

```

/*
 * change_peripheral_curves.c
 *
 * This file provides the function
 *
 *      FuncResult change_peripheral_curves(
 *          Triangulation *manifold,
 *          CONST MatrixInt22  change_matrices[]);
 *
 * If all the change_matrices (there should be one for each Cusp) have
 * determinant +1, then on each Cusp change_peripheral_curves() replaces
 *
 * the meridian with  change_matrices[i][0][0] meridians plus
 *                    change_matrices[i][0][1] longitudes, and
 * the longitude with  change_matrices[i][1][0] meridians plus
 *                    change_matrices[i][1][1] longitudes,
 *
 * where i is the index of the Cusp.  It returns func_OK.
 *
 * If some change_matrix has determinant != +1, change_peripheral_curves()
 * returns func_bad_input.
 */

#include "kernel.h"

FuncResult change_peripheral_curves(
    Triangulation *manifold,
    CONST MatrixInt22  change_matrices[])
{
    int          i,
                v,
                f,
                old_m,
                old_l;
    double       old_m_coef, /* changed from int to double, JRW 2000/01/18 */
                old_l_coef;
    Tetrahedron *tet;
    Cusp         *cusp;
    Complex       old_Hm,
                old_Hl;

    /*
     * First make sure all the change_matrices have determinant +1.
     */

    for (i = 0; i < manifold->num_cusps; i++)
        if (DET2(change_matrices[i]) != +1)
            return func_bad_input;

    /*
     * The change_matrices for Klein bottle cusps must have zeros in the
     * off-diagonal entries.  (Nothing else makes sense topologically.)
     */

    for (cusp = manifold->cusp_list_begin.next;
         cusp != &manifold->cusp_list_end;
         cusp = cusp->next)

        if (cusp->topology == Klein_cusp)

            for (i = 0; i < 2; i++)

                if (change_matrices[cusp->index][i][!i] != 0)

                    uFatalError("change_peripheral_curves", "change_peripheral_curves");

    /*
     * Change the peripheral curves according to the change_matrices.
     * As stated at the top of this file, the transformation rule is
     *
     *      | new m |   |   |   change_matrices[i]   |   | old m |
     *      | new l |   |   |   |   |   |   |   |   |   | old l |
     *
     */

```

```

for (tet = manifold->tet_list_begin.next;
    tet != &manifold->tet_list_end;
    tet = tet->next)

    for (i = 0; i < 2; i++)                /* which orientation */

        for (v = 0; v < 4; v++)            /* which vertex */

            for (f = 0; f < 4; f++)        /* which side */
            {
                old_m = tet->curve[M][i][v][f];
                old_l = tet->curve[L][i][v][f];

                tet->curve[M][i][v][f]
                    = change_matrices[tet->cuspid[v]->index][0][0] * old_m
                      + change_matrices[tet->cuspid[v]->index][0][1] * old_l;
                tet->curve[L][i][v][f]
                    = change_matrices[tet->cuspid[v]->index][1][0] * old_m
                      + change_matrices[tet->cuspid[v]->index][1][1] * old_l;
            }

/*
* Change the Dehn filling coefficients to reflect the new
* peripheral curves. That is, we want the Dehn filling curves
* to be topologically the same as before, even though the
* coefficients will be different because we changed the basis.
*
* To keep our thinking straight, let's imagine all peripheral
* curves -- old and new -- in terms of some arbitrary but
* fixed basis for  $\pi_1(T^2) = \mathbb{Z} + \mathbb{Z}$ . We never actually compute
* such a basis, but it helps keep our thinking straight.
* Relative to this fixed basis, we have
*
*
*         old m = (old m [0],  old m [1])
*         old l = (old l [0],  old l [1])
*         new m = (new m [0],  new m [1])
*         new l = (new l [0],  new l [1])
*
* Note that these m's and l's are curves, not coefficients!
* They are elements of  $\pi_1(T^2) = \mathbb{Z} + \mathbb{Z}$ .
*
* We can then rewrite the above transformation rule, with
* each peripheral curve (old m, old l, new m and new l)
* appearing as a row in a 2 x 2 matrix:
*
*
*         | <--new m--> |   |   | <--old m--> |
*         | <--new l--> |   |   | <--old l--> |
*         |               |   |   |
*         |               |   |   |
*
* We can invert the change_matrix to solve for the old curves
* in terms of the new ones:
*
*
*         | <--old m--> |   |   | <--new m--> |
*         | <--old l--> |   |   | <--new l--> |
*         |               |   |   |
*         |               |   |   |
*
* The Dehn filling curve is
*
*
*         old_m_coef * old_m + old_l_coef * old_l
*
*         = old_m_coef * [ change_matrices[i][1][1] * new_m
*                           - change_matrices[i][0][1] * new_l ]
*         + old_l_coef * [ - change_matrices[i][1][0] * new_m
*                           + change_matrices[i][0][0] * new_l ]
*
*         = new_m * [ old_m_coef * change_matrices[i][1][1]
*                       - old_l_coef * change_matrices[i][1][0] ]
*         + new_l * [ - old_m_coef * change_matrices[i][0][1]
*                       + old_l_coef * change_matrices[i][0][0] ]
*
* Therefore
*
*         new_m_coef = old_m_coef * change_matrices[i][1][1]

```

```

*          - old_l_coef * change_matrices[i][1][0]
*      new_l_coef = - old_m_coef * change_matrices[i][0][1]
*                  + old_l_coef * change_matrices[i][0][0]
*/

for (cusp = manifold->cusp_list_begin.next;
     cusp != &manifold->cusp_list_end;
     cusp = cusp->next)

    if (cusp->is_complete == FALSE)
    {
        old_m_coef = cusp->m;
        old_l_coef = cusp->l;

        cusp->m =  old_m_coef * change_matrices[cusp->index][1][1]
                  - old_l_coef * change_matrices[cusp->index][1][0];
        cusp->l = - old_m_coef * change_matrices[cusp->index][0][1]
                  + old_l_coef * change_matrices[cusp->index][0][0];
    }

/*
*   Update the holonomies according to the rule
*
*       | new H(m) |   |
*       | new H(l) | = | change_matrices[i] | | old H(m) |
*                   | | old H(l) |
*
*   (These are actually logs of holonomies, so it's correct to
*   add -- not multiply -- them.)
*
*   For complete Cusps the holonomies should be zero, but that's OK.
*/

for (cusp = manifold->cusp_list_begin.next;
     cusp != &manifold->cusp_list_end;
     cusp = cusp->next)

    for (i = 0; i < 2; i++)      /* i = ultimate, penultimate */
    {
        old_Hm = cusp->holonomy[i][M];
        old_Hl = cusp->holonomy[i][L];

        cusp->holonomy[i][M] = complex_plus(
            complex_real_mult(
                change_matrices[cusp->index][0][0],
                old_Hm
            ),
            complex_real_mult(
                change_matrices[cusp->index][0][1],
                old_Hl
            )
        );

        cusp->holonomy[i][L] = complex_plus(
            complex_real_mult(
                change_matrices[cusp->index][1][0],
                old_Hm
            ),
            complex_real_mult(
                change_matrices[cusp->index][1][1],
                old_Hl
            )
        );
    }

/*
*   Update the cusp_shapes.
*/

for (cusp = manifold->cusp_list_begin.next;
     cusp != &manifold->cusp_list_end;
     cusp = cusp->next)
{
    cusp->cusp_shape[initial] = transformed_cusp_shape(

```

```
        cusp->cusp_shape[initial],
        change_matrices[cusp->index]);

    if (cusp->is_complete == TRUE)
        cusp->cusp_shape[current] = transformed_cusp_shape(
            cusp->cusp_shape[current],
            change_matrices[cusp->index]);

    /*
     * else cusp->cusp_shape[current] == Zero, and needn't be changed.
     */
}

return func_OK;
}
```