

```

/*
 * edge_classes.c
 *
 * This file provides the functions
 *
 * void create_edge_classes(Triangulation *manifold);
 * void replace_edge_classes(Triangulation *manifold);
 * void orient_edge_classes(Triangulation *manifold);
 *
 * which are used within the kernel.
 *
 * create_edge_classes() adds EdgeClasses to a partially
 * constructed manifold which does not yet have them.
 * It assumes the tet->neighbor and tet->gluing fields
 * contain correct values.
 *
 * replace_edge_classes() removes all EdgeClasses from a manifold
 * and adds fresh ones. replace_edge_classes() is typically called
 * by functions which would rather replace invalid EdgeClasses
 * at the end of an algorithm rather than try to maintain them
 * as they go along.
 *
 * orient_edge_classes() orients a neighborhood of each EdgeClass.
 * Relative to this orientation, each of the incident tetrahedra
 * will be seen as right_ or left_handed.
 *
 * The edges of a tetrahedron are indexed according to the following table:
 *
 *      lies      lies
 *      edge  between  between
 *           faces    vertices
 *
 *      0      0,1      2,3
 *      1      0,2      1,3
 *      2      0,3      1,2
 *      3      1,2      0,3
 *      4      1,3      0,2
 *      5      2,3      0,1
 *
 * orient_edge_classes() sets the field tet->edge_orientation[e] to be the
 * orientation of Tetrahedron tet as seen by EdgeClass tet->edge_class[e].
 * In an oriented manifold, all edge_orientations will be right_handed.
 *
 * orient_edge_classes() should be called as soon as a Triangulation
 * is created, and functions which modify a Triangulation should
 * maintain the edge_orientation[] fields.
 *
 * As explained in the documentation at the top of orient.c, orient()
 * and orient_edge_classes() may be called in either order, but both
 * should be called.
 */

#include "kernel.h"

static void initialize_tet_edge_classes(Triangulation *manifold);
static void create_one_edge_class(Triangulation *manifold, Tetrahedron *tet, EdgeIndex e);

void create_edge_classes(
    Triangulation *manifold)
{
    Tetrahedron *tet;
    EdgeIndex e;

    /*
     * First set tet->edge_class[] to NULL for all
     * edges of all Tetrahedra.
     */

    initialize_tet_edge_classes(manifold);

    /*
     * Go down the list of Tetrahedra, and whenever
     * an edge is found with no EdgeClass, create one
     * for it.
     */
}

```

```

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

        for (e = 0; e < 6; e++)

            if (tet->edge_class[e] == NULL)

                create_one_edge_class(manifold, tet, e);
}

static void initialize_tet_edge_classes(
    Triangulation *manifold)
{
    Tetrahedron *tet;
    EdgeIndex e;

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

        for (e = 0; e < 6; e++)

            tet->edge_class[e] = NULL;
}

static void create_one_edge_class(
    Triangulation *manifold,
    Tetrahedron *tet,
    EdgeIndex e)
{
    EdgeClass *new_edge_class;
    FaceIndex front,
              back,
              temp;
    Permutation gluing;
    Tetrahedron *tet0;
    EdgeIndex e0;

    /*
     * Create the new EdgeClass and add it to the list.
     */

    new_edge_class = NEW_STRUCT(EdgeClass);
    initialize_edge_class(new_edge_class);
    INSERT_BEFORE(new_edge_class, &manifold->edge_list_end);

    /*
     * Initialize the fields of the EdgeClass.
     */

    new_edge_class->order = 0;
    new_edge_class->incident_tet = tet;
    new_edge_class->incident_edge_index = e;

    /*
     * Walk around the edge class, setting the tet->edge_class
     * field for each edge we encounter.
     */

    front = one_face_at_edge[e];
    back = other_face_at_edge[e];

    tet0 = tet;
    e0 = e;

    do
    {
        /*
         * Set the edge_class pointer . . .
         */

```

```

    tet->edge_class[e] = new_edge_class;

    /*
     * . . . increment new_edge_class->order . . .
     */
    new_edge_class->order++;

    /*
     * . . . and move on to the next edge.
     */
    gluing = tet->gluing[front];
    tet    = tet->neighbor[front];
    temp   = front;
    front   = EVALUATE(gluing, back);
    back    = EVALUATE(gluing, temp);
    e       = edge_between_faces[front][back];
}
while (tet != tet0 || e != e0);
}

void replace_edge_classes(
    Triangulation *manifold)
{
    EdgeClass *dead_edge_class;

    /*
     * Remove all existing EdgeClasses . . .
     */

    while (manifold->edge_list_begin.next != &manifold->edge_list_end)
    {
        dead_edge_class = manifold->edge_list_begin.next;
        REMOVE_NODE(dead_edge_class);
        my_free(dead_edge_class);
    }

    /*
     * . . . and add fresh ones.
     */

    create_edge_classes(manifold);
}

void orient_edge_classes(
    Triangulation *manifold)
{
    EdgeClass *edge;
    Tetrahedron *tet;
    EdgeIndex e;
    FaceIndex front,
              back,
              temp;
    Orientation relative_orientation;
    Permutation gluing;
    int count;

    /*
     * For each EdgeClass in the Triangulation . . .
     */
    for (edge = manifold->edge_list_begin.next;
         edge != &manifold->edge_list_end;
         edge = edge->next)
    {
        /*
         * Find an incident edge.
         */
        tet = edge->incident_tet;
        e = edge->incident_edge_index;
        front = one_face_at_edge[e];
        back = other_face_at_edge[e];

        /*

```

```

    * View the incident Tetrahedron relative to the
    * right_handed Orientation.
    */
relative_orientation = right_handed;

/*
 * We'll walk around the EdgeClass, setting
 * the Orientation of each incident edge.
 */

for (count = edge->order; --count >= 0; )
{
    /*
     * Set the edge_orientation of the present edge . . .
     */
    tet->edge_orientation[e] = relative_orientation;

    /*
     * . . . and move on to the next edge.
     */
    gluing = tet->gluing[front];
    tet = tet->neighbor[front];
    temp = front;
    front = EVALUATE(gluing, back);
    back = EVALUATE(gluing, temp);
    e = edge_between_faces[front][back];

    /*
     * Change the relative_orientation iff the new
     * new Tetrahedron is oriented differently than
     * the old one.
     */
    if (parity[gluing] == orientation_reversing)
        relative_orientation = ! relative_orientation;
}

/*
 * When we return to the initial Tetrahedron,
 * the relative_orientation should again be right_handed.
 * If it isn't, the triangulation defines an orbifold
 * with a cone-on-a-projective-plane at the center of
 * the current edge class. This error should be rare --
 * in fact is should be possible only for hand-coded
 * Triangulations.
 */
if (relative_orientation != right_handed)
{
    uAcknowledge("The triangulation has a cone-on-a-projective-plane singularity at
the midpoint of an edge class.");
    uFatalError("orient_edge_classes", "edge_classes");
}
}
}

```