AN ERGONOMIC APPROACH TO TOPOLOGICAL TRANSFORMATIONS OF UNSTRUCTURED MESHES

Daniel Shapero[1]

[1] *University of Washington, Seattle, WA, USA, shapero@uw.edu*

ABSTRACT

In this paper, we will describe an approach to performing local transformations to the topology of an unstructured mesh. Our approach, borrowing some ideas from algebraic topology, represents the mesh topology using linear operators. We can then define transformations through purely linear algebraic means. This choice of representation makes for much easier verification of the transformation kernels.

Keywords: mesh generation, computational geometry, algebraic topology

1. INTRODUCTION

All algorithms for generating unstructured meshes are based on applying a sequence of local transformations to the mesh topology. For example, to compute the Delaunay triangulation, the Lawson algorithm uses a sequence of bistellar flips, while the Bowyer-Watson algorithm is based on splitting star-shaped polytopes along a vertex [1]. Implementing these transformations, however, can be difficult and error-prone.

In this paper, we will show how to implement topological transformations on the *boundary operators* from algebraic topology. Boundary operators are really just integer matrices, which means we can describe transformations using linear algebra. Our contribution is a linear algebraic representation of splitting a star-shaped collection of polytopes on a vertex, together with a proof-of-concept application to computing convex hulls. The key result is in equations (10) and (11).

The idea of using linear algebra as a low-level description language for building applications in other domains of science or engineering is not new. The GraphBLAS project aims to implement common algorithms for the analysis of large graphs using linear algebra [2]. In computational geometry, DiCarlo and others [3] used chain complexes to implement the Euler make/kill operators from solid modeling.

2. CHAIN COMPLEXES

CW complexes are the most general structure describing spaces obtained by gluing cells together. Simplicial, cubical, and polygonal meshes are all CW complexes, but each category has different constraints on what a cell can be. The boundary operators of a complex are integer matrices that encode the adjacency relations between k- and $k-1$-dimensional cells. The boundary operators describe what the faces of a given cell are, but they also describe how faces are oriented in the cells that contain them. Orientation allows us to express the intuitive idea that two neighboring cells see their common face from opposite sides. In the following, we briefly recall some definitions; see [4].

CW complexes are defined inductively by dimension. A 0-dimensional complex X^0 is a finite collection of points. A regular n-dimensional CW complex consists of an $n-1$-dimensional complex X^{n-1} together with a collection $\{\phi_\alpha^n\}$ of homeomorphisms, called *attaching maps*, from the unit sphere S^{n-1} into X^{n-1}. The space X^n is then the disjoint union of copies of the unit ball with each copy's boundary identified with its image in X^{n-1} under the attaching map with the quotient topology.

Given a k-cell σ and a $k-1$-cell τ of a complex X, we

can define the *incidence number* as

$$[\sigma,\tau] = \begin{cases} 0 & \tau \text{ is not in } \sigma \\ +1 & \tau \text{ is positively oriented in } \sigma \\ -1 & \tau \text{ is negatively oriented in } \sigma \end{cases} \quad (1)$$

The formal definition of incidence number uses the topological degree of the attaching map ϕ of τ into $\partial\sigma$, which we will not define here. For simplicial and cubical complexes, the incidence numbers between cells can be read off directly from their constituent vertices.

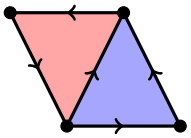
The key objects of interest for us are the *boundary operators*, which are mappings between *chain modules*. The chain module $\mathscr{C}_k$ is the set of all formal $\mathbb{Z}$-linear combinations of cells of dimension k. Given a single k-cell σ of X, we define

$$\partial_k \sigma = \sum_{\tau \in X^{k-1}} [\sigma,\tau]\tau \quad (2)$$

and extend ∂_k by $\mathbb{Z}$-linearity to a map from $\mathscr{C}_k$ to $\mathscr{C}_{k-1}$. The most important fact about boundary operators and the wellspring of homology theory is that

$$\partial_k \cdot \partial_{k+1} = 0. \quad (3)$$

A collection of $\mathbb{Z}$-modules $\mathscr{C}_k$ together with homomorphisms $\partial_k : \mathscr{C}_k \to \mathscr{C}_{k-1}$ that satisfy equation (3) is called a *chain complex*.



$$\partial_1 = \begin{bmatrix} - & & + & - & \\ + & - & & & + \\ & + & - & & \\ & & & + & - \end{bmatrix}$$

$$\partial_2 = \begin{bmatrix} + & - \\ + & \\ + & \\ & + \\ & + \end{bmatrix}$$

Figure 1: Pair of adjacent triangles (left) and their boundary matrices (right).

The abstract concept of a CW complex can be made computable by representing its boundary operators as (sparse) matrices with integer entries. We can read off the faces and sub-faces of a particular cell by looking at which rows are non-zero in the corresponding columns of the boundary matrices. Equation (3) is easy to verify in practice and guarantees the topological validity of the data structure. See figure 1 for an illustration of a simple topology together with the corresponding boundary operators.

We'll introduce one final conceit that makes later constructions much more elegant. The definition above assumes that the chain complex stops at $\mathscr{C}_0$ – formal linear combinations of vertices. Instead, we will add a *bottom* module $\mathscr{C}_{-1}$ consisting of a single cell $\perp$ of dimension -1. We then define the boundary of every vertex v_i to be $+\perp$. If we write $\mathbb{1}$ for the column vector of all 1s, then $\partial_0 = \mathbb{1}^*$, i.e. the row vector of all 1s. The addition of this bottom cell is especially convenient because the condition $\partial_0\partial_1 = 0$ implies that the boundary of every edge e has a negative and a positive vertex: $\partial e = v_i - v_j$ for some i, j. We cannot have, for example, $\partial e = v_i + v_j$, which would be undesirable.

3. TRANSFORMATIONS

If we choose to represent mesh topologies using boundary operators, we can describe transformations through linear algebra. Phrasing the problem in this way makes verifying correctness easy.

First, observe that if A, B are integer matrices such that the image of ∂_{k+1} is an invariant subspace of $A \cdot B$, then the matrices

$$\partial'_k = \partial_k \cdot A, \quad \partial'_{k+1} = B \cdot \partial_{k+1} \quad (4)$$

still satisfy $\partial'_k \cdot \partial'_{k+1} = 0$. A particular case is $A \cdot B = I$, which includes permutations of the cell ordering. The more general case regarding the image of $A \cdot B$ is needed for some irreversible transformations.

3.1 Merging

A *merge* of a set of k-cells replaces them with a single cell (provided that their union is simply-connected). Merging is a column operation on the matrix ∂_k. In the simplest case, the result column is the sum of all the columns to be merged, but in general we might need to flip some signs:

$$\partial'_k = \partial_k \cdot \mathrm{diag}(s_0, \ldots, s_m) \cdot \mathbb{1}, \quad (5)$$

$$\partial'_{k+1} = \begin{bmatrix} 0 \ldots 1 \ldots 0 \end{bmatrix} \partial_{k+1}. \quad (6)$$

where s_i are all ± 1. The signs are chosen so that any higher-dimensional cell σ has the same incidence with respect to any of the cells τ to be merged. The transformation to the rows of ∂_{k+1} collapses all incidence to any of the desired k-cells into incidence to the merged k-cell. For merging cells of top dimension n, there are no higher-dimensional cells to apply equation (6) to and this step is left out.

Edge collapsing, the key transformation in surface simplification algorithms [5], is a merge of two vertices.

3.2 Splitting

A *split* divides up the union of several polytopes along a vertex. The key correctness criteria for this operation are that (1) every newly-created polytope contains the splitting vertex and (2) the boundary of the sum of all polytopes does not change. We'll describe the 2D case first and then proceed to arbitrary dimensions.

Suppose that a collection of adjacent polygons has the boundary operators ∂_1 and ∂_2. We first have to draw edges between the new vertex and all the vertices of the polygon. The orientation of these new edges is arbitrary, so we can assume that every edge goes from the splitting vertex v to the polygon vertices. Another way of stating this is that every new edge e is negatively-incident to v and positively-incident to some polygon vertex. In terms of matrices, the new 1-boundary operator is

$$\partial_1' = \begin{bmatrix} \partial_1 & I \\ 0 & -\mathbb{1}^* \end{bmatrix} \tag{7}$$

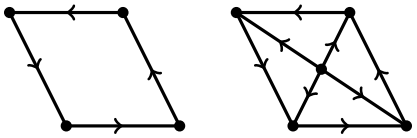
where I is the identity matrix. The key step here is defining the 2-boundary matrix:

$$\partial_2' = \begin{bmatrix} \mathrm{diag}(\partial_2 \cdot \mathbb{1}) \\ -\partial_1 \cdot \mathrm{diag}(\partial_2 \cdot \mathbb{1}) \end{bmatrix} \tag{8}$$

A rudimentary calculation shows that $\partial_1'\partial_2' = 0$. Since $\mathrm{diag}(z) \cdot \mathbb{1} = z$ for any vector z, we also find that

$$\partial_2' \mathbb{1} = \begin{bmatrix} \partial_2 \mathbb{1} \\ 0 \end{bmatrix} \tag{9}$$

or in other words the new polygon has the same boundary as the old. Figure 2 illustrates the split transformation on a single quadrilateral and shows the boundary matrices before and after.



$$\partial_1 = \begin{bmatrix} - & & & + \\ + & - & & \\ & + & - & \\ & & + & - \end{bmatrix}, \quad \partial_2 = \begin{bmatrix} + \\ + \\ + \\ + \end{bmatrix}$$

$$\partial_1' = \left[\begin{array}{cccc|cccc} - & & & + & + & & & \\ + & - & & & & + & & \\ & + & - & & & & + & \\ & & + & - & & & & + \\ \hline & & & & - & - & - & - \end{array}\right], \quad \partial_2' = \left[\begin{array}{c} + \\ + \\ + \\ + \\ \hline - \\ - \\ - \\ - \end{array}\right]$$

Figure 2: Quadrilateral before and after splitting on a new vertex in the center (top) and boundary matrices before and after (bottom).

We can get an idea for how to extend this to n dimensions by remembering that $\mathbb{1}^* = \partial_0$ in equation (7). This suggests the transformation

$$\partial_k' = \begin{bmatrix} \partial_k & I \\ 0 & -\partial_{k-1} \end{bmatrix} \tag{10}$$

$$\partial_n' = \begin{bmatrix} \mathrm{diag}(\partial_n \cdot \mathbb{1}) \\ -\partial_{n-1}\mathrm{diag}(\partial_n \cdot \mathbb{1}) \end{bmatrix} \tag{11}$$

Again, a rudimentary calculation shows that the fundamental equation (3) still holds and that the new polytopes have the same boundary as the old.

Equation (10) has appeared before in the literature on homological algebra as the expression for the boundary operators of the *cone* of a space [6]. To our knowledge, using these equations for doing real computations is entirely new.

The Bowyer-Watson algorithm for Delaunay triangulation and all common algorithms for computing convex hulls only require the split transformation [1].

3.3 Splitting and merging

Other transformations can be defined by combining a sequence of splits and merges. For example, figure 3 shows how to perform a 2-2 flip in 2D by first splitting the quadrilateral into four triangles and then performing a sequence of merges and figure 4 shows the same process for a 2-3 flip in 3D. (We've shown an initial merge step for illustrative purposes, but this merge is actually part of the subsequent split.)

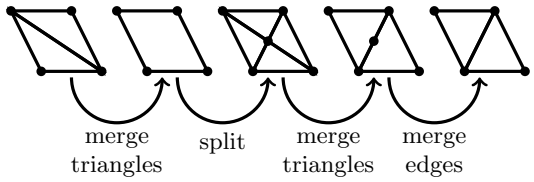


Figure 3: A 2-2 flip, implemented as a sequence of merges and splits. The vertex added by splitting the quadrilateral is deleted when the two edges are merged in the final transformation.

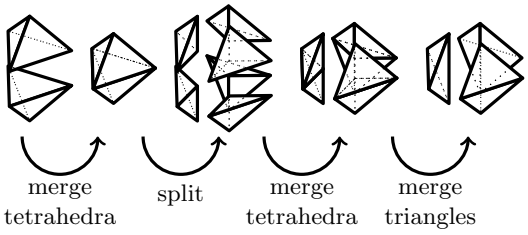


Figure 4: A 2-3 flip implemented as a sequence of merges and splits. We've shown the tetrahedra in an "exploded" view to help with visualization.

A few papers have proposed using multi-cell transformations for 3D mesh improvement [7]. Multi-cell transformations are more complex than 2-3 or 3-2 flips, but [8] showed that they can be implemented as a sequence of flips. Using the boundary operators might make it possible to implement complex transformations with less effort. The split transformation that

```python
import numpy as np
from typing import List

def split(D: List[np.ndarray]) -> List[np.ndarray]:
    # Create the boundary matrices for 1 <= k < n
    n_vertices = D[0].shape[1]
    E = [np.ones((1, n_vertices + 1), int)]
    for k in range(1, len(D) - 1):
        n_cells = D[k].shape[1]
        n_sub_faces, n_faces = D[k - 1].shape
        I = np.identity(n_faces, int)
        Z = np.zeros((n_sub_faces, n_cells), int)
        E_k = np.block([[D[k], I], [Z, -D[k - 1]]])
        E.append(E_k)

    # Create the top-dimensional boundary matrix
    C = np.diag(np.sum(D[-1], axis=1))
    E_n = np.vstack((C, -D[-2] @ C))
    E.append(E_n)
    return E
```

Figure 5: Python source code for the split transformation. Line 13 corresponds to equation (10); lines 17 and 18 correspond to equation (11). The real version has some additional logic to remove empty cells.

we derived here is based on computing the topological cone of a space and then removing the base of the cone. Multiface retriangulation, as advocated in [8], is a transformation of a *suspension*, a related construction in algebraic topology [4].

4. DEMONSTRATION

As a proof of concept, we developed a software package called *zmsh* which implements the split transformation defined above [9]. The source code for the split transformation is shown in figure 5. We used the split transformation to implement a convex hull algorithm that works in arbitrary dimensions. To test the convex hull code, we used (1) random point sets up to dimension 5 and (2) several synthetic input sets with various degeneracies such as coplanarity. For geometric predicates, we used the Python standard library's built-in routines to convert a floating point value exactly to a ratio of big integers.

5. CONCLUSION

Doubly-connected edge lists have historically been popular for mesh generation because they offer a simple interface for traversing the topology [10]. Here we propose that boundary operators are an ideal representation if the goal is to perform topological transformations. This idea has appeared before, for example in the work of DiCarlo and others [3].

Boundary operators are only necessary for representing a small patch of the topology at a time. Once the transformed patch is computed, it can be translated back to a set of simplexes. Boundary operators are useful for describing non-simplicial intermediate states of a transformation; they are not space-optimal for describing an entire simplicial complex.

When the objects of study can be represented as linear operators, we can apply linear algebraic reasoning to define transformations and verify that they preserve all of the important invariants. The condition in equation (3) that the product of two boundary operators is zero is a very powerful invariant for ensuring the validity of the underlying topology.

References

[1] Cheng S.W., Dey T.K., Shewchuk J., Sahni S. *Delaunay mesh generation*. CRC Press Boca Raton, 2013

[2] Mattson T., Bader D., Berry J., Buluc A., Dongarra J., Faloutsos C., Feo J., Gilbert J., Gonzalez J., Hendrickson B., et al. "Standards for graph algorithm primitives." *2013 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–2. IEEE, 2013

[3] DiCarlo A., Milicchio F., Paoluzzi A., Shapiro V. "Solid and physical modeling with chain complexes." *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pp. 73–84. 2007

[4] Hatcher A. *Algebraic Topology*. Cambridge University Press, 2002

[5] Guéziec A. "Surface simplification with variable tolerance." *Second Annual Symposium on Medical Robotics and Computer Assisted Surgery, 1995*. 1995

[6] Gelfand S.I., Manin Y.I. *Homological algebra*, vol. 38. Springer Science & Business Media, 1994

[7] Klingner B.M., Shewchuk J.R. "Aggressive tetrahedral mesh improvement." *Proceedings of the 16th international meshing roundtable*, pp. 3–23. Springer, 2008

[8] Misztal M.K., Bærentzen J.A., Anton F., Erleben K. "Tetrahedral mesh improvement using multiface retriangulation." *Proceedings of the 18th international meshing roundtable*, pp. 539–555. Springer, 2009

[9] Shapero D. "zmsh." URL `https://doi.org/10.5281/zenodo.7502592`

[10] Guibas L., Stolfi J. "Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams." *ACM transactions on graphics (TOG)*, vol. 4, no. 2, 74–123, 1985