

```

/*
 * tet_shapes.c
 *
 * This file contains the following low-level functions for
 * working with TetShapes.
 *
 * void add_edge_angles( Tetrahedron *tet0, EdgeIndex e0,
 *                      Tetrahedron *tet1, EdgeIndex e1,
 *                      Tetrahedron *tet2, EdgeIndex e2)
 *
 * Boolean angles_sum_to_zero( Tetrahedron *tet0, EdgeIndex e0,
 *                             Tetrahedron *tet1, EdgeIndex e1);
 *
 * void compute_remaining_angles(Tetrahedron *tet, EdgeIndex e);
 *
 * add_edge_angles() adds the edge angles at edge e0 of tet0
 * to the corresponding angles at edge e1 of tet1, and writes
 * the results to edge e2 of tet2. It pays careful attention
 * to the edge_orientations. Note that even though opposite
 * edges of a Tetrahedron have equal angles, they needn't have
 * the same edge_orientation, so you should pass an actual
 * EdgeIndex in the range 0-5, not merely a quasi-equivalent
 * index in the range 0-2. The edge angle of the sum will be
 * in the range  $[(-1/2)\pi, (3/2)\pi]$ , regardless of the angles
 * of the summands.
 *
 * angles_sum_to_zero() returns TRUE iff one of the angles
 * (shape[complete]->cwl[ultimate] or shape[filled]->cwl[ultimate])
 * at edge e0 of tet0 cancels the corresponding angle at edge e1
 * of tet1 (mod  $2\pi$ ). Accounts for edge_orientations.
 *
 * compute_remaining_angles() assumes the angle at edge e is
 * correct, and computes the remaining angles in terms of it.
 * The EdgeIndex may be given in either the range 0-5 or the
 * range 0-2. The arguments of the remaining angles will be
 * in the range  $[(-1/2)\pi, (3/2)\pi]$ .
 */

#include "kernel.h"

/*
 * CANCELLATION_EPSILON says how close the logs of two complex
 * numbers must be to be considered equal.
 */

#define CANCELLATION_EPSILON 1e-4

static void add_tet_shapes(
    TetShape *ts0, EdgeIndex e30, Orientation eo0,
    TetShape *ts1, EdgeIndex e31, Orientation eo1,
    TetShape *ts2, EdgeIndex e32, Orientation eo2);
static void add_complex_with_log(
    ComplexWithLog *cwl0, Orientation eo0,
    ComplexWithLog *cwl1, Orientation eo1,
    ComplexWithLog *cwl2, Orientation eo2);
static Boolean logs_sum_to_zero(
    Complex summand0, Orientation eo0,
    Complex summand1, Orientation eo1);
static void compute_cwl(ComplexWithLog cwl[3], EdgeIndex e);
static void normalize_angle(double *angle);

void add_edge_angles(
    Tetrahedron *tet0, EdgeIndex e0,
    Tetrahedron *tet1, EdgeIndex e1,
    Tetrahedron *tet2, EdgeIndex e2)
{
    int i;

    for (i = 0; i < 2; i++) /* i = complete, filled */
        add_tet_shapes(
            tet0->shape[i], edge3[e0], tet0->edge_orientation[e0],

```

```

        tet1->shape[i], edge3[e1], tet1->edge_orientation[e1],
        tet2->shape[i], edge3[e2], tet2->edge_orientation[e2]);
}

```

```

static void add_tet_shapes(
    TetShape *ts0,  EdgeIndex e30,  Orientation eo0,
    TetShape *ts1,  EdgeIndex e31,  Orientation eo1,
    TetShape *ts2,  EdgeIndex e32,  Orientation eo2)
{
    int i;

    for (i = 0; i < 2; i++)      /* i = ultimate, penultimate */
        add_complex_with_log(
            &ts0->cwl[i][e30], eo0,
            &ts1->cwl[i][e31], eo1,
            &ts2->cwl[i][e32], eo2);
}

```

```

static void add_complex_with_log(
    ComplexWithLog *cwl0,  Orientation eo0,
    ComplexWithLog *cwl1,  Orientation eo1,
    ComplexWithLog *cwl2,  Orientation eo2)
{
    /*
     * First compute the sum of the logs, then recover
     * the rectangular form.
     *
     * We do all computations relative to the Orientation
     * of the EdgeClass. So if a particular edge is seen
     * as left_handed by the EdgeClass, we must negate the
     * real part of the log of its complex angle. (Recall
     * that all all TetShapes are stored relative to the
     * right_handed Orientation of the Tetrahedron.)
     */

    Complex summand0,
           summand1,
           sum;

    summand0 = cwl0->log;
    if (eo0 == left_handed)
        summand0.real = - summand0.real;

    summand1 = cwl1->log;
    if (eo1 == left_handed)
        summand1.real = - summand1.real;

    sum = complex_plus(summand0, summand1);
    if (eo2 == left_handed)
        sum.real = - sum.real;

    normalize_angle(&sum.imag);

    cwl2->log = sum;
    cwl2->rect = complex_exp(sum);
}

```

```

Boolean angles_sum_to_zero(
    Tetrahedron *tet0,  EdgeIndex e0,
    Tetrahedron *tet1,  EdgeIndex e1)
{
    int i;

    for (i = 0; i < 2; i++)      /* i = complete, filled */

        if (logs_sum_to_zero(
            tet0->shape[i]->cwl[ultimate][edge3[e0]].log,
            tet0->edge_orientation[e0],
            tet1->shape[i]->cwl[ultimate][edge3[e1]].log,
            tet1->edge_orientation[e1]))

```

```

        return TRUE;

    return FALSE;
}

static Boolean logs_sum_to_zero(
    Complex summand0, Orientation eo0,
    Complex summand1, Orientation eo1)
{
    Complex sum;

    if (eo0 != eo1)
        summand1.real = - summand1.real;

    sum = complex_plus(summand0, summand1);

    normalize_angle(&sum.imag);

    return (complex_modulus(sum) < CANCELLATION_EPSILON);
}

void compute_remaining_angles(
    Tetrahedron *tet,
    EdgeIndex e)
{
    int i,
        j;

    for (i = 0; i < 2; i++)          /* i = complete, filled      */
        for (j = 0; j < 2; j++)      /* j = ultimate, penultimate */
            compute_cwl(tet->shape[i]->cwl[j], edge3[e]);
}

static void compute_cwl(
    ComplexWithLog cwl[3],
    EdgeIndex e)
{
    /*
     * Compute cwl[(e+1)%3] and cwl[(e+2)%3] in terms of cwl[e].
     */

    int i;

    for (i = 1; i < 3; i++)
    {
        cwl[(e+i)%3].rect = complex_div(One, complex_minus(One, cwl[(e+i-1)%3].rect));
        cwl[(e+i)%3].log = complex_log(cwl[(e+i)%3].rect, PI_OVER_2);
    }
}

static void normalize_angle(
    double *angle)
{
    /*
     * Normalize the angle to lie in the range  $[(-1/2)\pi, (3/2)\pi]$ .
     */

    while (*angle > THREE_PI_OVER_2)
        *angle -= TWO_PI;

    while (*angle < - PI_OVER_2)
        *angle += TWO_PI;
}

```