (BOLT) Flight Experiment*.
[32] Craig Ohlhorst, Wallace L. Vaughn, Philip O. Ransone, and Hwa tsu Tsou. Thermal conductivity database of various structural carbon-carbon composite materials, 1997.
[33] Jean R Lachaud, Tom van Eekelen, Daniele Bianchi, and Alexandre Martin. Tacot v3.0. In *Ablation Workshop: Code Comparison*, 2014.
[34] Giovanni Salazar and Adam J. Amar. *Contact Boundary Conditions in the CHarring Ablator Response (CHAR) Code*. 2021.

A. Derivatives of the RBF Distance Function

Derivatives of the RBF distance function, $s(\boldsymbol{x})$, are necessary when defining surface normals and Jacobians for various parts of the algorithm. Eq. (9) can be rewritten as

$$s(\boldsymbol{x}) = c_1 + c_2 x + c_3 y + c_4 z + \sum_{i=1}^{N} \lambda_i \sqrt{g_i}$$

where

$$g_i = (x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2$$

The first derivatives are

$$\frac{\partial s}{\partial x} = c_2 + \sum_{i=1}^{N} \lambda_i \left(x - x_i\right) g_i^{-\frac{1}{2}}$$
$$\frac{\partial s}{\partial y} = c_3 + \sum_{i=1}^{N} \lambda_i \left(y - y_i\right) g_i^{-\frac{1}{2}}$$
$$\frac{\partial s}{\partial z} = c_4 + \sum_{i=1}^{N} \lambda_i \left(z - z_i\right) g_i^{-\frac{1}{2}}$$

The second derivatives are

$$\frac{\partial^2 s}{\partial x^2} = \sum_{i=1}^{N} \lambda_i \left[g_i^{-\frac{1}{2}} - g_i^{-\frac{3}{2}} \left(x - x_i\right)^2\right]$$
$$\frac{\partial^2 s}{\partial y^2} = \sum_{i=1}^{N} \lambda_i \left[g_i^{-\frac{1}{2}} - g_i^{-\frac{3}{2}} \left(y - y_i\right)^2\right]$$
$$\frac{\partial^2 s}{\partial z^2} = \sum_{i=1}^{N} \lambda_i \left[g_i^{-\frac{1}{2}} - g_i^{-\frac{3}{2}} \left(z - z_i\right)^2\right]$$
$$\frac{\partial^2 s}{\partial x \partial y} = \frac{\partial^2 s}{\partial y \partial x} = -\sum_{i=1}^{N} \lambda_i g_i^{-\frac{3}{2}} \left(x - x_i\right)\left(y - y_i\right)$$
$$\frac{\partial^2 s}{\partial x \partial z} = \frac{\partial^2 s}{\partial z \partial x} = -\sum_{i=1}^{N} \lambda_i g_i^{-\frac{3}{2}} \left(x - x_i\right)\left(z - z_i\right)$$
$$\frac{\partial^2 s}{\partial y \partial z} = \frac{\partial^2 s}{\partial z \partial y} = -\sum_{i=1}^{N} \lambda_i g_i^{-\frac{3}{2}} \left(y - y_i\right)\left(z - z_i\right)$$

B. Jacobians for Domain Corner Node Displacement Algorithm

The domain corner node displacement algorithm described in Section IV.D is solved fully-implicit using Newton's method with analytical Jacobians, which are provided in this appendix. The residuals in Eqs. (14) and (15), written out in expanded form, are

$$R_1 = 2\left(x^{n+1} - x^n\right) + \sum_{k=1}^{K} \Lambda_k \frac{\partial s_k}{\partial x}$$
$$R_2 = 2\left(y^{n+1} - y^n\right) + \sum_{k=1}^{K} \Lambda_k \frac{\partial s_k}{\partial y}$$
$$R_3 = 2\left(z^{n+1} - z^n\right) + \sum_{k=1}^{K} \Lambda_k \frac{\partial s_k}{\partial z}$$
$$R_{3+k} = s_k \text{ for } k = 1, ..., K$$

The Jacobians are

$$\frac{\partial R_1}{\partial x} = 2 + \sum_{k=1}^{K} \Lambda_k \frac{\partial^2 s_k}{\partial x^2} \qquad \frac{\partial R_1}{\partial y} = \sum_{k=1}^{K} \Lambda_k \frac{\partial^2 s_k}{\partial x \partial y} \qquad \frac{\partial R_1}{\partial z} = \sum_{k=1}^{K} \Lambda_k \frac{\partial^2 s_k}{\partial x \partial z} \qquad \frac{\partial R_1}{\partial \Lambda_k} = \frac{\partial s_k}{\partial x}$$

$$\frac{\partial R_2}{\partial x} = \sum_{k=1}^{K} \Lambda_k \frac{\partial^2 s_k}{\partial y \partial x} \qquad \frac{\partial R_2}{\partial y} = 2 + \sum_{k=1}^{K} \Lambda_k \frac{\partial^2 s_k}{\partial y^2} \qquad \frac{\partial R_2}{\partial z} = \sum_{k=1}^{K} \Lambda_k \frac{\partial^2 s_k}{\partial y \partial z} \qquad \frac{\partial R_2}{\partial \Lambda_k} = \frac{\partial s_k}{\partial y}$$

$$\frac{\partial R_3}{\partial x} = \sum_{k=1}^{K} \Lambda_k \frac{\partial^2 s_k}{\partial z \partial x} \qquad \frac{\partial R_3}{\partial y} = \sum_{k=1}^{K} \Lambda_k \frac{\partial^2 s_k}{\partial z \partial y} \qquad \frac{\partial R_3}{\partial z} = 2 + \sum_{k=1}^{K} \Lambda_k \frac{\partial^2 s_k}{\partial z^2} \qquad \frac{\partial R_3}{\partial \Lambda_k} = \frac{\partial s_k}{\partial z}$$

$$\frac{\partial R_{3+k}}{\partial x} = \frac{\partial s_k}{\partial x} \qquad \frac{\partial R_{3+k}}{\partial y} = \frac{\partial s_k}{\partial y} \qquad \frac{\partial R_{3+k}}{\partial z} = \frac{\partial s_k}{\partial z} \qquad \frac{\partial R_{3+k}}{\partial \Lambda_k} = 0$$

and the derivatives of the RBF distance function, s, are given in Appendix A.