Hence every tuple must be given a sign indicating whether it agrees ( ) or disagrees ( ) with the intrinsic orientation of the simplex. Given a set of integers representing a simplex, there are two equivalence classes of orderings of the given tuple: the even and odd permutations of the integers in question. These two equivalence classes correspond to the two possible orientations of the simplex (see Fig. 2).

Note that assigning a sign to any one alias (*i.e.*, the representative) implicitly assigns a sign to all other aliases. Let us assume for a moment that the sign of all representatives is known. Then the sign S of an arbitrary tuple t, with representative r, is

$$\mathbf{S}\ t \qquad \begin{array}{ll} \mathbf{S}\ r & \text{if } t \text{ is in the same equivalence class as } r \\ \mathbf{S}\ r & \text{if } t \text{ is in the opposite equivalence class} \end{array}$$

More formally, let P be the permutation that permutes t into r (*i.e.*, $r \quad P\ t$). Then

$$\mathbf{S}\ t \quad \mathbf{S}\ P\ \mathbf{S}\ P\ t$$

(Here $\mathbf{S}\ P$ denotes the sign of the permutation P with 1 for even and 1 for odd permutations.)

All that remains, then, is to choose an intrinsic orientation for each simplex and set the sign of the representative alias accordingly. In general the assignment of orientations is arbitrary, as long as it is consistent. For all subsimplices we choose the representative to be positively oriented, so that the right-hand-side of the above expression reduces to $\mathbf{S}\ P$. For top-level simplices (tets in 3D, triangles in 2D), we use the convention that a positive volume corresponds to a positively oriented simplex. We therefore require a volume form which, together with an assignment of points to vertices, will allow us to orient all tets. Recall that a volume form accepts three (for 3D; two for 2D) vectors and returns either a positive or negative number (assuming the vectors are linearly independent). So the sign of a 4-tuple is:

$$\mathbf{S}\ i_0\ i_1\ i_2\ i_3 \quad \mathbf{S}\ \mathrm{Vol}\ p_{i_1} \quad p_{i_0}\ p_{i_2} \quad p_{i_0}\ p_{i_3} \quad p_{i_0}$$

4 The Boundary Operator

The *faces* of a k-simplex are the k 1 -simplices that are incident on it, *i.e.*, the subset of one lower dimension. Every k-simplex has k 1 faces. Each face corresponds to removing one integer from the tuple, and the relative orientation of the face is 1 ^{i} where i is the index of the integer that was removed. To clarify:

The faces of a tet $t_0\ t_1\ t_2\ t_3$ are $t_0\ t_1\ t_2$, $t_0\ t_1\ t_3$, $t_0\ t_2\ t_3$, and $t_1\ t_2\ t_3$.

The faces of a triangle $f_0\ f_1\ f_2$ are $f_0\ f_1$, $f_0\ f_2$, and $f_1\ f_2$.

The faces of an edge $e_0\ e_1$ are e_0 and e_1 .

We can now define the boundary operator which maps simplices to their their faces. Given the set of tets $\mathbf{T}$ we define $^3 : \mathbf{T} \quad \mathbf{F}^4$ as

$$^3 \quad i_0\ i_1\ i_2\ i_3 \qquad\qquad i_0\ i_1\ i_2 \quad i_0\ i_1\ i_3 \quad i_0\ i_2\ i_3 \quad i_1\ i_2\ i_3$$

Similarly for $^2 : \mathbf{F} \quad \mathbf{E}^3$ (which maps each triangle to its three edges) and $^1 : \mathbf{E} \quad \mathbf{V}^2$ (which maps each edge to its two vertices).

We represent these operators as sparse adjacency matrices (or, equivalently, signed adjacency lists), containing elements of type 1 and 1 only. So ^{3} is implemented as a matrix of size $\mathbf{F} \quad \mathbf{T}$ with 4 non-zero elements per column, ^{2} an $\mathbf{E} \quad \mathbf{F}$ matrix with 3 non-zero elements per column, and ^{1} a $\mathbf{V} \quad \mathbf{E}$ matrix with 2 non-zero elements per column (one 1 and one 1). The transposes of these matrices are known as the *coboundary* operators, and they map simplices to their *cofaces*—neighbor simplices of one higher dimension. For example, $^{2\ T}$ maps an edge to the "pinwheel" of triangles incident on that edge.

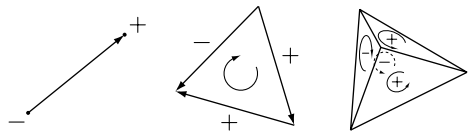

Figure 3: *The boundary operator identifies the faces of a simplex as well as their* relative *orientations. In this illustration, arrows indicate intrinsic orientations and signs indicate the relative orientation of a face to a parent.*

These matrices allow us to iterate over the faces or cofaces of any simplex, by walking down the columns or across the rows, respectively. In order to traverse neighbors that are more than one dimension removed (*i.e.*, the tets adjacent to an edge or the faces adjacent to a vertex) we simply concatenate the appropriate matrices, but without the signs. (If we kept the signs in the matrix multiplication any such consecutive product would simply return the zero matrix reflecting the fact that the boundary of a boundary is always empty.)

5 Construction

Although we still need a few auxiliary wrapper and iterator data structures to provide an interface to the mesh elements, the simplex lists and boundary matrices contain the entirety of the topological data of the mesh. All that remains, then, is to fill in this data.

We read in our mesh as a list of $x\ y\ z$ vertex positions and a list of 4-tuples specifying the tets. Reading the mesh in this format eliminates the possibility of many non-manifold scenarios; for example, there cannot be an isolated edge that does not belong to a tet. We assume that all integers in the range 0 n appear at least once in the tet list (this eliminates isolated vertices), and no integer outside of this range is present.

Once $\mathbf{T}$ is read in, building $\mathbf{E}$ and $\mathbf{F}$ is trivial; for each tuple in $\mathbf{T}$, append all subsets of size 2 and 3 to $\mathbf{E}$ and $\mathbf{F}$ respectively. We must be sure to avoid duplicates, either by using a unique associative container, or by sorting the list afterward and removing duplicates. Then the boundary operator matrices are constructed as follows:

```
for each simplex s
    construct a tuple for each face f of s
        as described in Section 4
    determine the index i of f by locating
        its representative
    set the entry of the appropriate matrix
        at row i, column s to S f
```

Figure 4 shows a complete example of a mesh and its associated data structure.

6 DEC Operators

Now we discuss the implementation of the two most commonly used DEC operators: the exterior derivative and the Hodge star. As we will see, in the end these also amount to nothing more than sparse matrices that can be applied to our form vectors.