CAD Mesh Intersection with 3D Finite Element Meshes

Jack Benner, Mentors: Nathaniel Morgan, PhD; Sarah Brown, PhD | E-2 Modern Manufacturing Methodologies

Abstract

This project presents a parallel C++ tool designed to calculate volume fractions of STL meshes intersected with finite element meshes. The work exports the results as VTK files for visualization. The workflow includes reading triangular STL geometry files, constructing a finite element mesh, and performing ray-triangle intersection tests to estimate how much of each 3D surface lies in an element of the mesh.

To improve performance, the code uses MATAR for performance and parallelism by interfacing with Kokkos. The resulting fields on the finite element mesh are color-coded by volume fraction and can be visualized using standard tools such as ParaView. This code is intended for integration into the Fierro code, a multiphysics simulation platform, where it will contribute to setting up finite element simulations. Future work includes improving sampling accuracy, testing performance on GPUs, and expanding to handle more complex mesh types.

Motivation

STL files, composed of surface triangle meshes, are a common format for representing 3D geometry. To simplify and expand simulation options with the Fierro code, it is advantageous to paint parts on finite element meshes in place of using conformal meshes. This project aims to intersect STL files with the finite element mesh and compute the volume fraction of geometry in each mesh element, a crucial step in pre-processing for multiphysics simulations. The final product is a reproducible, parallelized integration routine that can efficiently process arbitrary STL geometries and export VTK data fields for visualization.

Objectives

- Develop a discretization method using deterministic Monte Carlo sampling, normalize and center meshes of arbitrary STL geometry in a defined bounding box.
- Allow user control over bounding box size and mesh resolution.
- Compute mesh volume fractions via uniform point sampling.
- Visualize results with ParaView
- Parallelize the algorithm to run on a GPU with MATAR to leverage the power of modern high-performance computing.
- Validate results using analytical shapes (sphere, cone, cube, etc.).
- Prepare for integration into production Fierro code.

Methods

Transforming from Cartesian to Isoparametric coordinates $(x, y, z) \rightarrow (\xi, \eta, \zeta)$:

We are transforming a point in cartesian space on the STL geometry to a point in isoparametric space in the mesh, we will utilize Newton-Raphson inversion to accomplish this.

$\vec{x}_{target} = \sum_{p=0}^{7} \phi_p(\zeta, \xi, \eta)\, \vec{x}_p$ (for a hexahedron) where ϕ_p is a shape function for node p

$\phi_p(\zeta, \xi, \eta) := \frac{1}{8}(1 + \xi s_x)(1 + \eta s_y)(1 + \zeta s_z)$ s.t. $s_x, s_y, s_z \in \{-1, 1\}$

$\vec{x}_p$ are node positions, $x_p = (x_p, y_p, z_p) \forall p \in \{0, 1, ..., 7\}$

Using Newton-Raphson, we iteratively solve for x, y and z with the following equations:

$$x = \sum_{p=0}^{7} \phi_p(\zeta, \xi, \eta)\, x_p \quad y = \sum_{p=0}^{7} \phi_p(\zeta, \xi, \eta)\, y_p \quad z = \sum_{p=0}^{7} \phi_p(\zeta, \xi, \eta)\, z_p$$

Let residual $\vec{F} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ be given by $\vec{F}(\xi, \eta, \zeta) = \sum_{p=0}^{7} \phi_p(\xi, \eta, \zeta)\vec{x}_p - \vec{x}_{target}$

We seek $\vec{F}(\xi, \eta, \zeta) = 0$. The Jacobian of $\vec{F}$ is $\mathbf{J} = \frac{\partial \vec{x}}{\partial(\zeta, \xi, \eta)} = \begin{bmatrix} \frac{\partial x}{\partial \zeta} & \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \zeta} & \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \\ \frac{\partial z}{\partial \zeta} & \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} \end{bmatrix}$

We apply Newton-Raphson iteration to find $(\xi, \eta, \zeta) : \vec{x}(\xi, \eta, \zeta) = \vec{x}_{target}$.

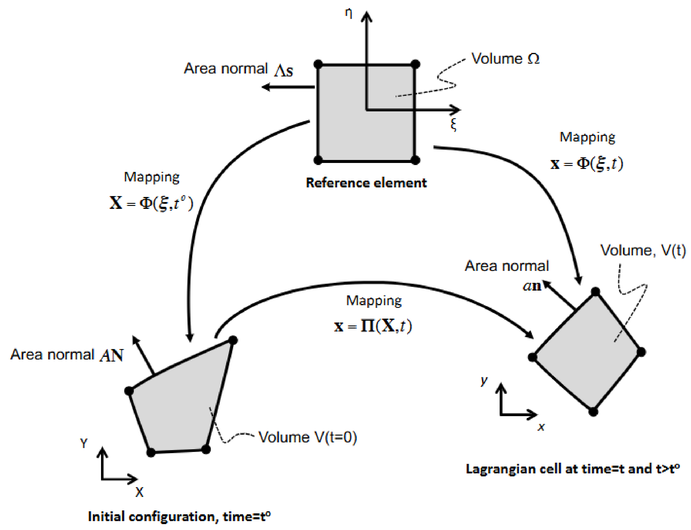
Newton-Raphson Iteration: $\begin{bmatrix} \zeta \\ \xi \\ \eta \end{bmatrix}^{(n+1)} = \begin{bmatrix} \zeta \\ \xi \\ \eta \end{bmatrix}^{(n)} - \mathbf{J}^{-1} \vec{F}^{(n)}$

We stop when $||\vec{F}(\xi, \eta, \zeta)|| <$ Tolerance, where the tolerance is defined by the user. The next step is to calculate how much of the element ($\hat{\Omega} \subset [-1, 1]$) is inside the STL geometry, this is the volume fraction that we seek.

Local Volume Fraction:

$$\frac{V_{\text{in}}}{V_{\text{ref}}} = \int_{\hat{\Omega}} H(\xi, \eta, \zeta)\, d\xi\, d\eta\, d\zeta \quad \text{where} \quad H(\xi, \eta, \zeta) = \begin{cases} 1 & \text{if the mapped point lies inside the STL geometry} \\ 0 & \text{otherwise} \end{cases}$$

The volume fraction is evaluated using integration over each element. Although the volume fraction is framed as an analytic integral, here it is approximated by discrete sampling, allowing for simpler integration of discontinuous fields. The integration is done using the Moller-Trumbore (1997) algorithm.



Results

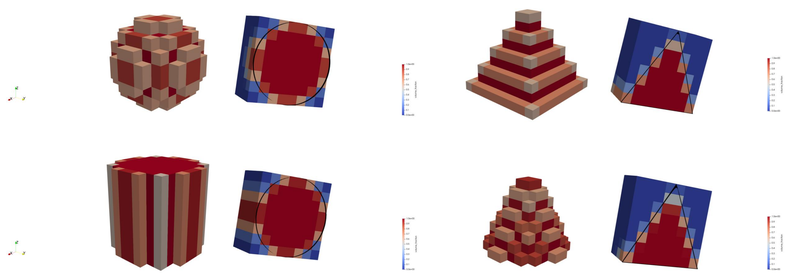


Chart 1. Discretized Geometries viewed in ParaView

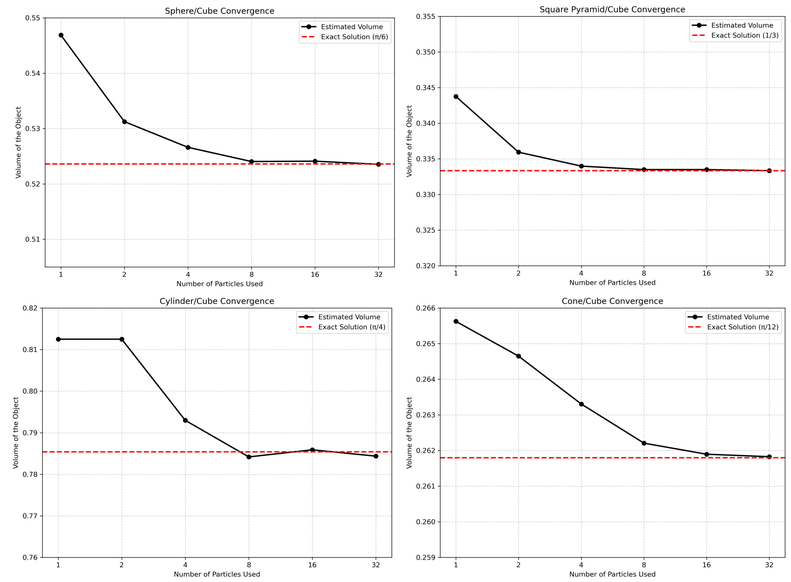


Chart 2. Convergence to the correct volume fraction of different geometries

References

Los Alamos National Laboratory. *MATAR: Memory Access Tracking and Reuse.* GitHub repository. https://github.com/lanl/MATAR

Los Alamos National Laboratory. *Fierro: A Massively Parallel Soild Mechanics Code.* GitHub repository. https://github.com/lanl/Fierro

Moller, T. and Trumbore, B. (1997). "Fast, Minimum Storage Ray/Triangle Intersection." https://www.graphics.cornell.edu/pubs/1997/MT97.pdf

Acknowledgements

We thank the LDRD program for supporting this work.