

```

/*
 * core_geodesics.c
 *
 * This file provides the function
 *
 * void core_geodesic( Triangulation *manifold,
 *                    int cusp_index,
 *                    int singularity_index,
 *                    Complex *core_length,
 *                    int *precision);
 *
 * which computes the core_length and singularity_index for the
 * Cusp of index cusp_index in Triangulation *manifold. The
 * singularity_index describes the nature of the cusp:
 *
 * If the Cusp is unfilled or the Dehn filling coefficients are
 * not integers, then
 *
 *     *singularity_index is set to zero,
 *     *core_length      is undefined.
 *
 * If the Cusp is filled and the Dehn filling coefficients are
 * relatively prime integers (so we have a manifold locally), then
 *
 *     *singularity_index is set to one,
 *     *core_length      is set to the complex length
 *                       of the central geodesic,
 *
 * If the Cusp is filled and the Dehn filling coefficients are
 * non relatively prime integers (so we have an orbifold locally), then
 *
 *     *singularity_index is set to the index of the singular locus,
 *     *core_length      is set to the complex length of the central
 *                       geodesic in the smallest manifold cover of
 *                       a neighborhood of the singular set.
 *
 * If the precision pointer is not NULL, *precision is set to the
 * number of decimal places of accuracy in the computed value of
 * core_length.
 *
 * Klein bottle cusps are OK. Their torsion will always be zero.
 *
 * This file also provides the function
 *
 * void compute_core_geodesic( Cusp *cusp,
 *                             int *singularity_index,
 *                             Complex length[2]);
 *
 * for use within the kernel. It is similar to core_geodesic(),
 * only it takes a Cusp pointer as input, and outputs the complex
 * lengths relative to the ultimate and penultimate hyperbolic
 * structures, rather than reporting a precision.
 *
 * The Algorithm.
 *
 * Say we're doing (p, q) Dehn filling on some Cusp. The (closed) core
 * geodesic lifts to a set of (infinite) geodesics in the universal cover.
 * Let L be one such (infinite) geodesic in the universal cover, and
 * consider the group G of covering transformations fixing L (setwise).
 * G is generated by the holonomies of the meridian and longitude,
 * which we denote H(m) and H(l), subject to the single relation
 *  $p H(m) + q H(l) = 0$ . We would like to find new generators g and h
 * for the group such that the relation takes the form  $n g + 0 h = 0$ .
 * The generator g will be the purely rotational part of the group G
 * (n will be the order of the singular locus), and h will generate
 * the translational part of G.
 *
 * One approach to finding g and h would be to proceed in the general
 * context of finitely generated abelian groups, and apply the Euclidean
 * algorithm to the relation  $p H(m) + q H(l) = 0$ , changing bases until
 * the relation simplifies down to something of the form  $n g + 0 h = 0$ .
 * But rather than messing with the bookkeeping of this approach, we'll
 * use a more pedestrian method.

```

```

*
* Let (a, b; c, d) be the matrix expressing (g, h) in terms
* of (H(m), H(l)):
*
*      | g |   | a b |   | H(m) |
*      | h |   | c d |   | H(l) |
*
* Because  $0 = n g = n (a H(m) + b H(l)) = na H(m) + nb H(l)$  is
* the identity, and  $p H(m) + q H(l)$  is the only relation in the
* presentation of the group, it follows that (a, b) must be
* proportional to (p, q). Furthermore, since the matrix (a, b; c, d)
* has determinant one, a and b must be relatively prime, so therefore
*  $(a, b) = (p, q)/\gcd(p, q)$ . It now follows that c and d are integers
* satisfying
*       $1 = a d - b c = d p / \gcd(p, q) - c q / \gcd(p, q)$ 
*       $\Leftrightarrow$ 
*       $d p - c q = \gcd(p, q)$ 
*
* We can find such integers using SnapPea's standard
* euclidean_algorithm() function.
*/

```

```
#include "kernel.h"
```

```
#define TORSION_EPSILON 1e-5
```

```

void core_geodesic(
    Triangulation *manifold,
    int cusp_index,
    int singularity_index,
    Complex *core_length,
    int *precision)
{
    Cusp *cusp;
    Complex length[2];

    cusp = find_cusp(manifold, cusp_index);

    /*
     * Compute the complex length relative to the ultimate
     * and penultimate hyperbolic structures.
     */

    compute_core_geodesic(cusp, singularity_index, length);

    /*
     * Package up the results.
     */

    if (*singularity_index != 0)
    {
        *core_length = length[ultimate];

        if (precision != NULL)
            *precision = complex_decimal_places_of_accuracy(
                length[ultimate], length[penultimate]);
    }
    else
    {
        *core_length = Zero;

        if (precision != NULL)
            *precision = 0;
    }
}

```

```

void compute_core_geodesic(
    Cusp *cusp,
    int singularity_index,
    Complex length[2])
{

```

```

int         i;
long int    positive_d,
            negative_c;
double      pi_over_n;

/*
 * If the Cusp is unfilled or the Dehn filling coefficients aren't
 * integers, then just write in some zeros (as explained at the top
 * of this file) and return.
 */

if (cusp->is_complete == TRUE
    || Dehn_coefficients_are_integers(cusp) == FALSE)
{
    *singularity_index = 0;
    length[ultimate] = Zero;
    length[penultimate] = Zero;

    return;
}

/*
 * The euclidean_algorithm() will give the singularity index
 * directly (as the g.c.d.), and the coefficients lead to the
 * complex length (cf. the explanation at the top of this file).
 */

*singularity_index = euclidean_algorithm(
                        (long int) cusp->m,
                        (long int) cusp->l,
                        &positive_d,
                        &negative_c);

for (i = 0; i < 2; i++)      /* i = ultimate, penultimate */
{
    /*
     * length[i] = c H(m) + d H(l)
     *
     * (The holonomies are already in logarithmic form.)
     */
    length[i] = complex_plus(
        complex_real_mult(
            (double) (- negative_c),
            cusp->holonomy[i][M]
        ),
        complex_real_mult(
            (double) positive_d,
            cusp->holonomy[i][L]
        )
    );

    /*
     * Make sure the length is positive.
     */
    if (length[i].real < 0.0)
        length[i] = complex_negate(length[i]);

    /*
     * We want to normalize the torsion to the range
     * [-pi/n + epsilon, pi/n + epsilon], where n is
     * the order of the singular locus.
     */

    pi_over_n = PI / *singularity_index;

    while (length[i].imag < - pi_over_n + TORSION_EPSILON)
        length[i].imag += 2 * pi_over_n;

    while (length[i].imag >  pi_over_n + TORSION_EPSILON)
        length[i].imag -= 2 * pi_over_n;

    /*
     * In the case of a Klein bottle cusp, H(m) will be purely
     * rotational and H(l) will be purely translational

```

```
    * (cf. the documentation at the top of holonomy.c).
    * But the longitude used in practice is actually the
    * double cover of the true longitude, so we have to
    * divide the core_length by two to compensate.
    */
    if (cusp->topology == Klein_cusp)
        length[i].real /= 2.0;
}
```