

```

/* normal_surfaces.h */

#ifndef _normal_surfaces_
#define _normal_surfaces_

#include "SnapPea.h"

/* What is a normal surface?

Embedded surfaces will be described using normal surface theory. A given normal surface intersects each ideal tetrahedron in a (possibly empty) disjoint collection of topological squares and triangles.


      / \
     /   \
    / . . . \
   /#####\ 
  /#####\ 
 /#####\ 
--|#####|--
 |#####| 
 |#####| 
 |#####| 
 \#####/ 
  \#####/ 
   \#####/ 
    \ . . . /
     \   /
      \ /

Each square cuts across the ideal tetrahedron as shown above, so as to separate the ideal tetrahedron's ideal vertices into two groups of two ideal vertices each. There are three ways to do this, but a given ideal tetrahedron may contain squares of only one type, because the surface is embedded.


          / \
         /   \
        / . . . \
       /####\ 
      /##\#\\ 
     / # \\# 
    /  ##  \ 
   /-----\ 
  /           \ 
 /             \ 
/               \ 
 \              / 
  \            / 
   \         / 
    \       / 
     \     / 
      \___/ 

Each triangle cuts across the ideal tetrahedron as shown above, so as to separate one ideal vertex from the other three. There are four ways to do this, and triangles of all four types may be present simultaneously, because they don't intersect each other or the squares.



How is a normal surface modelled in a Triangulation?

The squares and triangles are modelled in the Tetrahedron data structure as follows. The field tet->parallel_edge contains the EdgeIndex (= 0, 1 or 2) of an edge which does not intersect the squares. The opposite edge (= 5, 4 or 3, respectively) will lie on the opposite side of the squares, and all remaining edges will intersect the squares. The field tet->num_squares tells how many (parallel) squares lie in the Tetrahedron. The fields tet->num_triangles[v] tell how many (parallel) triangles lie near the ideal vertex of VertexIndex v. When several surfaces must be available simultaneously, the parallel_edge, num_squares and num_triangles[] fields are copied to the corresponding fields of a NormalSurface structure (below).
*/
```

```

/*
 * SnapPea.h contains an opaque typedef for a NormalSurfaceList.
 * Here we make a typedef for a NormalSurface, so the NormalSurfaceList
 * can refer to it.
 */
typedef struct NormalSurface    NormalSurface;

struct NormalSurfaceList
{
    /*
     * The normal surfaces all share the same underlying triangulation.
     * This triangulation is obtained from the user's triangulation
     * by filling in all closed cusps.
     */
    Triangulation    *triangulation;

    /*
     * The NormalSurfaces are kept on an array.
     */
    int              num_normal_surfaces;
    NormalSurface    *list;
};

struct NormalSurface
{
    /*
     * The UI should report the information on orientability, two-sidedness
     * and Euler characteristic to the user.  In the present implementation
     * of find_normal_surfaces(), is_connected will always be TRUE.
     */
    Boolean          is_connected,
                    is_orientable,
                    is_two_sided;
    int              Euler_characteristic;

    /*
     * The following fields describe the normal surface.
     * In the i-th Tetrahedron of the Triangulation, the squares (if any)
     * are parallel to the edge of EdgeIndex parallel_edge[i]
     * (= 0, 1 or 2).  The number of such squares is num_squares[i].
     * The number of triangles at vertex v is num_triangles[i][v].
     * These fields record the values of the corresponding fields
     * in the Tetrahedron data structure.
     */
    EdgeIndex        *parallel_edge;
    int              *num_squares,
                    (*num_triangles)[4];

    /*
     * find_normal_surfaces() temporarily keeps the NormalSurfaces on
     * a NULL-terminated singly linked list.  As soon as it know how
     * many it has found, it transfers them to an array, and this
     * "next" field is ignored from then on.
     */
    NormalSurface    *next;    /* used locally within find_normal_surfaces() */
};

#endif

```