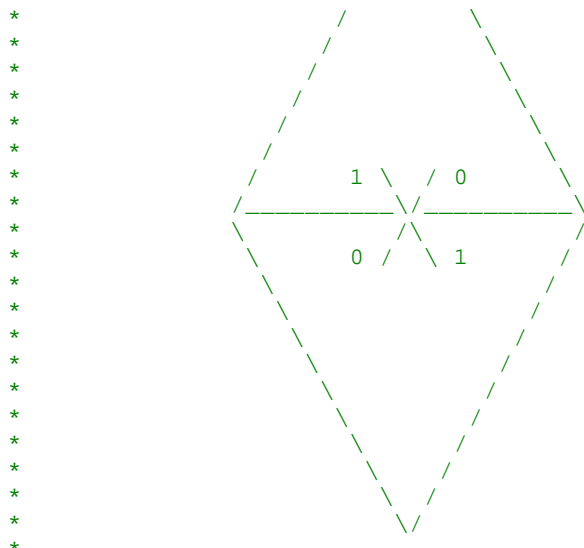


```

/*
 * intersection_numbers.c
 *
 * This file provides the kernel function
 *
 * void compute_intersection_numbers(Triangulation *manifold);
 *
 * which computes the intersection numbers of the curves stored
 * in the scratch_curve[][][][] fields of the Tetrahedra, and
 * writes the results to the intersection_number[][] fields of
 * the Cusps. That is,
 *
 * intersection_number[M][M] will be the intersection number of
 * scratch_curve[0][M] with scratch_curve[1][M],
 *
 * intersection_number[M][L] will be the intersection number of
 * scratch_curve[0][M] with scratch_curve[1][L],
 *
 * intersection_number[L][M] will be the intersection number of
 * scratch_curve[0][L] with scratch_curve[1][M],
 *
 * intersection_number[L][L] will be the intersection number of
 * scratch_curve[0][L] with scratch_curve[1][L].
 *
 * Each intersection number is the algebraic sum of the crossing
 * numbers. As viewed from infinity (looking toward the fat
 * part of the manifold), each crossing of the form
 *
 * scratch_curve[1]
 *   ^
 *   |
 * ---|---> scratch_curve[0]      contributes +1,
 *   |
 *
 * while each crossing of the form
 *
 * scratch_curve[0]
 *   ^
 *   |
 * ---|---> scratch_curve[1]      contributes -1,
 *   |
 *
 * This file also provides the utility
 *
 * void copy_curves_to_scratch(    Triangulation *manifold,
 *                                int           which_set,
 *                                Boolean        double_copy_on_tori);
 *
 * which copies the current peripheral curves to the scratch_curves[which_set]
 * fields of the manifold's Tetrahedra. If double_copy_on_tori is TRUE,
 * it copies peripheral curves on orientable cusps to both sheets of
 * the Cusps' orientation double covers.
 *
 * Overview of Intersection Number Algorithm.
 *
 * Consider the triangulation of the boundary components by the
 * triangles at the (truncated) ideal vertices. As explained
 * in peripheral_curves.c, we work in the orientation double cover,
 * so in fact each ideal vertex contributes two triangles, one
 * left_handed and the other right_handed. Relative to the
 * orientation on the cusp (and even nonorientable cusps are
 * effectively oriented, since we work in the orientation double
 * cover) we imagine each scratch_curve[0] entering a given triangle
 * on the right side of a given edge, and each scratch_curve[1]
 * entering on the left:
 *
 *
 *      /\
 *     /\
 *    /\
 *   /\
 *  /\
 * /\

```



\* Of necessity, the curves must cross on the edge (if both are  
 \* nonzero). There may be additional crossings in the interior of  
 \* the triangle, depending on where the various curves are entering  
 \* and exiting. To avoid counting the intersections on the edges  
 \* twice (once for each of the two incident triangles) we make the  
 \* convention to count edge crossings only where scratch\_curve[0] is  
 \* entering (not exiting) the triangle.

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

```

void compute_intersection_numbers(
    Triangulation *manifold)
{
    Cusp *cusp;
    Tetrahedron *tet;
    int f,
        g,
        h,
        i,
        j,
        face_on_the_left,
        face_on_the_right;

    /*
     * Initialize all the intersection numbers to zero.
     */

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

        for (i = 0; i < 2; i++) /* i = M, L */

            for (j = 0; j < 2; j++) /* j = M, L */

                cusp->intersection_number[i][j] = 0;

    /*
     * Count the intersections on the edges.
     */

    /* which Tetrahedron */
    for (tet = manifold->tet_list_begin.next;
         tet != &manifold->tet_list_end;
         tet = tet->next)

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

            /* which side of the vertex */
            for (j = 0; j < 4; j++)

```

```

    {
        if (i == j)
            continue;

        /* which sheet (right_handed or left_handed) */
        for (f = 0; f < 2; f++)

            /* which scratch_curve[0] (meridian or longitude) */
            for (g = 0; g < 2; g++)

                /* which scratch_curve[1] (meridian or longitude) */
                for (h = 0; h < 2; h++)

                    /*
                     * Recall the convention (described at the top
                     * of this file) that edge crossings are counted
                     * only where scratch_curve[0] is entering --
                     * not exiting -- the triangle.
                     */
                    if (tet->scratch_curve[0][g][f][i][j] > 0)

                        tet->cusp[i]->intersection_number[g][h]
                            += tet->scratch_curve[0][g][f][i][j]
                                * tet->scratch_curve[1][h][f][i][j];
    }

/*
 * Count the intersections in the interiors of triangles.
 */

/* which Tetrahedron */
for (tet = manifold->tet_list_begin.next;
    tet != &manifold->tet_list_end;
    tet = tet->next)

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

        /* which side of the vertex */
        for (j = 0; j < 4; j++)
        {
            if (i == j)
                continue;

            /*
             * Name the two remaining faces of the Tetrahedron
             * according to the right_handed orientation of
             * the Tetrahedron.
             */
            face_on_the_left    = remaining_face[i][j];
            face_on_the_right   = remaining_face[j][i];

            /* which scratch_curve[0] (meridian or longitude) */
            for (g = 0; g < 2; g++)

                /* which scratch_curve[1] (meridian or longitude) */
                for (h = 0; h < 2; h++)
                {
                    /*
                     * We'll count only those intersections on the
                     * strand of scratch_curve[0] running from the
                     * current side of the triangle (side j) towards
                     * the right. The other possibilities will be
                     * handled by other values of j.
                     *
                     * When we see the Tetrahedron as left_handed
                     * relative to the Orientation of the cusp, the
                     * face on the right is face_on_the_left.
                     * Got that?
                     */

                    tet->cusp[i]->intersection_number[g][h]
                        += FLOW(tet->scratch_curve[0][g][right_handed][i][j],

```

