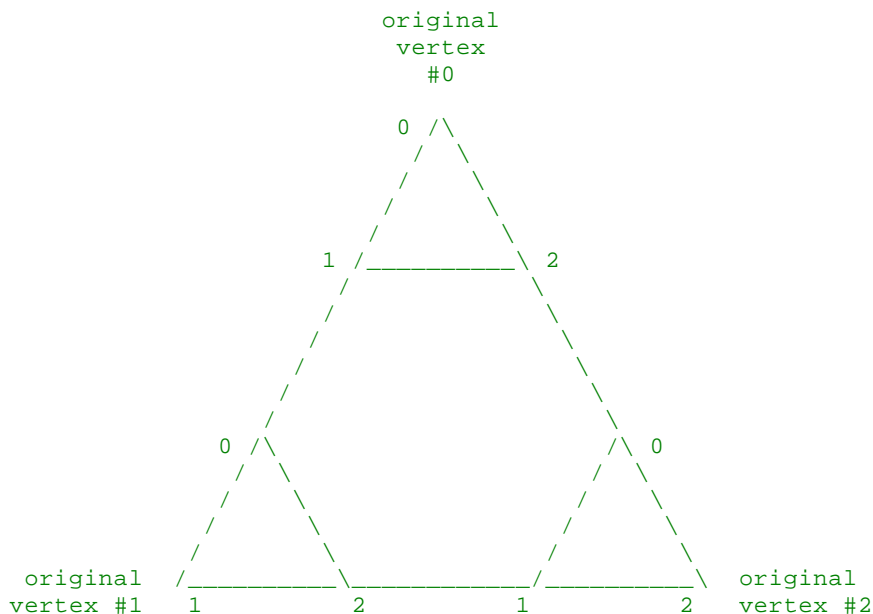


```

/*
 * subdivide.c
 *
 * This file contains the function
 *
 *     Triangulation *subdivide(Triangulation *manifold, char *new_name);
 *
 * which accepts a Triangulation *manifold, copies it, and
 * subdivides the copy as described below into a Triangulation
 * with finite as well as ideal vertices. The original
 * triangulation is not changed.
 *
 * Triangulations produced by subdivide() differ from
 * ordinary triangulations in that they have finite vertices
 * as well as ideal vertices.
 *
 * At present, only the function fill_cusps() calls subdivide(),
 * but other kernel functions could use it too, if the need arises.
 *
 * Note: subdivide() has a subtle dependence on the implementation
 * of orient(). The first Tetrahedron on the new Triangulation's
 * tet list has the correct orientation, and orient() must propagate
 * that orientation to all the other Tetrahedra.
 *
 * The function subdivide() subdivides each Tetrahedron of a
 * Triangulation as follows. First, a regular neighborhood of
 * each ideal vertex is sliced off, to form its own new
 * Tetrahedron. Each such new Tetrahedron will have three finite
 * vertices as well as the original ideal vertex. The ideal
 * vertex keeps the same VertexIndex as the original, while
 * each of the three finite vertices inherits the VertexIndex
 * of the nearest (other) ideal vertex of the original Tetrahedron.
 * I wish I could provide a good illustration, but it's hard to
 * embed 3-D graphics in ASCII files; here's a 2-D illustration
 * which shows the general idea:

```



```

* After removing the four Tetrahedra just described, you are
* left with a solid which has four hexagonal faces and four
* triangular faces. Please make yourself a sketch of a
* truncated tetrahedron to illustrate this. Each hexagonal
* face is subdivided into six triangles by coning to the
* center:

```




```

Tetrahedron *inner_vertex_tet[4];

/*
 * The "edge Tetrahedra" are the 12 Tetrahedra which have
 * precisely one edge contained within an edge of the
 * old Tetrahedron.  edge_tet[i][j] is the Tetrahedron
 * which has a face contained in face i of the old
 * Tetrahedron, on the side (of face i) opposite vertex j.
 * edge_tet[i][j] is defined iff i != j.
 */

Tetrahedron *edge_tet[4][4];

/*
 * The "face Tetrahedra" are the 12 remaining Tetrahedra.
 * face_tet[i][j] has a face contained in face i of the
 * old Tetrahedron, on the side near vertex j (of the old
 * Tetrahedron).  face_tet[i][j] is defined iff i != j.
 */

Tetrahedron *face_tet[4][4];

};

static void attach_extra(Triangulation *manifold);
static void free_extra(Triangulation *manifold);
static void create_new_tetrahedra(Triangulation *new_triangulation, Triangulation *
    old_triangulation);
static void allocate_new_tetrahedra(Triangulation *new_triangulation, Triangulation *
    old_triangulation);
static void set_outer_vertex_tets(Tetrahedron *old_tet);
static void set_inner_vertex_tets(Tetrahedron *old_tet);
static void set_edge_tets(Tetrahedron *old_tet);
static void set_face_tets(Tetrahedron *old_tet);
static void create_new_cusps(Triangulation *new_triangulation, Triangulation *
    old_triangulation);
static void create_real_cusps(Triangulation *new_triangulation, Triangulation *
    old_triangulation);

Triangulation *subdivide(
    Triangulation *old_triangulation,
    char *new_name)
{
    Triangulation *new_triangulation;

    /*
     * Allocate storage for the new_triangulation
     * and initialize it.
     *
     * (In spite of what I wrote at the top of this file,
     * we don't explicitly make a copy of *old_triangulation,
     * but instead we create *new_triangulation from scratch.)
     */

    new_triangulation = NEW_STRUCT(Triangulation);
    initialize_triangulation(new_triangulation);
    new_triangulation->name = NEW_ARRAY(strlen(new_name) + 1, char);
    strcpy(new_triangulation->name, new_name);
    new_triangulation->num_tetrahedra = 32 * old_triangulation->num_tetrahedra;
    new_triangulation->num_cusps = old_triangulation->num_cusps;
    new_triangulation->num_or_cusps = old_triangulation->num_or_cusps;
    new_triangulation->num_nonor_cusps = old_triangulation->num_nonor_cusps;

    /*
     * Attach an Extra field to each Tetrahedron in the
     * old_triangulation, to keep track of the corresponding
     * 32 Tetrahedra in the new_triangulation.
     */

    attach_extra(old_triangulation);

    /*

```

```

    * Create the new Tetrahedra.
    * Set fields such as tet->neighbor and tet->gluing,
    * which can be determined immediately.
    * Postpone determination of fields such as tet->cusp
    * and tet->edge_class.
    */

    create_new_tetrahedra(new_triangulation, old_triangulation);

    /*
    * Copy the Cusps from the old_triangulation to
    * the new_triangulation. (Technical note: functions
    * called by create_new_cusps() rely on the fact that
    * create_new_tetrahedra() initializes all tet->cusp
    * fields to NULL.)
    */

    create_new_cusps(new_triangulation, old_triangulation);

    /*
    * We're done with the Extra fields, so free them.
    */

    free_extra(old_triangulation);

    /*
    * Add the bells and whistles.
    */

    create_edge_classes(new_triangulation);
    orient_edge_classes(new_triangulation);
    orient(new_triangulation);

    /*
    * Return the new_triangulation.
    */

    return new_triangulation;
}

static void attach_extra(
    Triangulation *manifold)
{
    Tetrahedron *tet;

    for (tet = manifold->tet_list_begin.next;
         tet != &manifold->tet_list_end;
         tet = tet->next)
    {
        /*
        * Make sure no other routine is using the "extra"
        * field in the Tetrahedron data structure.
        */
        if (tet->extra != NULL)
            uFatalError("attach_extra", "filling");

        /*
        * Attach the locally defined struct extra.
        */
        tet->extra = NEW_STRUCT(Extra);
    }
}

static void free_extra(
    Triangulation *manifold)
{
    Tetrahedron *tet;

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

```

```

    /*
     * Free the struct extra.
     */
    my_free(tet->extra);

    /*
     * Set the extra pointer to NULL to let other
     * modules know we're done with it.
     */
    tet->extra = NULL;
}
}

static void create_new_tetrahedra(
    Triangulation *new_triangulation,
    Triangulation *old_triangulation)
{
    Tetrahedron *old_tet;

    /*
     * Allocate the memory for all the new Tetrahedra.
     * We do this before setting any of the fields, so
     * that the tet->neighbor fields will all have something
     * to point to.
     */

    allocate_new_tetrahedra(new_triangulation, old_triangulation);

    /*
     * For each Tetrahedron in the old_triangulation, we want
     * to set the various fields of the 32 corresponding Tetrahedra
     * in the new triangulation. The new Tetrahedra are accessed
     * through the Extra fields of the old Tetrahedra.
     */

    for (old_tet = old_triangulation->tet_list_begin.next;
         old_tet != &old_triangulation->tet_list_end;
         old_tet = old_tet->next)
    {
        set_outer_vertex_tets(old_tet);
        set_inner_vertex_tets(old_tet);
        set_edge_tets(old_tet);
        set_face_tets(old_tet);
    }
}

static void allocate_new_tetrahedra(
    Triangulation *new_triangulation,
    Triangulation *old_triangulation)
{
    int i, j;
    Tetrahedron *old_tet,
                *new_tet;

    /*
     * For each Tetrahedron in the old_triangulation, we want
     * to allocate and initialize the 32 corresponding Tetrahedra
     * in the new_triangulation.
     */

    /*
     * IMPORTANT: It's crucial that one of the outer vertex tetrahedra
     * appears first on new_triangulation's tet list, so that orient()
     * will preserve the manifold's original orientation.
     */

    for (old_tet = old_triangulation->tet_list_begin.next;
         old_tet != &old_triangulation->tet_list_end;
         old_tet = old_tet->next)
    {
        for (i = 0; i < 4; i++)

```

```

    {
        new_tet = NEW_STRUCT(Tetrahedron);
        initialize_tetrahedron(new_tet);
        old_tet->extra->outer_vertex_tet[i] = new_tet;
        INSERT_BEFORE(new_tet, &new_triangulation->tet_list_end);
    }

    for (i = 0; i < 4; i++)
    {
        new_tet = NEW_STRUCT(Tetrahedron);
        initialize_tetrahedron(new_tet);
        old_tet->extra->inner_vertex_tet[i] = new_tet;
        INSERT_BEFORE(new_tet, &new_triangulation->tet_list_end);
    }

    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
        {
            if (i == j)
            {
                old_tet->extra->edge_tet[i][j] = NULL;
                continue;
            }

            new_tet = NEW_STRUCT(Tetrahedron);
            initialize_tetrahedron(new_tet);
            old_tet->extra->edge_tet[i][j] = new_tet;
            INSERT_BEFORE(new_tet, &new_triangulation->tet_list_end);
        }

    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
        {
            if (i == j)
            {
                old_tet->extra->face_tet[i][j] = NULL;
                continue;
            }

            new_tet = NEW_STRUCT(Tetrahedron);
            initialize_tetrahedron(new_tet);
            old_tet->extra->face_tet[i][j] = new_tet;
            INSERT_BEFORE(new_tet, &new_triangulation->tet_list_end);
        }
    }
}

static void set_outer_vertex_tets(
    Tetrahedron *old_tet)
{
    int i,
        j,
        k,
        l;
    Tetrahedron *new_tet;

    /*
     * Set the fields for each of the four outer_vertex_tets.
     */

    for (i = 0; i < 4; i++)
    {
        /*
         * For notational clarity, let new_tet be a pointer
         * to the outer_vertex_tet under consideration.
         */

        new_tet = old_tet->extra->outer_vertex_tet[i];

        /*
         * Set the new_tet's four neighbor fields.
         */
    }
}

```

```

    for (j = 0; j < 4; j++)
        new_tet->neighbor[j] =
            (i == j) ?
                old_tet->extra->inner_vertex_tet[i] :
                old_tet->neighbor[j]->extra->outer_vertex_tet[EVALUATE(old_tet->gluing[j],
i)];

    /*
     * Set the new_tet's four gluing fields.
     */

    for (j = 0; j < 4; j++)
        new_tet->gluing[j] =
            (i == j) ?
                IDENTITY_PERMUTATION :
                old_tet->gluing[j];

    /*
     * Copy the peripheral curves from the old_tet to the ideal
     * vertex of the new_tet. The peripheral curves of
     * finite vertices have already been set to zero.
     */

    for (j = 0; j < 2; j++)
        for (k = 0; k < 2; k++)
            for (l = 0; l < 4; l++)
                new_tet->curve[j][k][i][l] = old_tet->curve[j][k][i][l];
}

static void set_inner_vertex_tets(
    Tetrahedron *old_tet)
{
    int i,
        j;
    Tetrahedron *new_tet;

    /*
     * Set the fields for each of the four inner_vertex_tets.
     */

    for (i = 0; i < 4; i++)
    {
        /*
         * For notational clarity, let new_tet be a pointer
         * to the inner_vertex_tet under consideration.
         */

        new_tet = old_tet->extra->inner_vertex_tet[i];

        /*
         * Set the new_tet's four neighbor fields.
         */

        for (j = 0; j < 4; j++)
            new_tet->neighbor[j] =
                (i == j) ?
                    old_tet->extra->outer_vertex_tet[i] :
                    old_tet->extra->face_tet[j][i];

        /*
         * Set the new_tet's four gluing fields.
         */

        for (j = 0; j < 4; j++)
            new_tet->gluing[j] = IDENTITY_PERMUTATION;
    }
}

```

```

static void set_edge_tets(
    Tetrahedron *old_tet)
{
    int      i,
             j,
             k,
             l;
    Tetrahedron *new_tet;

    /*
     * Set the fields for each of the twelve edge_tets.
     */

    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
        {
            /*
             * Only the case i != j is meaningful.
             */

            if (i == j)
                continue;

            /*
             * For notational clarity, let new_tet be a pointer
             * to the edge_tet under consideration.
             */

            new_tet = old_tet->extra->edge_tet[i][j];

            /*
             * Let i, j, k and l be the VertexIndices of
             * new_tet. i and j are already defined as
             * loop variables. We define k and l here.
             */

            k = remaining_face[i][j];
            l = remaining_face[j][i];

            /*
             * Set the new_tet's four neighbor fields.
             */

            new_tet->neighbor[i] = old_tet->extra->edge_tet[j][i];
            new_tet->neighbor[j] = old_tet->neighbor[i]->extra->edge_tet[EVALUATE(old_tet->gluing[i], i)][EVALUATE(old_tet->gluing[i], j)];
            new_tet->neighbor[k] = old_tet->extra->face_tet[i][k];
            new_tet->neighbor[l] = old_tet->extra->face_tet[i][l];

            /*
             * Set the new_tet's four gluing fields.
             */

            new_tet->gluing[i] = CREATE_PERMUTATION(i, j, j, i, k, k, l, l);
            new_tet->gluing[j] = old_tet->gluing[i];
            new_tet->gluing[k] = CREATE_PERMUTATION(i, i, j, k, k, j, l, l);
            new_tet->gluing[l] = CREATE_PERMUTATION(i, i, j, l, k, k, l, j);
        }
}

static void set_face_tets(
    Tetrahedron *old_tet)
{
    int      i,
             j,
             k,
             l;
    Tetrahedron *new_tet;

    /*
     * Set the fields for each of the twelve face_tets.
     */

```



```

for (i = 0; i < 4; i++)
    for (j = 0; j < 4; j++)
    {
        /*
         * Only the case i != j is meaningful.
         */

        if (i == j)
            continue;

        /*
         * For notational clarity, let new_tet be a pointer
         * to the face_tet under consideration.
         */

        new_tet = old_tet->extra->face_tet[i][j];

        /*
         * Let i, j, k and l be the VertexIndices of
         * new_tet. i and j are already defined as
         * loop variables. We define k and l here.
         */

        k = remaining_face[i][j];
        l = remaining_face[j][i];

        /*
         * Set the new_tet's four neighbor fields.
         */

        new_tet->neighbor[i] = old_tet->extra->inner_vertex_tet[j];
        new_tet->neighbor[j] = old_tet->neighbor[i]->extra->face_tet[EVALUATE(old_tet->gluing[i], i)][EVALUATE(old_tet->gluing[i], j)];
        new_tet->neighbor[k] = old_tet->extra->edge_tet[i][k];
        new_tet->neighbor[l] = old_tet->extra->edge_tet[i][l];

        /*
         * Set the new_tet's four gluing fields.
         */

        new_tet->gluing[i] = IDENTITY_PERMUTATION;
        new_tet->gluing[j] = old_tet->gluing[i];
        new_tet->gluing[k] = CREATE_PERMUTATION(i,i,j,k,k,j,l,l);
        new_tet->gluing[l] = CREATE_PERMUTATION(i,i,j,l,k,k,l,j);
    }
}

static void create_new_cusps(
    Triangulation *new_triangulation,
    Triangulation *old_triangulation)
{
    /*
     * The Cusp data structures for the real cusps are
     * handled separately from the Cusp data structures
     * for the finite vertices.
     */

    create_real_cusps(new_triangulation, old_triangulation);
    create_fake_cusps(new_triangulation);
}

static void create_real_cusps(
    Triangulation *new_triangulation,
    Triangulation *old_triangulation)
{
    Cusp *old_cusp,
        *new_cusp;
    Tetrahedron *old_tet;
    int i;

```

```
/*
 * The Cusp data structures for the ideal vertices
 * in the new_triangulation are essentially just
 * copied from those in the old_triangulation.
 */

/*
 * Allocate memory for the new Cusps, copy in the
 * values from the old cusps, and put them on the
 * queue.
 */

for (old_cusp = old_triangulation->cusp_list_begin.next;
     old_cusp != &old_triangulation->cusp_list_end;
     old_cusp = old_cusp->next)
{
    new_cusp = NEW_STRUCT(Cusp);
    *new_cusp = *old_cusp;
    new_cusp->is_finite = FALSE;
    INSERT_BEFORE(new_cusp, &new_triangulation->cusp_list_end);
    old_cusp->matching_cusp = new_cusp;
}

/*
 * Set the cusp field for ideal vertices in
 * the new_triangulation.
 */

for (old_tet = old_triangulation->tet_list_begin.next;
     old_tet != &old_triangulation->tet_list_end;
     old_tet = old_tet->next)

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

        old_tet->extra->outer_vertex_tet[i]->cusp[i] = old_tet->cusp[i]->matching_cusp;

}
```