

```

/*
 * filling.c
 *
 * This file contains the functions
 *
 *      Triangulation  *fill_cusps(Triangulation  *manifold,
 *                                Boolean          fill_cusp[],
 *                                char            *new_name,
 *                                Boolean          fill_all_cusps);
 *
 *      Triangulation  *fill_reasonable_cusps(Triangulation *manifold);
 *
 *      Boolean        cusp_is_fillable(Cusp *cusp);
 *      Boolean        is_closed_manifold(Triangulation *manifold);
 *
 * which the kernel provides to the UI.
 *
 * fill_cusps() permanently fills k of the cusps of an n-cusp manifold.
 * It returns an ideal Triangulation of the resulting (n - k)-cusp manifold.
 *
 * 99/06/04 Previous versions of fill_cusps() insisted that at least
 * one cusp be left unfilled. The current version allows all cusps
 * to be filled, in which case it produces a finite triangulation.
 * Warning: Most SnapPea functions insist on an ideal triangulation --
 * the finite triangulation is provided mainly for writing to disk.
 *
 * Arguments:
 *
 *      manifold        is the original manifold. The Dehn filling
 *                      coefficients cusp->m and cusp->l specify how
 *                      each cusp is to be filled.
 *
 *      fill_cusp        says which cusps are to be filled. The cusp
 *                      of index i will be filled iff fill_cusp[i] is TRUE.
 *                      If fill_all_cusps (see below) is TRUE, then
 *                      fill_cusp is ignored and may be NULL.
 *
 *      new_name        provides the name for the new Triangulation.
 *
 *      fill_all_cusps  says whether to fill all the cusps, producing
 *                      a triangulation with finite vertices only.
 *                      Usually fill_all_cusps is FALSE.
 *
 * The UI should decide how to present fill_cusps() to the user.
 * Should all currently Dehn filled cusps be filled at once?
 * Should the user be presented with a list of check boxes to
 * specify which cusps to fill? Should cusps be filled one at a time?
 * My hope is that fill_cusps() is sufficiently general to support
 * whatever approach the UI developer prefers.
 *
 * Having said that, let me now mention fill_reasonable_cusps(), which
 * makes a decision about which cusps to fill, and then makes a call
 * to fill_cusp(). fill_reasonable_cusps() will fill all cusps which
 * have relatively prime Dehn filling coefficients, unless this would
 * leave no unfilled cusps, in which case it leaves cusp 0 unfilled.
 * It copies the name from the manifold being filled.
 *
 * cusp_is_fillable() determines whether an individual cusp is fillable.
 *
 * The original manifold is always left unaltered.
 *
 * The files subdivide.c, close_cusps.c, and remove_finite_vertices.c
 * document the algorithm in detail.
 */

#include "kernel.h"

static Boolean  check_fill_cusp_array(Triangulation *manifold, Boolean fill_cusp[]);
static Boolean  cusp_is_fillable_x(Cusp *cusp);
static Boolean  no_cusps_to_be_filled(int num_cusps, Boolean fill_cusp[]);

Triangulation *fill_cusps(
    Triangulation *manifold,

```

```

    Boolean      fill_cusp[],
    char         *new_name,
    Boolean      fill_all_cusps)
{
    Triangulation *new_triangulation;
    Boolean      at_least_one_cusp_is_left;
    Boolean      *all_true;
    int          i;

    /*
     * 95/10/1 JRW
     * The following algorithm works correctly even if no cusps are
     * to be filled, but we can speed it up a bit by simply copying
     * the Triangulation.
     */
    if (fill_all_cusps == FALSE
        && no_cusps_to_be_filled(manifold->num_cusps, fill_cusp) == TRUE)
    {
        copy_triangulation(manifold, &new_triangulation);
        return new_triangulation;
    }

    /*
     * First let's do a little error checking on the fill_cusp[] array.
     */
    if (fill_all_cusps == FALSE)
    {
        /*
         * Check that Dehn filling coefficients are relatively prime integers,
         * and also that at least one cusp is left unfilled.
         */
        at_least_one_cusp_is_left = check_fill_cusp_array(manifold, fill_cusp);
        if (at_least_one_cusp_is_left == FALSE)
            uFatalError("fill_cusps", "filling");
    }
    else
    {
        /*
         * Check that Dehn filling coefficients are relatively prime integers.
         */
        all_true = NEW_ARRAY(manifold->num_cusps, Boolean);
        for (i = 0; i < manifold->num_cusps; i++)
            all_true[i] = TRUE;
        (void) check_fill_cusp_array(manifold, all_true);
        /*
         * Do NOT free all_true just yet.
         */
    }

    /*
     * Subdivide the triangulation, introducing finite vertices.
     * Note that the original triangulation is left unharmed.
     */
    new_triangulation = subdivide(manifold, new_name);

    /*
     * Close the Cusps specified in the fill_cusp[] array.
     */
    close_cusps(new_triangulation, fill_all_cusps ? all_true : fill_cusp);

    /*
     * We're done with the all_true array.
     */
    if (fill_all_cusps == TRUE)
        my_free(all_true);

    /*
     * Retriangulate with no finite vertices.
     */
    if (fill_all_cusps == FALSE)
        remove_finite_vertices(new_triangulation); /* includes basic_simplification() */
    else
        basic_simplification(new_triangulation);
}

```

```

/*
 * If the old manifold had a hyperbolic structure,
 * try to find one for the new_triangulation as well.
 */
if (fill_all_cusps == FALSE
    && manifold->solution_type[complete] != not_attempted)
{
    find_complete_hyperbolic_structure(new_triangulation);
    do_Dehn_filling(new_triangulation);

    /*
     * If the old manifold had a known Chern-Simons invariant,
     * pass it to the new_triangulation.
     */
    if (manifold->CS_value_is_known == TRUE)
    {
        new_triangulation->CS_value_is_known      = manifold->CS_value_is_known;
        new_triangulation->CS_value[ultimate]      = manifold->CS_value[ultimate];
        new_triangulation->CS_value[penultimate]    = manifold->CS_value[penultimate];

        /*
         * The solution_type may or may not be good enough to compute
         * the fudge factor, but we'll let compute_CS_fudge_from_value()
         * worry about that.
         */
        compute_CS_fudge_from_value(new_triangulation);
    }
}

return new_triangulation;
}

Triangulation *fill_reasonable_cusps(
    Triangulation *manifold)
{
    Boolean      *fill_cusp;
    Cusp         *cusp;
    int          i;
    Boolean      all_cusps_are_fillable;
    Triangulation *new_triangulation;

    /*
     * Allocate the fill_cusp[] array.
     */

    fill_cusp = NEW_ARRAY(manifold->num_cusps, Boolean);

    /*
     * See which cusps are fillable.
     */

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

        fill_cusp[cusp->index] = cusp_is_fillable_x(cusp);

    /*
     * If all the cusps are fillable, leave cusp 0 unfilled.
     */

    all_cusps_are_fillable = TRUE;

    for (i = 0; i < manifold->num_cusps; i++)
        if (fill_cusp[i] == FALSE)
            all_cusps_are_fillable = FALSE;

    if (all_cusps_are_fillable == TRUE)
        fill_cusp[0] = FALSE;

    /*
     * Call fill_cusps().
     */
}

```

```

    new_triangulation = fill_cusps(manifold, fill_cusp, manifold->name, FALSE);

    /*
     * Free the fill_cusp[] array.
     */

    my_free(fill_cusp);

    /*
     * Done.
     */

    return new_triangulation;
}

static Boolean check_fill_cusp_array(
    Triangulation *manifold,
    Boolean fill_cusp[])
{
    Boolean at_least_one_cusp_is_left;
    Cusp *cusp;

    at_least_one_cusp_is_left = FALSE;

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

        if (fill_cusp[cusp->index])
        {
            if (cusp_is_fillable_x(cusp) == FALSE)
                uFatalError("check_fill_cusp_array", "filling");
        }
        else
            at_least_one_cusp_is_left = TRUE;

    return at_least_one_cusp_is_left;
}

Boolean cusp_is_fillable(
    Triangulation *manifold,
    int cusp_index)
{
    return cusp_is_fillable_x(find_cusp(manifold, cusp_index));
}

static Boolean cusp_is_fillable_x(
    Cusp *cusp)
{
    return( cusp->is_complete == FALSE
        && Dehn_coefficients_are_relatively_prime_integers(cusp) == TRUE);
}

static Boolean no_cusps_to_be_filled(
    int num_cusps,
    Boolean fill_cusp[])
{
    int i;

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

        if (fill_cusp[i] == TRUE)

            return FALSE;

    return TRUE;
}

```

```
Boolean is_closed_manifold(  
    Triangulation *manifold)  
{  
    return (all_cusps_are_filled(manifold)  
        && all_Deohn_coefficients_are_relatively_prime_integers(manifold));  
}
```