

```

/*
 * volume.c
 *
 * This file contains the function
 *
 *     double volume(Triangulation *manifold, int *precision);
 *
 * which the kernel provides for the UI. It computes and returns
 * the volume of the manifold. If the pointer "precision" is not NULL,
 * volume() estimates the number of decimal places of accuracy, and
 * places the result in the variable *precision. The error estimate is
 * the difference in the computed volumes at the last and next-to-the-last
 * iterations of Newton's method (cf. hyperbolic_structures.c).
 *
 * 94/11/30 JRW This file now contains the function
 *
 *     double birectangular_tetrahedron_volume(      O31Vector  a,
 *                                                    O31Vector  b,
 *                                                    O31Vector  c,
 *                                                    O31Vector  d);
 *
 * as well, for use within the kernel.
 */

#include "kernel.h"

static double Lobachevsky(double theta);

double volume(
    Triangulation *manifold,
    int *precision)
{
    int i,
        j;
    double vol[2]; /* vol[ultimate/penultimate] */
    Tetrahedron *tet;

    for (i = 0; i < 2; i++) /* i = ultimate, penultimate */
        vol[i] = 0.0;

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

        if (tet->shape[filled] != NULL)

            for (i = 0; i < 2; i++) /* i = ultimate, penultimate */

                for (j = 0; j < 3; j++)

                    vol[i] += Lobachevsky(tet->shape[filled]->cwl[i][j].log.imag);

    if (precision != NULL)
        *precision = decimal_places_of_accuracy(vol[ultimate], vol[penultimate]);

    return vol[ultimate];
}

/*
 * The Lobachevsky() function is based on the formula in
 * Milnor's article "Hyperbolic geometry: The first 150 years",
 * Bulletin of the American Mathematical Society, volume 6, number 1,
 * January 1982, pp. 9-24. The actual formula appears about 2/3 of
 * the way down page 18.
 */

static double Lobachevsky(double theta)
{
    double term,
        sum,
        product,

```

```

        theta_over_pi_squared;
const double *lobcoefptr;

const static double lobcoef[30] = {
    5.4831135561607547882413838888201e-1,
    1.0823232337111381915160036965412e-1,
    4.8444907713545197129262758561472e-2,
    2.7891037672165120538296812180796e-2,
    1.8199901365960328824311744707278e-2,
    1.2823667776324462157674846128817e-2,
    9.5243928393815114745643670962394e-3,
    7.3530535460250636167039159668208e-3,
    5.8479755397266959054962366178048e-3,
    4.7619093045811136799814912001842e-3,
    3.9525701124525799515137944808229e-3,
    3.3333335320272968375315987081340e-3,
    2.8490028914574211634331659107080e-3,
    2.4630541963678177950454607261557e-3,
    2.1505376364114568439132649094016e-3,
    1.8939393943803620897114593734851e-3,
    1.680672269005391127527764855335e-3,
    1.5015015015233512340706336099638e-3,
    1.3495276653220485553945730785293e-3,
    1.2195121951230603594927151084491e-3,
    1.1074197120711266596728487987281e-3,
    1.0101010101010675186059356321779e-3,
    9.2506938020352840967144128733280e-4,
    8.5034013605442478972252664720549e-4,
    7.8431372549019677504189889653457e-4,
    7.2568940493468811469129541350086e-4,
    6.7340067340067343805464730535677e-4,
    6.2656641604010025932192218654462e-4,
    5.8445353594389246257711686275038e-4,
    5.4644808743169398954500641421420e-4};

/*
 * As long as DBL_EPSILON > 5e-22 there will be enough lobcoefs
 * for the series. If DBL_EPSILON is smaller than this you'll
 * need to add more coefficients to the list.
 */

#if (DBL_DIG > 19)
    You need to check DBL_EPSILON.
    If it is less than 5e-22 you will need to provide more lobcoefs.
#endif

/*
 * Milnor (Lemma 1, p. 17) shows that the Lobachevsky function is
 * periodic with period pi. Put theta in the range [-pi/2, +pi/2].
 */

while (theta > PI_OVER_2)
    theta -= PI;
while (theta < -PI_OVER_2)
    theta += PI;

/*
 * Milnor (Lemma 1, p. 17) also shows that the Lobachevsky function
 * is an odd function, so we can further restrict theta to the
 * range [0, pi/2].
 */

if (theta < 0.0)
    return -Lobachevsky(-theta);

/*
 * Handle theta == 0.0 specially, to avoid encountering
 * log(0.0) later on.
 */

if (theta == 0.0)
    return 0.0;

theta_over_pi_squared = (theta/PI)*(theta/PI);

```

```

    sum = 0.0;
    product = 1.0;
    lobcoefptr = lobcoef;
    do
    {
        product *= theta_over_pi_squared;
        term = *lobcoefptr++ * product;
        sum += term;
    }
    while (term > DBL_EPSILON);

    return theta*(1.0 - log(2*theta) + sum);
}

double birectangular_tetrahedron_volume(
    O31Vector  a,
    O31Vector  b,
    O31Vector  c,
    O31Vector  d)
{
    /*
     * Compute the volume of the birectangular tetrahedron with vertices
     * a, b, c and d, using the method found in section 4.3 of
     *
     *      E. B. Vinberg, Ob'emy neevklidovykh mnogogrannikov,
     *      Uspekhi Matematicheskix Nauk, May(?) 1993, 17-46.
     *
     * Our a, b, c and d correspond to Vinberg's A, B, C and D, as shown
     * in his Figure 9. We need to compute the dual basis {aa, bb, cc, dd}
     * defined by <aa, a> = 1, <aa, b> = <aa, c> = <aa, d> = 0, etc.
     * Let m be the matrix whose rows are the vectors {a, b, c, d}, but
     * with the entries in the first column negated to account for the
     * indefinite inner product, and let mm be the matrix whose columns
     * are the vectors {aa, bb, cc, dd}. Then (m)(mm) = identity, by the
     * definition of the dual basis.
     */

    GL4RMatrix  m,
                mm;
    O31Vector  aa,
                bb,
                cc,
                dd;

    double      alpha,
                beta,
                gamma,
                delta,
                big_delta,
                tetrahedron_volume;
    int         i;

    /*
     * Set up the matrix m.
     */
    for (i = 0; i < 4; i++)
    {
        m[0][i] = a[i];
        m[1][i] = b[i];
        m[2][i] = c[i];
        m[3][i] = d[i];
    }
    for (i = 0; i < 4; i++)
        m[i][0] = - m[i][0];

    /*
     * The matrix mm will be the inverse of m, as explained above.
     * When m is singular, the birectangular tetrahedron's volume is zero.
     */
    if (gl4R_invert(m, mm) != func_OK)
        return 0.0;

    /*
     * Read the dual basis {aa, bb, cc, dd} from mm.

```

```

    */
    for (i = 0; i < 4; i++)
    {
        aa[i] = mm[i][0];
        bb[i] = mm[i][1];
        cc[i] = mm[i][2];
        dd[i] = mm[i][3];
    }

    /*
     * Any pair of dual vectors lies in a positive definite 2-plane
     * in  $E^{(3,1)}$ . Normalize them to have length one, so we can use
     * their dot products to compute the dihedral angles.
     */
    o31_constant_times_vector(
        1.0 / safe_sqrt ( o31_inner_product(aa,aa) ),
        aa,
        aa);
    o31_constant_times_vector(
        1.0 / safe_sqrt ( o31_inner_product(bb,bb) ),
        bb,
        bb);
    o31_constant_times_vector(
        1.0 / safe_sqrt ( o31_inner_product(cc,cc) ),
        cc,
        cc);
    o31_constant_times_vector(
        1.0 / safe_sqrt ( o31_inner_product(dd,dd) ),
        dd,
        dd);

    /*
     * Compute the angles alpha, beta and gamma, as shown in
     * Vinberg's Figure 9.
     */
    alpha = PI - safe_acos(o31_inner_product(aa, bb));
    beta  = PI - safe_acos(o31_inner_product(bb, cc));
    gamma = PI - safe_acos(o31_inner_product(cc, dd));

    /*
     * Compute big_delta and delta, as in
     * Vinberg's Sections 4.2 and 4.3.
     */
    big_delta = sin(alpha) * sin(alpha) * sin(gamma) * sin(gamma)
        - cos(beta) * cos(beta);

    if (big_delta >= 0.0)
        uFatalError("birectangular_tetrahedron_volume", "volume");

    delta = atan( safe_sqrt( - big_delta) /
        (cos(alpha) * cos(gamma)) );

    tetrahedron_volume = 0.25 * (
        Lobachevsky(alpha + delta)
        - Lobachevsky(alpha - delta)
        + Lobachevsky(gamma + delta)
        - Lobachevsky(gamma - delta)
        - Lobachevsky(PI_OVER_2 - beta + delta)
        + Lobachevsky(PI_OVER_2 - beta - delta)
        + 2.0 * Lobachevsky(PI_OVER_2 - delta)
    );

    return tetrahedron_volume;
}

```