

```
/*
 * terse_triangulation.h
 *
 * This data structure describes a Triangulation in a "terse" format.
 * The data structure is intended to minimize storage requirements,
 * and contains only the minimum information required to reconstruct
 * the Triangulation in the full triangulation.h format.
 *
 * The data structure allows the option of including the value of the
 * Chern-Simons invariant (mod 1/2).
 *
 * Most often the TerseTriangulation will be used in conjunction with
 * the TersestTriangulation data format (cf. tersest_triangulation.h),
 * which represents the TerseTriangulation as a short sequence of
 * characters, suitable for storage in a file or transmission over
 * a network.
 *
 * Bill Thurston provided the original idea for this data structure.
 * The implementation has evolved from code written by Martin Hildebrand at
 * the Geometry Center (then the Geometry Supercomputer Project) in 1988.
 *
 * [The following description of the TerseTriangulation assumes basic
 * familiarity with the discussion "Gluing of ideal tetrahedra" preceding
 * the Permutation typedef in kernel_typedefs.h.]
 *
 * The TerseTriangulation data structure provides the data for an
 * algorithm for constructing an ideal triangulation. The algorithm is
 * as follows. We begin with n ideal tetrahedra, which we are going
 * to assemble into an ideal triangulation. The vertices of each
 * tetrahedron are numbered 0, 1, 2 and 3, and the faces are numbered
 * according to the opposite vertices. The tetrahedra are all identical.
 * Take one tetrahedron and call it tetrahedron 0. Consider face 0
 * of tetrahedron 0. It may glue to some other face of tetrahedron 0,
 * or it may glue to a different tetrahedron. The first entry of the
 * Boolean array glues_to_old_tet tells us which is the case. That is,
 * if glues_to_old_tet[0] is TRUE, then face 0 glues to some other face
 * of tetrahedron 0; if glues_to_old_tet[0] is FALSE, it glues to a
 * different tetrahedron.
 *
 * In the case that glues_to_old_tet[0] is TRUE, which_old_tet[0] will tell
 * us which old tetrahedron it glues to (here which_old_tet[0] must
 * obviously be 0, because tetrahedron 0 is the only tetrahedron we're
 * working with so far, but in a minute you'll see the full generality of
 * the which_old_tet array), and which_gluing[0] tells us the gluing
 * pattern (cf. the discussion "Gluing of ideal tetrahedra" in
 * kernel_prototypes.h).
 *
 * In the case that glues_to_old_tet[0] is FALSE, we attach one of the
 * as-yet-unused tetrahedra to face 0 of tetrahedron 0. The new
 * tetrahedron will be called tetrahedron 1, and its vertices will
 * inherit indices from the vertices of tetrahedron 0 in the canonical
 * way, by reflection across their common face. (The gluing will be
 * the identity Permutation.)
 *
 * Once we've taken care of face 0 of tetrahedron 0, we move on to
 * face 1 of tetrahedron 0. If it's already been glued (to face 0),
 * we do nothing. Otherwise we consult the next entry in the
 * glues_to_old_tet[] array: if it's TRUE, then the next entries in
 * the which_old_tet[] and which_gluing[] arrays tell us which tetrahedron
 * it glues to (maybe tetrahedron 0, or maybe tetrahedron 1), and what
 * the gluing Permutation is; if it's FALSE, we attach a new tetrahedron
 * as described above.
 *
 * We continue this process for faces 2 and 3 of tetrahedron 0, then
 * move on to faces 0, 1, 2 and 3 (in that order) of tetrahedron 1,
 * then tetrahedron 2, etc., until we have constructed the entire
 * triangulation. Note that this algorithm assumes the triangulation
 * is connected.
 *
 * If there are n tetrahedra, there'll be 4n faces, and  $(4n)/2 = 2n$ 
 * decisions as to whether to glue to an old tetrahedron or a new one.
 * (The last two "decisions" will always be to glue to an old tetrahedron,
 * but we won't worry about that. The terse string data format does,
```

```
* however, take this into account.) Precisely (n - 1) of those
* decisions will be to add a new tetrahedron, and the remaining (n + 1)
* decisions will be to glue to an old tetrahedron. This means that
* the glues_to_old_tet[] array must have length 2n, while the
* which_old_tet[] and which_gluing[] arrays must have length (n + 1).
*
* The file SnapPea.h contains the "opaque typedef"
*
*     typedef struct TerseTriangulation    TerseTriangulation;
*
* which lets the UI declare and pass pointers to TerseTriangulations
* without actually knowing what they are. This file provides the
* kernel with the actual definition.
*/

#ifndef _terse_triangulation_
#define _terse_triangulation_

#include "kernel.h"

struct TerseTriangulation
{
    /*
     * The first four fields provide the basic data described above.
     */
    int          num_tetrahedra;
    Boolean      *glues_to_old_tet;
    int          *which_old_tet;
    Permutation  *which_gluing;

    /*
     * Optionally, we may wish to record the Chern-Simons invariant,
     * since it isn't computable from scratch (at least not in 1993).
     */
    Boolean      CS_is_present;
    double       CS_value;
};

#endif
```