

```

/*
 * gluing_equations.c
 *
 * This file provides the function
 *
 * void compute_gluing_equations(Triangulation *manifold);
 *
 * which the function do_Dehn_filling() in hyperbolic_structure.c calls
 * to compute the edge and cusp equations and their derivatives.
 * It computes complex gluing equations for oriented manifolds, and
 * real gluing equations for nonoriented manifolds. It assumes that
 * space for the equations has already been assigned to the cusps and
 * edges, and that a coordinate system has been chosen for each
 * tetrahedron (cf. allocate_equations() and choose_coordinate_system()
 * in hyperbolic_structures.c).
 */

#include "kernel.h"

static void initialize_gluing_equations(Triangulation *manifold);
static void compute_derivative(Triangulation *manifold);
static void compute_rhs(Triangulation *manifold);

void compute_gluing_equations(
    Triangulation *manifold)
{
    compute_holonomies(manifold);
    compute_edge_angle_sums(manifold);
    initialize_gluing_equations(manifold);
    compute_derivative(manifold);
    compute_rhs(manifold);
}

static void initialize_gluing_equations(
    Triangulation *manifold)
{
    EdgeClass *edge;
    Cusp *cusp;
    int i;

    /*
     * Initialize edge equations.
     */

    for (edge = manifold->edge_list_begin.next;
         edge != &manifold->edge_list_end;
         edge = edge->next)

        for (i = 0; i < manifold->num_tetrahedra; i++)

            if (manifold->orientability == oriented_manifold)
                edge->complex_edge_equation[i] = Zero;
            else
            {
                edge->real_edge_equation_re[2*i] = 0.0;
                edge->real_edge_equation_re[2*i + 1] = 0.0;
                edge->real_edge_equation_im[2*i] = 0.0;
                edge->real_edge_equation_im[2*i + 1] = 0.0;
            }

    /*
     * Initialize cusp equations.
     */

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

        for (i = 0; i < manifold->num_tetrahedra; i++)

            if (manifold->orientability == oriented_manifold)
                cusp->complex_cusp_equation[i] = Zero;

```

```

else
{
    cusp->real_cusp_equation_re[2*i] = 0.0;
    cusp->real_cusp_equation_re[2*i + 1] = 0.0;
    cusp->real_cusp_equation_im[2*i] = 0.0;
    cusp->real_cusp_equation_im[2*i + 1] = 0.0;
}
}

/*
 * The coordinate system for the parameter space of each
 * tetrahedron has already been chosen as explained in
 * the comment preceding the function choose_coordinate_system()
 * in the file hyperbolic_structure.c. The derivatives computed
 * in that comment may be expressed as
 *
 *
 * d(log z0)          d(log z0)          d(log z0)   -1
 * ----- = 1        ----- = -z2       ----- = --
 * d(log z0)          d(log z1)          d(log z2)   z1
 *
 * d(log z1)   -1      d(log z1)          d(log z1)
 * ----- = --         ----- = 1        ----- = -z0
 * d(log z0)    z2      d(log z1)          d(log z2)
 *
 * d(log z2)          d(log z2)   -1      d(log z2)
 * ----- = -z1       ----- = --        ----- = 1
 * d(log z0)          d(log z1)    z0      d(log z2)
 *
 * compute_derivative() uses these forms to compute the entries
 * of the derivative matrix. If the manifold is oriented, these complex
 * numbers are added directly to the appropriate entries in the matrix.
 * If the manifold is unoriented, each complex number (a + bi) is
 * converted to a 2 x 2 real matrix
 *
 *
 *           a   -b
 *
 *           b    a
 *
 * This matrix mimics the action of the complex derivative. That is,
 * (a + bi)(dx + i dy) = (a dx - b dy) + i(b dx + a dy), and
 *
 *
 * | a dx - b dy |   | a   -b |   | dx |
 * | b dx + a dy |   | b    a |   | dy |
 *
 *
 * If the Tetrahedron is seen as right_handed by its EdgeClass, then
 * the above matrix is added directly to the appropriate 2 x 2 block
 * in the derivative matrix. If the Tetrahedron is seen as left_handed
 * by it EdgeClass, then we must account for the fact that the EdgeClass
 * sees the conjugate-inverse of the edge parameter. That is, the
 * imaginary part of the log (i.e. the angle) will be the same, but
 * the real part of the log (i.e. the compression/expansion factor)
 * will be negated. We therefore use the following matrix instead.
 *
 *
 *      -a   b
 *
 *      b    a
 */

static void compute_derivative(
Triangulation *manifold)
{
Tetrahedron *tet;
Complex      z[3],
             d[3],
             *eqn_coef = NULL,
             dz[2];
EdgeIndex    e;
VertexIndex v;
FaceIndex    initial_side,
             terminal_side;
int          init[2][2],

```

```

        term[2][2];
double    m,
          l,
          a,
          b,
          *eqn_coef_00 = NULL,
          *eqn_coef_01 = NULL,
          *eqn_coef_10 = NULL,
          *eqn_coef_11 = NULL;
int       i,
          j;

for (tet = manifold->tet_list_begin.next;
     tet != &manifold->tet_list_end;
     tet = tet->next)
{
    /*
     * Note the three edge parameters.
     */

    for (i = 0; i < 3; i++)
        z[i] = tet->shape[filled]->cwl[ultimate][i].rect;

    /*
     * Set the derivatives of log(z0), log(z1) and log(z2)
     * with respect to the given coordinate system, as
     * indicated by the above table.
     */

    switch (tet->coordinate_system)
    {
        case 0:
            d[0] = One;
            d[1] = complex_div(MinusOne, z[2]);
            d[2] = complex_minus(Zero, z[1]);
            break;

        case 1:
            d[0] = complex_minus(Zero, z[2]);
            d[1] = One;
            d[2] = complex_div(MinusOne, z[0]);
            break;

        case 2:
            d[0] = complex_div(MinusOne, z[1]);
            d[1] = complex_minus(Zero, z[0]);
            d[2] = One;
            break;
    }

    /*
     * Record this tetrahedron's contribution to the edge equations.
     */

    for (e = 0; e < 6; e++)      /* Look at each of the six edges. */
    {
        /*
         * Find the matrix entry(ies) corresponding to the
         * derivative of the edge equation with respect to this
         * tetrahedron. If the manifold is oriented it will be
         * a single entry in the complex matrix. If the manifold
         * is unoriented it will be a 2 x 2 block in the real matrix.
         */

        if (manifold->orientability == oriented_manifold)
            eqn_coef = &tet->edge_class[e]->complex_edge_equation[tet->index];
        else
        {
            eqn_coef_00 = &tet->edge_class[e]->real_edge_equation_re[2 * tet->index];
            eqn_coef_01 = &tet->edge_class[e]->real_edge_equation_re[2 * tet->index + 1];
            eqn_coef_10 = &tet->edge_class[e]->real_edge_equation_im[2 * tet->index];
            eqn_coef_11 = &tet->edge_class[e]->real_edge_equation_im[2 * tet->index + 1];
        }
    }
}

```

```

1];
    }

    /*
    * Add in the derivative of the log of the edge parameter
    * with respect to the chosen coordinate system. Please
    * see the comment preceding this function for details.
    */

    if (manifold->orientability == oriented_manifold)

        *eqn_coef = complex_plus(*eqn_coef, d[edge3[e]]);

    else
    {
        /*
        * These are the same a and b as in the comment
        * preceding this function.
        */

        a = d[edge3[e]].real;
        b = d[edge3[e]].imag;

        if (tet->edge_orientation[e] == right_handed)
        {
            *eqn_coef_00 += a;
            *eqn_coef_01 -= b;
            *eqn_coef_10 += b;
            *eqn_coef_11 += a;
        }
        else
        {
            *eqn_coef_00 -= a;
            *eqn_coef_01 += b;
            *eqn_coef_10 += b;
            *eqn_coef_11 += a;
        }
    }
}

/*
* Record this tetrahedron's contribution to the cusp equations.
*/

for (v = 0; v < 4; v++) /* Look at each ideal vertex. */
{
    /*
    * Note the Dehn filling coefficients on this cusp.
    * If the cusp is complete, use m = 1.0 and l = 0.0.
    */

    if (tet->cusp[v]->is_complete)
    {
        m = 1.0;
        l = 0.0;
    }
    else
    {
        m = tet->cusp[v]->m;
        l = tet->cusp[v]->l;
    }

    /*
    * Find the matrix entry(ies) corresponding to the
    * derivative of the cusp equation with respect to this
    * tetrahedron. If the manifold is oriented it will be
    * a single entry in the complex matrix. If the manifold
    * is unoriented it will be a 2 x 2 block in the real matrix.
    */

    if (manifold->orientability == oriented_manifold)
        eqn_coef = &tet->cusp[v]->complex_cusp_equation[tet->index];
    else

```

```

{
    eqn_coef_00 = &tet->cuspid[v]->real_cusp_equation_re[2 * tet->index];
    eqn_coef_01 = &tet->cuspid[v]->real_cusp_equation_re[2 * tet->index + 1];
    eqn_coef_10 = &tet->cuspid[v]->real_cusp_equation_im[2 * tet->index];
    eqn_coef_11 = &tet->cuspid[v]->real_cusp_equation_im[2 * tet->index + 1];
}

/*
 * Each ideal vertex contains two triangular cross sections,
 * one right_handed and the other left_handed. We want to
 * compute the contribution of each angle of each triangle
 * to the holonomy. We begin by considering the right_handed
 * triangle, looking at each of its three angles. A directed
 * angle is specified by its initial and terminal sides.
 * We find the number of strands of the Dehn filling curve
 * passing from the initial side to the terminal side;
 * it is m * (number of strands of meridian)
 * + l * (number of strands of longitude), where (m,l) are
 * the Dehn filling coefficients (in practice, m and l need
 * not be integers, but it's simpler to imagine them to be
 * integers as you try to understand the following code).
 * The number of strands of the Dehn filling curves passing
 * from the initial to the terminal side is multiplied by
 * the derivative of the log of the complex angle, to yield
 * the contribution to the derivative matrix. If the manifold
 * is oriented, that complex number is added directly to
 * the relevant matrix entry. If the manifold is unoriented,
 * we convert the complex number to a 2 x 2 real matrix
 * (cf. the comments preceding this function) and add it to
 * the appropriate 2 x 2 block of the real derivative matrix.
 * The 2 x 2 matrix for the left_handed triangle is modified
 * to account for the fact that although the real part of the
 * derivative of the log (i.e. the compression/expansion
 * factor) is the same, the imaginary part (i.e. the rotation)
 * is negated. [Note that in computing the edge equations
 * the real part was negated, while for the cusp equations
 * the imaginary part is negated. I will leave an explanation
 * of the difference as an exercise for the reader.]
 *
 * Note that we cannot possibly handle curves on the
 * left_handed sheet of the orientation double cover of
 * a cusp of an oriented manifold. The reason is that the
 * log of the holonomy of the Dehn filling curve is not
 * a complex analytic function of the shape of the tetrahedron
 * (it's the complex conjugate of such a function). I.e.
 * it doesn't have a derivative in the complex sense. This
 * is why we make the convention that all peripheral curves
 * in oriented manifolds lie on the right_handed sheet of
 * the double cover.
 */

for (initial_side = 0; initial_side < 4; initial_side++)
{
    if (initial_side == v)
        continue;

    terminal_side = remaining_face[v][initial_side];

    /*
     * Note the intersection numbers of the meridian and
     * longitude with the initial and terminal sides.
     */

    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++) {
            /* which curve */
            /* which sheet */
            init[i][j] = tet->curve[i][j][v][initial_side];
            term[i][j] = tet->curve[i][j][v][terminal_side];
        }
    }

    /*
     * For each triangle (right_handed and left_handed),
     * multiply the number of strands of the Dehn filling
     * curve running from initial_side to terminal_side

```

```

    * by the derivative of the log of the edge parameter.
    */

    for (i = 0; i < 2; i++) /* which sheet */
        dz[i] = complex_real_mult(
            m * FLOW(init[M][i],term[M][i]) + 1 * FLOW(init[L][i],term[L][i])
            ,
            d[ edge3_between_faces[initial_side][terminal_side] ]
        );

    /*
    * If the manifold is oriented, the Dehn filling curve
    * must lie of the right_handed sheet of the orientation
    * double cover (cf. above). Add its contribution to
    * the cusp equation.
    */

    if (manifold->orientability == oriented_manifold)

        *eqn_coef = complex_plus(*eqn_coef, dz[right_handed]);

    /* "else" follows below */

    /*
    * If the manifold is unoriented, treat the right_ and
    * left_handed sheets separately. Add in the contribution
    * of the right_handed sheet normally. For the left_handed
    * sheet, we must account for the fact that even though
    * the modulus of the derivative (i.e. the expansion/
    * contraction factor) is correct, its argument (i.e. the
    * angle of rotation) is the negative of what it should be.
    */

    else
    {
        a = dz[right_handed].real;
        b = dz[right_handed].imag;
        *eqn_coef_00 += a;
        *eqn_coef_01 -= b;
        *eqn_coef_10 += b;
        *eqn_coef_11 += a;

        a = dz[left_handed].real;
        b = dz[left_handed].imag;
        *eqn_coef_00 += a;
        *eqn_coef_01 -= b;
        *eqn_coef_10 -= b;
        *eqn_coef_11 -= a;
    }
}

}

}

/*
* compute_complex_rhs() assumes that compute_holonomies() and
* compute_edge_angle_sums() have already been called.
*/

static void compute_rhs(
    Triangulation *manifold)
{
    EdgeClass *edge;
    Cusp *cusp;
    Complex desired_holonomy,
        current_holonomy,
        rhs;

    /*
    * The right hand side of each equation will be the desired value
    * of the edge angle sum or the holonomy (depending on whether it's
    * an edge equation or a cusp equation) minus the current value.
    */

```

```

    * Thus, when the equations are solved and the Shapes of the
    * Tetrahedra are updated, the edge angle sums and the holonomies
    * will take on their desired values, to the accuracy of the
    * linear approximation.
    */

/*
    * Set the right hand side (rhs) of each edge equation.
    */

for (    edge = manifold->edge_list_begin.next;
        edge != &manifold->edge_list_end;
        edge = edge->next)
{
    /*
    * The desired value of the sum of the logs of the complex
    * edge parameters is 2 pi i. The current value is
    * edge->edge_angle_sum.
    */

    rhs = complex_minus(TwoPiI, edge->edge_angle_sum);

    if (manifold->orientability == oriented_manifold)
        edge->complex_edge_equation[manifold->num_tetrahedra] = rhs;
    else
    {
        edge->real_edge_equation_re[2 * manifold->num_tetrahedra] = rhs.real;
        edge->real_edge_equation_im[2 * manifold->num_tetrahedra] = rhs.imag;
    }
}

/*
    * Set the right hand side (rhs) of each cusp equation.
    */

for (cusp = manifold->cusp_list_begin.next;
     cusp != &manifold->cusp_list_end;
     cusp = cusp->next)
{
    /*
    * For complete cusps we want the log of the holonomy of the
    * meridian to be zero. For Dehn filled cusps we want the
    * log of the holonomy of the Dehn filling curve to be 2 pi i.
    */

    if (cusp->is_complete)
    {
        desired_holonomy    = Zero;
        current_holonomy    = cusp->holonomy[ultimate][M];
    }
    else
    {
        desired_holonomy    = TwoPiI;
        current_holonomy    = complex_plus(
            complex_real_mult(cusp->m, cusp->holonomy[ultimate][M]),
            complex_real_mult(cusp->l, cusp->holonomy[ultimate][L])
        );
    }

    rhs = complex_minus(desired_holonomy, current_holonomy);

    if (manifold->orientability == oriented_manifold)
        cusp->complex_cusp_equation[manifold->num_tetrahedra] = rhs;
    else
    {
        cusp->real_cusp_equation_re[2 * manifold->num_tetrahedra] = rhs.real;
        cusp->real_cusp_equation_im[2 * manifold->num_tetrahedra] = rhs.imag;
    }
}
}

```