

```

/*
 * isometry.h
 *
 * This file provides the definition of an IsometryList. It is private
 * to the kernel.
 *
 * An isometry is a map which takes the Tetrahedra of one Triangulation
 * to the Tetrahedra of another, preserving the gluings. Strictly speaking
 * this is a combinatorial equivalence of the two Triangulations, but
 * the Triangulations are typically the canonical ones for their respective
 * manifolds, so the notions of combinatorial equivalence and isometry
 * coincide.
 *
 * The two Triangulations need not be distinct -- if they're the same
 * then an "isometry" is what one normally calls a symmetry.
 *
 * An isometry may be represented in two ways: internally within the
 * Triangulation data structure, and externally using the Isometry data
 * structure.
 *
 * (1) Internal representation. The Triangulation data structure may
 * represent at most one isometry at a time. Each Tetrahedron contains
 * the fields
 *
 *          Tetrahedron    *image;
 *          Permutation    map;
 *
 * The image field contains a pointer to the image of the given
 * Tetrahedron under the isometry. The map field says how the
 * vertices of the given Tetrahedron are taken to the vertices
 * of its image. E.g. if the Permutation is 2013 (meaning 3210 -> 2013)
 * then vertex 0 of the given Tetrahedron maps to vertex 3 of the image
 * Tetrahedron, vertex 1 maps to vertex 1 of the image, vertex 2 maps
 * to vertex 0, and vertex 3 maps to vertex 2.
 *
 * (2) External representation. The Isometry data structure is similar
 * to the internal representation described in the preceding paragraph,
 * only it refers to Tetrahedra by indices rather than pointers.
 * Each Tetrahedron in each manifold is numbered by setting its index
 * field equal to the Tetrahedron's position in its Triangulation's
 * doubly-linked list (the indices run from 0 through n-1).
 *
 * The images and maps are stored in the tet_image and tet_map arrays
 * in the Isometry data structure. The elements of each array are
 * implicitly indexed by the indices of the Tetrahedra in the domain
 * manifold. For example, if
 *
 *          my_isometry->tet_image = {1, 0, 2}
 *
 * and
 *
 *          my_isometry->tet_map = {3102, 3201, 1023},
 *
 * then Tetrahedron #0 of the domain Triangulation will map to
 * Tetrahedron #1 of the image Triangulation via the Permutation
 * 3102; Tetrahedron #1 will map to Tetrahedron #0 via 3201;
 * and Tetrahedron #2 will map to Tetrahedron #2 via 1023.
 *
 * In addition, the Isometry data structure records the action
 * of the Isometry on the Cusps, as explained in the Isometry
 * definition below.
 *
 * The IsometryList data structure contains an array of pointers to
 * Isometries, an integer saying how many there are, and an integer
 * say how many Tetrahedra each Triangulation has. The file SnapPea.h
 * contains the "opaque typedef"
 *
 *          typedef struct IsometryList    IsometryList;
 *
 * which lets the UI declare and pass pointers to IsometryLists without
 * actually knowing what they are. This file provides the kernel with
 * the actual definition.
 *
 * The Isometry data structure also contains a "next" field, which the
 * function compute_cusped_isometries() uses internally while assembling
 * its IsometryList. Other functions ignore the "next" field.
 */

```

```

#ifndef _isometry_
#define _isometry_

#include "kernel.h"

typedef struct Isometry      Isometry;

struct Isometry
{
    /*
     * How many Tetrahedra and Cusps are there?
     */
    int          num_tetrahedra,
                num_cusps;

    /*
     * The Isometry sends Tetrahedron n in the domain
     * manifold to Tetrahedron tet_image[n] in the image manifold.
     */
    int          *tet_image;

    /*
     * The Isometry sends vertex v of Tetrahedron n in the
     * domain manifold to vertex EVALUATE(tet_map[n], v) of
     * Tetrahedron tet_image[n] in the image manifold.
     */
    Permutation  *tet_map;

    /*
     * The Isometry sends Cusp k of the domain manifold
     * to Cusp cusp_image[k] in the image manifold.
     */

    int          *cusp_image;

    /*
     * The matrix cusp_map[k][][] takes the peripheral curves
     * of Cusp k in the domain manifold to the peripheral curves
     * of Cusp cusp_image[k] in the image manifold. That is,
     * the image of a meridian of Cusp k in the domain is
     * cusp_matrix[k][M][M] meridians plus cusp_matrix[k][L][M]
     * longitudes in the image, and similarly for the image of
     * a longitude.
     *
     * Note that the cusp_map matrix is defined even for
     * nonorientable Cusps, since the peripheral curves are stored
     * in the Cusp's orientation double cover. For nonorientable
     * cusps, cusp_matrix[k][L][M] will be a diagonal matrix.
     * The entry cusp_matrix[k][L][L] tells whether the direction
     * of the cusp is reversed. Det(cusp_matrix[k]) tells whether
     * the Isometry acts in an orientation-preserving or orientation-
     * reversing way on the Cusp's orientation double cover (Question:
     * what is the significance of this information, if any?).
     *
     * This scheme applies only to the real cusps -- finite vertices
     * (whose cusp indices are negative) are ignored.
     */

    MatrixInt22  *cusp_map;

    /*
     * Does this Isometry from one cusped manifold to another extend
     * to the closed manifolds obtained by meridional Dehn fillings?
     * If the cusp manifolds are link complements (in any manifolds,
     * not necessarily 3-spheres) this is equivalent to asking whether
     * the Isometry extends to a link homeomorphism.
     */

    Boolean      extends_to_link;

    /*
     * This "next" field is used internally in isometry_cusped.c

```

```
    * while assembling the IsometryList. (The Isometries are
    * temporarily stored on a linked list, then eventually
    * transferred to an array of pointers for external use.)
    */
    Isometry      *next;
};

struct IsometryList
{
    /*
    * How many Isometries are on the list?
    */
    int          num_isometries;

    /*
    * isometry[n] is a pointer to the n-th Isometry
    * on the list. (The "isometry" field itself
    * contains a pointer to an array. Each element of
    * the array is a pointer to an Isometry.)
    *
    * If there are no isometries (num_isometries == 0)
    * then the isometry field is set to NULL. That is,
    * we don't try to allocate an array of zero pointers.
    */
    Isometry      **isometry;
};

#endif
```