Implementing Topological Mesh Interpolation for PETSc

Harvey Kwong
harveykw@buffalo.edu

Advised by Dr. Matthew Knepley
knepley@buffalo.edu

December 15, 2025

Abstract

Mesh based discretizations are fundamental to scientific computing, particularly for the numerical solution of partial differential equations (PDEs). Many mesh transformations—such as refinement, coarsening, and discretizations require access to the full topological structure of a mesh. This includes intermediate cell strata such as edges and faces. However, in many practical settings, only top dimensional cells and vertices are explicitly available, with intermediate entities implicitly defined.

This project presents an approach for interpolating the complete topology of a mesh given only its cells and vertices. Although the interpolation is designed as a standalone transformation, it is implemented as a feature within PETSc, a widely used scientific computing library. The approach reconstructs all intermediate cell types in a manner that is easily extensible to multiple polytope shapes, and incorporates deduplication to ensure topological consistency. While the current implementation does not yet handle orientation, the framework is designed to support it, and the required methodology is well understood. This work demonstrates that full topology interpolation can be modularized within an existing scientific framework, enabling reuse, extensibility, and support for downstream mesh transformations.

1 Introduction

1.1 Importance of Topological Interpolation

Meshes are the primary data structure underlying a wide range of scientific and engineering simulations, particularly those involving the numerical solution of partial differential equations. Applications include fluid dynamics, structural mechanics, electromagnetics, and climate modeling.

In practice, meshes are rarely static. Mesh generation pipelines commonly begin from minimal cell complexes and progressively introduce additional structure to support discretization, numerical stability, or adaptivity requirements. Topological interpolation provides a mechanism for enriching a mesh by explicitly constructing lower dimensional entities, such as edges and faces.

Many mesh transformations require explicit access to all topological strata. Operations such as mesh refinement, coarsening, and finite element methods depend on the existence of intermediate entities, including edges and faces, as well as well defined connectivity between cells. Without these entities, a large class of transformations cannot be expressed or applied.

Workflows that rely heavily on mesh transformation therefore naturally depend on topological interpolation as a foundational operation. Interpolation enables transformations to be defined locally, applied uniformly across cell types, and composed within larger pipelines. Prior work has shown that mesh generation and manipulation play a central role in large scale multiphysics simulations, where complex discretizations and repeated mesh transformations are commonplace [1]. As a result, topology interpolation is frequently performed implicitly as an integral step in mesh generation and transformation frameworks.

1.2 Limitations of Existing Approaches

In existing software systems, topology interpolation is typically embedded within large, monolithic mesh generation libraries. These implementations are tightly coupled to specific workflows and data structures, making them difficult to reuse independently or extend to support additional cell types.

To the best of our knowledge, there exists no open source program that implements topological interpolation as a standalone operation.

1.3 Contributions

This project addresses these limitations by implementing full topology interpolation as a logically standalone transformation within PETSc. The main contributions are:

- A general algorithm for reconstructing full mesh topology from cells and vertices
- Easily extensible to multiple polytope types
- A deduplication mechanism ensuring consistent representation of shared entities
- Integration with PETSc's mesh infrastructure for immediate practical use

The current implementation focuses on topological reconstruction and does not yet compute orientation information. Orientation handling is left as future work.

2 Problem Definition

Given a mesh representation consisting of:

- A set of vertices
- A set of top dimensional cells defined by vertex connectivity

The objective is to reconstruct the complete mesh topology. This includes all implied lower dimensional entities (such as edges and faces), as well as correct adjacency relationships between entities. The resulting mesh must be consistent, deduplicated, and suitable for subsequent transformations within PETSc, including refinement and any transformation that requires the existence of intermediate entities within the mesh topology.

3 Interpolation Algorithm and PETSc Implementation

3.1 Overview

Although our interpolation is designed as a standalone transformation, it is implemented within PETSc to leverage existing mesh data structures, validation tools, and downstream operations. The algorithm applies cell specific transformation rules, deduplicates shared entities, and assigns contiguous index ranges to each topological stratum, producing a mesh representation that conforms to PETSc's internal conventions and can be used directly by other PETSc mesh transformations.

3.2 Cell Transformation Rules

Each cell type is associated with a transformation rule that specifies:

- Which lower dimensional cell types it produces
- How many entities of each type are produced

These transformation rules are queried through PETSc's existing transformation interface. By expressing the interpolation logic in terms of generic transformation rules, the algorithm remains easily extensible to additional polytope types. The current implementation serves as a proof of concept and is limited to meshes composed of triangular cells. This restriction is not fundamental to the algorithm. Extending support to additional cell shapes or higher dimensional polytopes requires only the specification of the corresponding transformation rules for the new shape. In addition, the keys used by the internal hash tables for deduplication must be extended to accommodate the larger cones associated with higher dimensional entities. No other structural changes to the interpolation algorithm are required.

Table 1: Examples of transformation rules used for topology interpolation. Each cell produces itself and the entities in its immediate cone (its boundary).

Producer cell type	**Dim**	**Produces (types)**	**Counts per cell**
Segment (edge)	1	Segment, Vertex	1 segment (self), 2 vertices
Triangle (face)	2	Triangle, Segment	1 triangle (self), 3 segments
Tetrahedron (cell)	3	Tetrahedron, Triangle	1 tetrahedron (self), 4 triangles

3.3 PETSc Numbering Convention

PETSc requires all mesh entities to be numbered consecutively according to a fixed ordering based on topological dimension. This ordering is assumed throughout the PETSc API and must be respected by any mesh produced through topology interpolation[2].

In three dimensions, entities are numbered in decreasing order of dimension: first top dimensional cells, followed by vertices, then faces, and finally edges. In two dimensions, the convention is simplified so that faces are numbered first, followed by vertices, and then edges. In all cases, entities of the same type occupy a contiguous range of indices.

As an illustrative two dimensional example, consider a triangle doublet mesh containing two faces, four vertices, and five edges. The corresponding numbering is

$$\begin{aligned} &f_0 \mapsto 0, \quad f_1 \mapsto 1, \\ &v_0 \mapsto 2, \ldots, v_3 \mapsto 5, \\ &e_0 \mapsto 6, \ldots, e_4 \mapsto 10 \end{aligned}$$

The interpolation algorithm constructs prefix offsets for each cell type to ensure that all newly created entities are assigned indices consistent with this convention. Adhering to this ordering is essential for compatibility with PETSc's internal data structures and for the correctness of subsequent mesh transformations.

```
 5--10---4--14---7
 |       |       |
11   0   9   1  13
 |       |       |
 2---8---3--12---6
```

Figure 1: PETSc compliant numbering for the full topology for two quadrilaterals sharing a face. ASCII source:[3]

3.4 Offset Table

A central data structure in the implementation is the *offset table*. The offset table records, for each pair of cell types, the starting index of entities of one type produced by another.

Formally, the offset table entry

$$\texttt{off}[ct, ctNew]$$

stores the starting index of entities of type `ctNew` produced by cells of type `ct`, where the numbering is ordered according to PETSc's internal cell type ordering. The offset table enables efficient mapping from a produced entity to the original cell that generated it. This is essential for computing the final numbering of all cells within the produced mesh. Entries are set to -1 for combinations where a given cell type does not produce entities of the target type.

3.5 Deduplication of Intermediate Entities

When interpolating the mesh topology, following our cell transformation rules, intermediate entities such as edges or faces may be produced multiple times by different cells. Ensuring that these entities are represented uniquely is essential to maintain topological consistency. The implementation employs a two level deduplication strategy based on hash tables to address this requirement.

For each coordinate $(ct, ctNew)$ in the offset table, a *local hash table* is used to deduplicate entities of type `ctNew` produced by all producer types that precede `ct` in PETSc's cell type ordering. As the algorithm traverses the producer types in Plex order, this local hash table accumulates all unique entities of the target type `ctNew` produced by earlier producers. The size of this table therefore corresponds to the number of distinct entities of type `ctNew` that have already been produced before processing cells of type `ct`. This count is used to compute the starting index stored in the offset table, which identifies where the entities produced by `ct` begin in the global numbering.

In parallel, a *global hash table* is maintained throughout the interpolation process. This table deduplicates entities of each type across all producers and assigns each unique entity a global index within its topological stratum, as well as the numbering of the source entity that produced the current entity. The global index corresponds to the position of the entity among all entities of that type in the interpolated mesh, such as the fourth triangle or the tenth edge overall. The global hash table enables consistent indexing and supports reverse mappings from newly created entities back to their producing cells.

Together, these two hash tables separate local offset computation from global entity numbering. This design allows the algorithm to compute accurate starting offsets for each producer while simultaneously constructing a globally consistent and deduplicated representation of the mesh topology.

3.6 Indexing and Final Numbering

After all mesh entities have been generated and deduplicated, the final step of the interpolation algorithm is to assign a global numbering to each entity that conforms to PETSc's required ordering. This numbering is constructed using prefix offsets stored in the arrays `ctStart` and `ctStartNew`.

For each cell type, `ctStart` stores the starting index of that type in the original mesh, while `ctStartNew` stores the starting index of the same type in the interpolated mesh. These arrays are computed by first counting the number of entities of each type and then performing a prefix sum in PETSc's prescribed cell-type order. As a result, all entities of a given type occupy a contiguous range of indices.

The final global index of a newly created entity is obtained by adding its local index within its type to the corresponding entry in `ctStartNew`. For example, if `ctStartNew[segment] = 6`, then the first segment in the interpolated mesh is assigned index 6, the second index 7, and so on. This ensures that all faces, vertices, and edges are numbered consecutively and in the correct order.

The offset table constructed during interpolation provides the link between producers and produced entities. For a given producer type `ct` and produced type `ctNew`, the entry `off[ct, ctNew]` specifies the starting local index of entities of type `ctNew` produced by `ct`. Combined with `ctStartNew`, this allows the algorithm to map a produced entity to its global index and to recover the original producing cell when required.

By deriving the final numbering entirely from `ctStart`, `ctStartNew`, and the offset table, the algorithm guarantees consistency with PETSc's numbering conventions.

3.7 Integration with PETSc

The interpolation algorithm is implemented within PETSc's mesh infrastructure. This allows it to:

- Reuse PETSc's existing cell transformation interfaces
- Integrate seamlessly with downstream mesh operations such as refinement
- Benefit from PETSc's validation and debugging tools

Although embedded in PETSc, the interpolation logic itself is modular and could be extracted or extended independently.

3.8 Orientation Handling

The current implementation focuses on reconstructing the topological structure of the mesh and does not yet compute orientation information for intermediate entities. Orientation is required for certain finite element formulations and consistent traversal of shared entities.

The method for computing orientation, based on reference cell conventions and vertex ordering, is well understood and can be incorporated without altering the overall algorithmic structure. Due to time constraints, this component is left as future work.

4 Results and Validation

The interpolation algorithm was validated using a combination of structural checks and topological invariants. In addition to verifying consistency with PETSc's internal mesh representation, a dedicated validation test was constructed based on the *Euler characteristic*, which provides a global topological check on the correctness of the interpolated mesh.

```
        4                     4
      / | \                 / | \
     /  |  \               8  |  9
    /   |   \             /   |   \
   2  0 | 1  5    ->     2  0 7 1  5
    \   |   /             \   |   /
     \  |  /               6  |  10
      \ | /                 \ | /
        3                     3
```

Figure 2: Interpolating the full topology for a triangle doublet. ASCII souce:[3]

4.1 Euler Characteristic Test

For a two dimensional mesh, the Euler characteristic is defined as

$$\chi = V - E + F,$$

where V is the number of vertices, E is the number of edges, and F is the number of faces. For a simply connected planar mesh without holes, the Euler characteristic has a known expected value (e.g., $\chi = 1$ for a planar mesh with boundary, or $\chi = 2$ for a closed surface, depending on boundary conventions).

A test was implemented that computes the Euler characteristic of the interpolated mesh by counting the number of entities in each topological stratum after interpolation. This test serves as a lightweight but effective validation mechanism, as it captures global topological properties that cannot be detected through purely local consistency checks.

4.2 Detection of Topological Errors

The Euler characteristic test is particularly sensitive to errors arising from incorrect interpolation, including:

- Duplicated shared edges or faces, which artificially increase the number of intermediate entities
- Missing intermediate entities, which reduce the expected entity counts

For example, duplicating a shared edge between two adjacent cells increases the number of edges without a corresponding change in the number of vertices or faces, resulting in an incorrect Euler characteristic. Similarly, failure to correctly identify shared entities may introduce topological gaps that manifest as deviations from the expected value of χ.

4.3 Validation Results

The interpolated meshes produced by the implementation were subjected to the Euler characteristic test across multiple test cases. We primarily tested interpolation on two dimensional planar meshes with triangular cells. These meshes range from reference meshes with single triangular cells to meshes with shared edges and multiple adjacent cells. In all tested cases, the computed Euler characteristic yielded 1, which matches the expected value for the corresponding mesh topology.

These results provide strong evidence that the interpolation process correctly reconstructs intermediate strata and properly deduplicates shared entities. This offers a robust global validation that complements PETSc's existing local consistency checks.

5 Conclusion and Future Work

This project presents an algorithm for reconstructing the full topological structure of a mesh given only its vertices and top dimensional cells. The method explicitly generates intermediate entities, deduplicates shared topology, and assigns a global numbering consistent with PETSc's internal conventions. Unlike existing approaches embedded within monolithic mesh generation pipelines, this work isolates topology interpolation as a standalone transformation.

The algorithm is implemented within PETSc using existing transformation interfaces, while remaining modular and extensible. Key components include cell specific transformation rules, a two level hash based deduplication strategy, and an offset table that enables consistent global indexing and reverse mapping from produced entities to their source cells.

Correctness is validated using a test based on the Euler characteristic, which detects topological inconsistencies such as duplicated shared entities or unintended holes. All tested meshes satisfy the expected Euler characteristic, indicating globally consistent interpolation.

The current implementation focuses on triangular meshes as a concrete demonstration of the approach. This limitation is not fundamental; Support for additional cell types or higher dimensional meshes requires only the definition of corresponding transformation rules and extensions to the deduplication keys. Future works include implementing orientation handling for supported types, as well as extending interpolation functionality to more polytope types.

References

[1] Keyes, D. E., McInnes, L. C., Woodward, C., Gropp, W., Myra, E., Pernice, M., et al. (2013). *Multiphysics simulations: Challenges and opportunities.* International Journal of High Performance Computing Applications, 27(1), 4–83. doi:10.1177/1094342012468181

[2] PETSc Developers. *DMPlex User Manual*. Available at: https://petsc.org/release/manual/dmplex/. Accessed 14 Dec 2025.

[3] PETSc Developers. *DMPlex interpolation test (`ex7.c`)*. GitLab repository, branch `knepley/featureplextransforminterpolate`. Available at https://gitlab.com/petsc/petsc/-/blob/knepley/feature-plextransform-interpolate/src/dm/impls/plex/tests/ex7.c. Accessed 14 December 2025.