Putting simplices in a canonical form using `miscmath.h`

```
StdEdge labelEdge(Edge& e, Vertex& v)
StdFace labelFace(Face& f, Vertex& v)
StdFace labelFace(Face& f, Edge& e)
StdTetra labelTetra(Tetra& t, Edge& e)
StdTetra labelTetra(Tetra& t, Vertex& v)
StdTetra labelTetra(Tetra& t, Face& f)
StdTetra labelTetra(Tetra& t)
```

Keywords

simplex, labeling, canonical form

Authors

Joseph Thomas

Introduction

To perform a calculation on a triangulation, one often needs to inspect the topology of the triangulation. This inspection usually involves examining simplices in a particular region of the triangulation and understanding how they are connected. For example, given a face we might like to know its edges and vertices and their relative positions. This module provides a convenient way to address these questions.

Description

Currently, our topological data structure consists of a collection of `C++ Map` data structures that map integers to simplex objects. Consequently, one way of specifying a simplex in the current triangulation is with an integer. A labeling of a given simplex can thus be given as a `struct` of integers, where each integer refers to a particular simplex:

For example, below is the definition for a labeled face:

```
struct stand_psn_face{
       int v1;
       int v2;
       int v3;

       int e12;
       int e13;
       int e23;
};
typedef stand_psn_face StdFace;
```

The integers `v1`, `v2`, and `v3` each represent a vertex in `Triangulation::vertexTable`. Likewise, `e12` represents an edge in `Triangulation::edgeTable`, and this edge has as its endpoints the vertices represented by `v1` and `v2`. The other structures, `StdEdge` and `StdTetra` are analogous.

Each procedure (with the exception of the last) answers the following request: ¨Label Simplex A with respect to Simplex B,¨ subject to the requirement that Simplex B is of lower dimension than Simplex A, and Simplex B is adjacent to Simplex A.

The labeling is determined as follows. Depending on the dimension of Simplex B, the chosen labeling procedure makes sure that the integer `v1`, `e12`, or `f123` refers to Simplex B. In some cases, this will mean that other positions in the labeling are ambiguous. For example, if we used `labelTetra( Tetra& t, Edge& e )` to label a tetrahedron with respect to one of its edges, then the `v1` and `v2` entries could be permuted while maintaining a correct labeling—we only insist that `e12` denotes our chosen edge, and that the labeling is consistent with the tetrahedron's topology.

A description of our current system of computing a labeling makes this clearer. Suppose we are labeling simplex A with respect to simplex B. First, we compute the vertices of B and the vertices of A. The vertices of B are assigned to `v1`, `v2`, and `v3`, depending on how many vertices B contains and any mapping of B's vertices to these variables is valid. The vertices of A that are not in B are then mapped into the remaining `v` variables; again, any mapping is valid. Now all of the vertices are labeled, and the remaining references are determined by this labeling—in other words `e12` must be assigned to the edge containing vertex `v1` and vertex `v2`, face `f234` must be the face containing vertices `v2`, `v3`, and `v4`, etc.

The only exceptions to this rule are the labeling procedures that take a single simplex as input. In these procedures, we assume the user wants *some* consistent labeling of the given simplex, without any constraints on how the vertices are to be labeled.

All of the procedures in this module work in constant time. This is achieved by using the fact that a given simplex can only be incident on a bounded number of lower dimensional simplices.

Example

Below is a portion of code used in calculating two edge heights for a given face `f` and tetrahedron `t` (see edge_height.cpp for more details about this particular quantity). In this calculation, we use `labelTetra` to set up the topological portion of the calculation:

```
// Declared elsewhere: Face& f, Tetra& t
StdTetra st = labelTetra( t, f );
Edge& ed12 = Triangulation::edgeTable[ st.e12 ];
Face& fa123 = Triangulation::faceTable[ st.f123 ];
Face& fa124 = Triangulation::faceTable[ st.f124 ];

double hij_l = EdgeHeight::At( ed12, fa123 )->getValue();
double hij_k = EdgeHeight::At( ed12, fa124 )->getValue();
```

Limitations

- There is currently no error checking in this module; the results of labeling one simplex with respect to a non-adjacent simplex is not defined.
- It isn't totally clear that this module will necessarily cover ¨unusual boundary cases¨ such as a face with only one or two vertices.

Testing

This module is used extensively in our implementation of the Einstein-Hilbert-Regge geometry. Since we have thoroughly debugged that module, we believe this code is correct as well.

Future Work

It would be nice to be able to write labellings as collections of pointers rather than collections of integers. The current labellings are **not type-safe** and often require an extra step (a lookup in a `Triangulation` table) to be used.

Warning: It would be tempting to implement the above improvement by recording pointers to simplices in our global `Triangulation` data structure. We have already tried this and it **did not work.** Essentially, improving the labeling code to use pointers would require us to rewrite the triangulation data structure to use dynamic memory allocation (rather than global variables). This is probably an improvement we should look into, but it is a big commitment, since most every module in the project depends on the topology/triangulation module.