

```

/*
 * dual_one_skeleton_curve.h
 *
 * This file defines the data structure used to represent
 * a simple closed curve in the 1-skeleton of a Triangulation's
 * dual complex.
 *
 * The 0-skeleton of the Triangulation's dual complex consists
 * of one vertex in the interior of each Tetrahedron.
 *
 * The 1-skeleton of the Triangulation's dual complex (the
 * "dual 1-skeleton") consists of one edge transverse to each
 * face of each Tetrahedron (more precisely, to each pair
 * of identified faces in the Triangulation).
 *
 * To specify a curve in the dual 1-skeleton, we specify the
 * 2-cells which it intersects in the original Triangulation.
 * Specifically, for each Tetrahedron we keep four Boolean
 * flags which say whether the given dual curve intersects
 * each of the Tetrahedron's four faces.
 *
 * Note that a dual curve is described in terms of a specific
 * Triangulation. If the Triangulation is modified, the
 * description of the dual curve becomes meaningless.
 *
 * The file SnapPea.h contains the "opaque typedef"
 *
 *     typedef struct DualOneSkeletonCurve      DualOneSkeletonCurve;
 *
 * which lets the UI declare and pass pointers to DualOneSkeletonCurves
 * without actually knowing what they are. This file provides the kernel
 * with the actual definition.
 */

#ifndef _dual_one_skeleton_curve_
#define _dual_one_skeleton_curve_

#include "SnapPea.h"

/*
 * An array of four Booleans represents the intersection
 * of a dual curve with the four faces of a given
 * Tetrahedron. The i-th Boolean is TRUE iff the dual
 * curve passes through the Tetrahedron's i-th face.
 */

typedef Boolean DualOneSkeletonCurvePiece[4];

/*
 * An array of DualOneSkeletonCurvePieces represents
 * a complete curve in the dual 1-skeleton. The i-th
 * element in the array describes the curve's intersection
 * with the Triangulation's i-th Tetrahedron.
 *
 * Note that this definition relies on the Tetrahedra
 * being consecutively indexed, e.g. by the kernel function
 * number_the_tetrahedra(). The numbering is that of
 * the indices rather than the position in the doubly-
 * linked list (normally the two will be the same, but
 * we don't assume this).
 */

struct DualOneSkeletonCurve
{
    /*
     * tet_intersection will contain the address of
     * an array of n DualOneSkeletonCurvePieces, where n
     * is the number of Tetrahedra in the Triangulation.
     * tet_intersection[i][j] will be TRUE iff the dual
     * curve passes through face j of Tetrahedron i.
     */
    DualOneSkeletonCurvePiece *tet_intersection;

```

```
/*
 * Is this curve orientation_reversing or orientation_preserving?
 */
MatrixParity                parity;

/*
 * The length field will contain the complex length of
 * the geodesic in the homotopy class of the dual curve.
 * length[complete] and length[filled] give the length
 * relative to the complete and filled hyperbolic
 * structures, respectively.
 */
Complex                    length[2];

/*
 * The size field will contain the number
 * of segments in the curve.
 */
int                        size;

/*
 * We'll be working with large numbers of DualOneSkeletonCurves,
 * many of which will be homotopic to each other, so for efficiency
 * in sorting them out we'll keep them on a binary tree, keyed by
 * complex length (i.e. keyed by length.real, with length.imag
 * considered in case of ties). The next_subtree field is used
 * locally for tree-handling operations to avoid doing recursions
 * on the system stack; the latter run the risk of stack/heap
 * collisions.
 */
DualOneSkeletonCurve      *left_child,
                          *right_child,
                          *next_subtree;

};

#endif
```