

```

/*
 * positioned_tet.c
 *
 * This file provides the following functions for working with PositionedTets:
 *
 * void      veer_left(PositionedTet *ptet);
 * void      veer_right(PositionedTet *ptet);
 * void      veer_backwards(PositionedTet *ptet);
 * Boolean   same_positioned_tet(PositionedTet *ptet0, PositionedTet *ptet1);
 * void      set_left_edge(EdgeClass *edge, PositionedTet *ptet);
 *
 * Their use is described in kernel_prototypes.h.
 */

```

```

#include "kernel.h"

```

```

void veer_left(
    PositionedTet *ptet)
{
    Permutation left_gluing;
    FaceIndex   temp;

    left_gluing = ptet->tet->gluing[ptet->left_face];

    ptet->tet = ptet->tet->neighbor[ptet->left_face];

    temp          = ptet->near_face;
    ptet->near_face = EVALUATE(left_gluing, ptet->left_face);
    ptet->left_face = EVALUATE(left_gluing, temp);
    ptet->right_face = EVALUATE(left_gluing, ptet->right_face);
    ptet->bottom_face = EVALUATE(left_gluing, ptet->bottom_face);

    if (parity[left_gluing] == orientation_reversing)
        ptet->orientation = ! ptet->orientation;
}

void veer_right(
    PositionedTet *ptet)
{
    Permutation right_gluing;
    FaceIndex   temp;

    right_gluing = ptet->tet->gluing[ptet->right_face];

    ptet->tet = ptet->tet->neighbor[ptet->right_face];

    temp          = ptet->near_face;
    ptet->near_face = EVALUATE(right_gluing, ptet->right_face);
    ptet->right_face = EVALUATE(right_gluing, temp);
    ptet->left_face = EVALUATE(right_gluing, ptet->left_face);
    ptet->bottom_face = EVALUATE(right_gluing, ptet->bottom_face);

    if (parity[right_gluing] == orientation_reversing)
        ptet->orientation = ! ptet->orientation;
}

void veer_backwards(
    PositionedTet *ptet)
{
    Permutation near_gluing;
    FaceIndex   temp;

    near_gluing = ptet->tet->gluing[ptet->near_face];

    ptet->tet = ptet->tet->neighbor[ptet->near_face];

    temp          = ptet->left_face;
    ptet->left_face = EVALUATE(near_gluing, ptet->right_face);
    ptet->right_face = EVALUATE(near_gluing, temp);
    ptet->near_face = EVALUATE(near_gluing, ptet->near_face);
    ptet->bottom_face = EVALUATE(near_gluing, ptet->bottom_face);
}

```

```
    if (parity[near_gluing] == orientation_reversing)
        ptet->orientation = ! ptet->orientation;
}

Boolean same_positioned_tet(
    PositionedTet    *ptet0,
    PositionedTet    *ptet1)
{
    return (
        ptet0->tet == ptet1->tet
        && ptet0->near_face == ptet1->near_face
        && ptet0->left_face == ptet1->left_face
        && ptet0->right_face == ptet1->right_face);
    /*
     * If three faces match, so must the fourth, and so must the orientation.
     */
}

void set_left_edge(
    EdgeClass    *edge,
    PositionedTet    *ptet)
{
    ptet->tet = edge->incident_tet;
    ptet->near_face = one_face_at_edge[edge->incident_edge_index];
    ptet->left_face = other_face_at_edge[edge->incident_edge_index];
    ptet->right_face = remaining_face[ptet->left_face][ptet->near_face];
    ptet->bottom_face = remaining_face[ptet->near_face][ptet->left_face];
    ptet->orientation = right_handed;
}
```