

```

/*
 * cusps.c
 *
 * This file contains the functions
 *
 * void create_cusps(Triangulation *manifold);
 * void create_fake_cusps(Triangulation *manifold);
 * void count_cusps(Triangulation *manifold);
 * Boolean mark_fake_cusps(Triangulation *manifold);
 *
 * create_cusps() is used within the kernel to assign Cusp data
 * structures to Triangulations. It assumes the neighbor and gluing
 * fields have been set (all other fields are optional). It assigns
 * cusp indices, but does not install peripheral curves, determine
 * the CuspTopologies, or count the cusps. You should call
 * peripheral_curves() to install the peripheral curves and determine
 * the CuspTopologies, then call count_cusps() to set num_cusps,
 * num_or_cusps and num_nonor_cusps.
 *
 * create_fake_cusps() is used within the kernel to assign Cusp data
 * structures to the "fake cusps" corresponding to finite vertices.
 * It assumes fake cusps are indicated by tet->cusp[v] fields of NULL.
 * The fake cusps are numbered -1, -2, etc. As explained in the
 * documentation at the top of finite_vertices.c, finite vertices use
 * only the is_finite, index, prev and next fields of the Cusp data
 * structure. create_fake_cusps() does not disturb the real cusps or
 * the non-NULL tet->cusp[v] fields.
 *
 * count_cusps() counts the Cusps of each CuspTopology, and sets
 * manifold->num_cusps, manifold->num_or_cusps and manifold->num_nonor_cusps.
 *
 * mark_fake_cusps() distinguishes real cusps from fake cusps
 * ( = finite vertices) by computing the Euler characteristic.
 * Sets is_finite to TRUE for fake cusps, and renumbers all cusps so that
 * real cusps have consecutive nonnegative indices beginning at 0 and
 * fake cusps have consecutive negative indices beginning at -1.
 * Returns TRUE if fake cusps are present, FALSE otherwise.
 */

#include "kernel.h"

typedef struct
{
    Tetrahedron *tet;
    VertexIndex v;
} IdealVertex;

static void compute_cusp_Euler_characteristics(Triangulation *manifold);

void create_cusps(
    Triangulation *manifold)
{
    int count;
    Tetrahedron *tet;
    VertexIndex v;

    /*
     * Make sure no Cusps are present, and everything is neat and tidy.
     */

    error_check_for_create_cusps(manifold);

    /*
     * The variable "count" will keep track of the next index
     * to be assigned. The first Cusp we create will have
     * index 0, the next will have 1, and so on.
     */

    count = 0;

    /*

```

```

    * We look at each vertex of each Tetrahedron, and whenever we
    * encounter a vertex with no assigned Cusp, we create a Cusp
    * for it and recursively assign it to neighboring ideal vertices.
    */

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

        for (v = 0; v < 4; v++)

            if (tet->cusp[v] == NULL)

                create_one_cusp(manifold, tet, FALSE, v, count++);
}

void error_check_for_create_cusps(
    Triangulation *manifold)
{
    Tetrahedron *tet;
    VertexIndex v;

    if (manifold->num_cusps != 0
        || manifold->num_or_cusps != 0
        || manifold->num_nonor_cusps != 0
        || manifold->cusp_list_begin.next != &manifold->cusp_list_end)

        uFatalError("error_check_for_create_cusps", "cusps");

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

        for (v = 0; v < 4; v++)

            if (tet->cusp[v] != NULL)

                uFatalError("error_check_for_create_cusps", "cusps");
}

void create_fake_cusps(
    Triangulation *manifold)
{
    int count;
    Tetrahedron *tet;
    VertexIndex v;

    /*
    * The variable "count" will keep track of the (negative) index
    * most recently assigned. The first finite vertex we create
    * will have index -1, the next will have -2, and so on.
    */

    count = 0;

    /*
    * We look at each vertex of each Tetrahedron, and whenever we
    * encounter an ideal vertex with no assigned Cusp, we create a Cusp
    * for it and assign it recursively to neighboring ideal vertices.
    */

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

        for (v = 0; v < 4; v++)

            if (tet->cusp[v] == NULL)

                create_one_cusp(manifold, tet, TRUE, v, --count);
}

```

```

void create_one_cusp(
    Triangulation *manifold,
    Tetrahedron *tet,
    Boolean is_finite,
    VertexIndex v,
    int cusp_index)
{
    Cusp *cusp;
    IdealVertex *queue;
    int queue_first,
        queue_last;
    Tetrahedron *tet1,
        *nbr;
    VertexIndex v1,
        nbr_v;
    FaceIndex f;

    /*
     * Create the cusp, add it to the list, and set
     * the is_finite and index fields.
     */

    cusp = NEW_STRUCT(Cusp);
    initialize_cusp(cusp);
    INSERT_BEFORE(cusp, &manifold->cusp_list_end);
    cusp->is_finite = is_finite;
    cusp->index = cusp_index;

    /*
     * We don't set the topology, is_complete, m, l, holonomy,
     * cusp_shape or shape_precision fields.
     *
     * For "real" cusps the calling routine may
     *
     * (1) call peripheral_curves() to set the cusp->topology,
     *
     * (2) keep the default values of cusp->is_complete,
     *     cusp->m and cusp->l as set by initialize_cusp(), and
     *
     * (3) let the holonomy and cusp_shape be computed automatically
     *     when hyperbolic structure is computed.
     *
     * Alternatively, the calling routine may set these fields in other
     * ways, as it sees fit.
     *
     * If we were called by create_fake_cusps(), then the above fields
     * are all irrelevant and ignored.
     */

    /*
     * Set the tet->cusp field at all vertices incident to the new cusp.
     */

    /*
     * Allocate space for a queue of pointers to the IdealVertices.
     * Each IdealVertex will appear on the queue at most once, so an
     * array of length 4 * manifold->num_tetrahedra will suffice.
     */
    queue = NEW_ARRAY(4 * manifold->num_tetrahedra, IdealVertex);

    /*
     * Set the cusp of the given IdealVertex...
     */
    tet->cusp[v] = cusp;

    /*
     * ...and put it on the queue.
     */
    queue_first = 0;
    queue_last = 0;
    queue[0].tet = tet;
    queue[0].v = v;

```

```

/*
 * Start processing the queue.
 */
do
{
    /*
     * Pull an IdealVertex off the front of the queue.
     */
    tet1 = queue[queue_first].tet;
    v1   = queue[queue_first].v;
    queue_first++;

    /*
     * Look at the three neighboring IdealVertices.
     */
    for (f = 0; f < 4; f++)
    {
        if (f == v1)
            continue;

        nbr    = tet1->neighbor[f];
        nbr_v  = EVALUATE(tet1->gluing[f], v1);

        /*
         * If the neighbor's cusp hasn't been set...
         */
        if (nbr->cusp[nbr_v] == NULL)
        {
            /*
             * ...set it...
             */
            nbr->cusp[nbr_v] = cusp;

            /*
             * ...and add it to the end of the queue.
             */
            ++queue_last;
            queue[queue_last].tet = nbr;
            queue[queue_last].v   = nbr_v;
        }
    }
} while (queue_first <= queue_last);

/*
 * Free the memory used for the queue.
 */
my_free(queue);
}

```

```

void count_cusps(
    Triangulation *manifold)
{
    Cusp *cusp;

    manifold->num_cusps      = 0;
    manifold->num_or_cusps   = 0;
    manifold->num_nonor_cusps = 0;

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

        switch (cusp->topology)
        {
            case torus_cusp:
                manifold->num_or_cusps++;
                break;

            case Klein_cusp:

```

```

        manifold->num_nonor_cusps++;
        break;

    default:
        uFatalError("count_cusps", "cusps");
    }
}

Boolean mark_fake_cusps(
    Triangulation *manifold)
{
    int      real_cusp_count,
            fake_cusp_count;
    Cusp     *cusp;

    compute_cusp_Euler_characteristics(manifold);

    real_cusp_count = 0;
    fake_cusp_count = 0;

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

        switch (cusp->euler_characteristic)
        {
            case 0:
                cusp->is_finite = FALSE;
                cusp->index = real_cusp_count++;
                break;

            case 2:
                cusp->is_finite = TRUE;
                cusp->index = --fake_cusp_count;
                break;

            default:
                uFatalError("mark_fake_cusps", "cusps");
        }

    return (fake_cusp_count < 0);
}

static void compute_cusp_Euler_characteristics(
    Triangulation *manifold)
{
    Cusp      *cusp;
    EdgeClass *edge;
    Tetrahedron *tet;
    VertexIndex v,
                v0,
                v1;

    /*
     * Initialize all Euler characteristics to zero.
     */

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

        cusp->euler_characteristic = 0;

    /*
     * It'll be easier to count the edges twice (once from each side)
     * so compute twice the Euler characteristic and divide by two
     * at the end.
     */

    /*
     * Count the vertices in the triangulation of the boundary,

```

```

    *   which come from edges in the manifold itself.
    */

    for (edge = manifold->edge_list_begin.next;
         edge != &manifold->edge_list_end;
         edge = edge->next)
    {
        tet = edge->incident_tet;
        v0 = one_vertex_at_edge[edge->incident_edge_index];
        v1 = other_vertex_at_edge[edge->incident_edge_index];
        tet->cusp[v0]->euler_characteristic += 2;
        tet->cusp[v1]->euler_characteristic += 2;
    }

    /*
    *   Count the edges in the triangulation of the boundary,
    *   which come from faces in the manifold itself.
    */

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

        for (v = 0; v < 4; v++)

            tet->cusp[v]->euler_characteristic -= 3;

    /*
    *   Count the faces in the triangulation of the boundary,
    *   which come from tetrahedra in the manifold itself.
    */

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

        for (v = 0; v < 4; v++)

            tet->cusp[v]->euler_characteristic += 2;

    /*
    *   Divide by two (cf. above).
    */

    for (cusp = manifold->cusp_list_begin.next;
         cusp != &manifold->cusp_list_end;
         cusp = cusp->next)
    {
        if (cusp->euler_characteristic % 2 != 0)
            uFatalError("compute_cusp_Euler_characteristics", "cusps");
        cusp->euler_characteristic /= 2;
    }
}

```