

```

/*
 * isometry_closed.c
 *
 * This file provides the function
 *
 *      FuncResult compute_closed_isometry( Triangulation  *manifold0,
 *                                          Triangulation  *manifold1,
 *                                          Boolean        *are_isometric);
 *
 * If compute_closed_isometry() determines with absolute rigor that
 * manifold0 and manifold1 are isometric, it sets *are_isometric
 * to TRUE and returns func_OK.
 *
 * If it determines with absolute rigor that manifold0 and manifold1 are
 * nonhomeomorphic, sets *are_isometric to FALSE and returns func_OK.
 * [But at present this case doesn't occur -- see below.]
 *
 * If it fails to decide, it returns func_failed.
 *
 * AT PRESENT compute_closed_isometry() WILL NEVER REPORT THE MANIFOLDS
 * TO BE NONISOMETRIC. It relies on compute_isometries() to detect
 * different numbers of cusps or different first homology.
 * This insures that the reported results are always 100% rigorous,
 * whenever they are reported at all.
 *
 * Technical details: In the interest of speed and robustness,
 * compute_closed_isometry() does not at present use a length spectrum.
 * So if it drills the unique geodesic of a given length from each of
 * the two manifolds and finds the complements are nonhomeomorphic, it
 * will not report the nonhomeomorphism. If need be, this capability
 * could be added later.
 */

#include "kernel.h"

#define MAX_DUAL_CURVE_LENGTH 8

/*
 * We can afford to make LENGTH_EPSILON and TORSION_EPSILON large,
 * because the only danger is a loss of speed resulting from unnecessary
 * comparisons, and even that is unlikely. In particular,
 * compute_closed_isometry() will never report incorrect results,
 * no matter how large LENGTH_EPSILON and TORSION_EPSILON are.
 */
#define LENGTH_EPSILON 1e-3
#define TORSION_EPSILON 1e-3

static Boolean manifolds_are_isometric(Triangulation *original_manifold0, Triangulation *
original_manifold1, DualOneSkeletonCurve *curve0, DualOneSkeletonCurve *curve1);
static void change_Deahn_filling_description(Triangulation **manifold,
DualOneSkeletonCurve *curve);

FuncResult compute_closed_isometry(
Triangulation *manifold0,
Triangulation *manifold1,
Boolean *are_isometric)
{
    int num_curves0,
        num_curves1;
    DualOneSkeletonCurve **the_curves0,
        **the_curves1;
    int singularity_index;
    Complex length0,
        length1;
    int i,
        j;

    /*
     * We assume the calling function (e.g. compute_isometries())
     * has checked that the manifolds are not obviously nonhomeomorphic,
     * so we don't repeat the check here.
     */

```

```

    * We also assume that the manifolds have one cusp each,
    * and are Dehn filled.
    */

if (get_num_cusps(manifold0) != 1
    || all_cusps_are_filled(manifold0) == FALSE
    || all_Deohn_coefficients_are_relatively_prime_integers(manifold0) == FALSE

    || get_num_cusps(manifold1) != 1
    || all_cusps_are_filled(manifold1) == FALSE
    || all_Deohn_coefficients_are_relatively_prime_integers(manifold1) == FALSE)
{
    uFatalError("compute_closed_isometry", "isometry_closed");
}

/*
 * For later convenience, change the bases on the cusps
 * so that the Dehn filling curves become meridians.
 */
{
    MatrixInt22      basis_change[1];

    current_curve_basis(manifold0, 0, basis_change[0]);
    change_peripheral_curves(manifold0, basis_change);

    current_curve_basis(manifold1, 0, basis_change[0]);
    change_peripheral_curves(manifold1, basis_change);
}

/*
 * See what curves are drillable.
 */
dual_curves(manifold0, MAX_DUAL_CURVE_LENGTH, &num_curves0, &the_curves0);
dual_curves(manifold1, MAX_DUAL_CURVE_LENGTH, &num_curves1, &the_curves1);

/*
 * Compare each drillable curve in manifold0 (including the core
 * geodesic of the given Dehn filling description) to each drillable
 * curve in manifold1 (including the core geodesic of the given
 * Dehn filling description).  If the complex lengths match, drill
 * out each curve (if it's not already the core geodesic) and see
 * whether the complements match by a meridian-preserving isometry.
 *
 * In the following code, the case i = -1 (resp. j = -1) handles
 * the original core geodesic of manifold0 (resp. manifold1).
 */

*are_isometric = FALSE;      /* assume FALSE until proven TRUE */

for (i = -1; i < num_curves0 && *are_isometric == FALSE; i++)
{
    /*
     * Get the length of curve i in manifold0.
     */
    if (i == -1)      /* get the length of the core geodesic */
        core_geodesic(manifold0, 0, &singularity_index, &length0, NULL);
    else
        /* get the length of the_curves0[i] */
        get_dual_curve_info(the_curves0[i], NULL, &length0, NULL);

    for (j = -1; j < num_curves1 && *are_isometric == FALSE; j++)
    {
        /*
         * Get the length of curve j in manifold1.
         */
        if (j == -1)      /* get the length of the core geodesic */
            core_geodesic(manifold1, 0, &singularity_index, &length1, NULL);
        else
            /* get the length of the_curves1[j] */
            get_dual_curve_info(the_curves1[j], NULL, &length1, NULL);

        /*
         * If the lengths and absolute values of torsions match,
         * drill out the corresponding curves and check for a
         * meridian-preserving isometry.
         */
    }
}

```

```

        if (fabs(      length0.real  -      length1.real ) < LENGTH_EPSILON
            && fabs(fabs(length0.imag) - fabs(length1.imag)) < TORSION_EPSILON)

            if (manifolds_are_isometric(
                manifold0,
                manifold1,
                (i != -1) ? the_curves0[i] : NULL,
                (j != -1) ? the_curves1[j] : NULL)
                == TRUE)

                *are_isometric = TRUE;
    }
}

/*
 * Free the lists of drillable curves.
 */
free_dual_curves(num_curves0, the_curves0);
free_dual_curves(num_curves1, the_curves1);

if (*are_isometric == TRUE)
    return func_OK;
else
    return func_failed;
}

static Boolean manifolds_are_isometric(
    Triangulation      *original_manifold0,
    Triangulation      *original_manifold1,
    DualOneSkeletonCurve *curve0,
    DualOneSkeletonCurve *curve1)
{
    /*
     * manifolds_are_isometric() returns
     *
     *      TRUE   if the manifolds are definitely isometric, or
     *      FALSE  if it can't tell.
     *
     * It never reports a definite nonisometry.
     */

    Triangulation      *manifold0,
                      *manifold1;
    IsometryList      *isometry_list,
                      *isometry_list_of_links;
    Boolean            result;

    /*
     * Make copies of the manifolds so we don't trash the originals.
     */
    copy_triangulation(original_manifold0, &manifold0);
    copy_triangulation(original_manifold1, &manifold1);

    /*
     * Drill out the given curves if necessary.
     *
     * If manifold0 (resp. manifold1) requires no drilling,
     * curve0 (resp. curve1) will be NULL.
     *
     * If change_Deohn_filling_description() fails, it frees the manifold
     * and sets the pointer to NULL.
     */
    change_Deohn_filling_description(&manifold0, curve0);
    change_Deohn_filling_description(&manifold1, curve1);

    /*
     * Check for a failure.
     */
    if (manifold0 == NULL || manifold1 == NULL)
    {
        free_triangulation(manifold0); /* NULL is OK */
        free_triangulation(manifold1); /* NULL is OK */
    }
}

```

```

    return FALSE;
}

/*
 * Have we got an isometry?
 */
if (compute_cusped_isometries( manifold0,
                               manifold1,
                               &isometry_list,
                               &isometry_list_of_links) == func_OK)
{
    result = (isometry_list_of_links->num_isometries > 0);

    free_isometry_list(isometry_list);
    free_isometry_list(isometry_list_of_links);
}
else
{
    result = FALSE;

    free_triangulation(manifold0);
    free_triangulation(manifold1);

    return result;
}

static void change_Deohn_filling_description(
    Triangulation      **manifold,
    DualOneSkeletonCurve *curve)
{
    Triangulation      *new_manifold;
    Boolean            fill_cusp[2] = {TRUE, FALSE};

    /*
     * compute_closed_isometry() pass NULL to manifolds_are_isometric()
     * to indicate that it should keep the existing core curve, and
     * manifolds_are_isometric() pass that value on to us.
     */
    if (curve == NULL)
        return;

    /*
     * Drill out the indicated curve.
     */
    new_manifold = drill_cusp(*manifold, curve, "no name");
    free_triangulation(*manifold);
    *manifold = new_manifold;
    if (*manifold == NULL)
        return;
    new_manifold = NULL;

    /*
     * Set the new Dehn filling coefficient to (1, 0)
     * to recover the closed manifold.
     */
    set_cusp_info(*manifold, 1, FALSE, 1.0, 0.0);
    do_Deohn_filling(*manifold);

    /*
     * Permanently fill the original cusp.
     */
    new_manifold = fill_cusps(*manifold, fill_cusp, "no name", FALSE);
    free_triangulation(*manifold);
    *manifold = new_manifold;
    new_manifold = NULL;

    /*
     * *manifold may or may not be NULL, but we've done our best
     * so we return. (Actually, fill_cusps() is unlikely to fail.)
     */
}

```