Lecture 07 - Half-edges

Philip Caplan

CSCI 0422 - Geometric Modeling (Spring 2022)
Learning objectives

By the end of this lecture you will be able to:

- list the components defined by a half-edge data structure,
- extract the half-edges from a mesh,
- extract the one-ring vertices, edges and faces around a vertex using half-edges.
Getting started...

Switch Host & Client today!

$ git pull
$ make update
$ cmake .
$ make template_class07_halfedges
$ cmake .

Compiling and running the exercise:

$ make class07_halfedges

Compiling and running the solution (after class):

$ make class07_halfedges_sol
Face-based data structures are hard to use to modify meshes.

<table>
<thead>
<tr>
<th>Vertex</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>position</td>
</tr>
<tr>
<td>FaceRef</td>
<td>face</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Face</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VertexRef</td>
<td>vertex[3]</td>
</tr>
<tr>
<td>FaceRef</td>
<td>neighbor[3]</td>
</tr>
</tbody>
</table>
Edge-based data structures are good for modifying surface meshes.

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point position</td>
<td>VertexRef vertex[2]</td>
</tr>
<tr>
<td>EdgeRef edge</td>
<td>FaceRef face[2]</td>
</tr>
<tr>
<td>Face</td>
<td>EdgeRef next[2]</td>
</tr>
<tr>
<td>EdgeRef edge</td>
<td>EdgeRef prev[2]</td>
</tr>
</tbody>
</table>

Diagram of a mesh structure showing connectivity.
Half-based data structure is what we will focus on for surface meshes.

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Halfedge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point position</td>
<td>halfedge</td>
</tr>
<tr>
<td>HalfedgeRef</td>
<td>halfedge</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Halfedge</th>
</tr>
</thead>
<tbody>
<tr>
<td>VertexRef</td>
</tr>
<tr>
<td>FaceRef</td>
</tr>
<tr>
<td>HalfedgeRef</td>
</tr>
<tr>
<td>HalfedgeRef</td>
</tr>
<tr>
<td>HalfedgeRef</td>
</tr>
</tbody>
</table>
Building up the half-edge data structures

1. Loop through each vertex in the mesh:
   a. create a HalfVertex for this vertex.
   b. assign the index of this HalfVertex to the index of the vertex in the mesh.
   c. save the coordinates of the HalfVertex

As an exercise, we will apply this algorithm to the mesh of a cube (which is closed).
Building up the half-edge data structures

1. Loop through each vertex in the mesh:
   a. create a HalfVertex for this vertex.
   b. assign the index of this HalfVertex to the index of the vertex in the mesh.
   c. save the coordinates of the HalfVertex

2. Loop through the elements in the mesh:
   a. create a HalfFace for this element.

As an exercise, we will apply this algorithm to the mesh of a cube (which is closed).
Building up the half-edge data structures

1. Loop through each vertex in the mesh:
   a. create a HalfVertex for this vertex.
   b. assign the index of this HalfVertex to the index of the vertex in the mesh.
   c. save the coordinates of the HalfVertex

2. Loop through the elements in the mesh:
   a. create a HalfFace for this element.
   b. loop through the edges of this face in CCW order:
      • create a HalfEdge for this edge.

As an exercise, we will apply this algorithm to the mesh of a cube (which is closed).
Building up the half-edge data structures

1. Loop through each vertex in the mesh:
   a. create a HalfVertex for this vertex.
   b. assign the index of this HalfVertex to the index of the vertex in the mesh.
   c. save the coordinates of the HalfVertex

2. Loop through the elements in the mesh:
   a. create a HalfFace for this element.
   b. loop through the edges of this face in CCW order:
      ● create a HalfEdge for this edge.
      ● retrieve the global indices of the endpoints of this directed edge: \((e_0, e_1)\).
      ● retrieve the vertex at endpoint (vertex) \(e_0\).
      ● tell the created edge that its origin vertex is the vertex at \(e_0\).
      ● tell the vertex at \(e_0\) that it’s outgoing edge is the edge we created.
      ● tell the edge that it’s face (to the left) is the face created in step 2a.
      ● look for the opposite (twin) edge by checking if the edge \((e_1, e_0)\) exists yet.
         → if it does, then point the created edge and the twin to each other (as twins)
         → otherwise, initialize the twin to nullptr
      ● also set the next and prev pointers (as you circle around the element).

As an exercise, we will apply this algorithm to the mesh of a cube (which is closed).
Take a moment and examine the mesh we will use today.

I added a new feature to visualize the vertex numbers! (click the "numbers" checkbox)
What about meshes that have boundaries?

```cpp
for (auto & it = edges_.begin(); it != edges_.end(); ++it) {
    HalfEdge* he = it->get();
    if (he->face != nullptr) continue; // not a boundary edge
    flux_assert( he->next == nullptr );
    // keep looking for another boundary edge
    HalfEdge* he_next = he->twin;
    while (he_next->face != nullptr) {
        he_next = he_next->prev->twin;
    }
    he->next = he_next;
    he_next->prev = he;
}
```

**Convention:**
- if edge->face is nullptr, that edge is a boundary edge.
- to loop through boundary edges: find one boundary edge, then go to edge->next until getting back to the start.
What about meshes that have boundaries? (continued)

To retrieve a boundary edge, we can use:

```cpp
template<typename type>
HalfEdge*
HalfEdgeMesh<<type>::get_boundary_edge() const {
    for (auto& e_ptr = edges_.begin(); it != edges_.end(); ++it) {
        HalfEdge* e = e_ptr.get();
        if (e->face == nullptr) return e; // if the face is null, this is a boundary edge
    }
    return nullptr; // no boundary edge
}
```

To traverse the list of boundary edges, we can then use:

```cpp
HalfEdge* first = e; // some boundary edge we found from the previous algorithm
int count = 0;
do {
    count++;
    e = e->next; // go clockwise
} while (e != first);
std::cout << "there are " << count << " boundary edges" << std::endl;
```
Half-edges API in *flux* (what you need for now)

```cpp
#include "halfedges.h"

... 
HalfEdgeMesh<Triangle> half(mesh); // builds half edge data structures from regular mesh

// looping through vertices
for (auto& v_ptr : half.vertices()) {
    HalfVertex* v = v_ptr.get();
    // something with the half vertex v
}

// looping through edges
for (auto& e_ptr : half.edges()) {
    HalfEdge* e = e_ptr.get();
    // something with the half edge e
}

// looping through face
for (auto& f_ptr : half.faces()) {
    HalfEdge* f = f_ptr.get();
    // something with the half face f
}

// retrieving a boundary edge
HalfEdge* bnd_edge = half.get_boundary_edge();

// get the total number of boundary edges
int nb_bnd = half.nb_boundary(); // should be 0 for closed meshes

// retrieve a mesh from a half-edge representation
Mesh<Triangle> mesh2(dim);
half.extract(mesh2);
```
The one-ring is the set of entities surrounding a vertex.

```cpp
void get_onering( const HalfVertex* v, std::vector<HalfVertex*>& ring ) {
    HalfEdge* edge = v->edge;
    HalfEdge* first = edge;
    do {
        ring.push_back( edge->twin->vertex );
        edge = edge->twin->next;
    } while (first != edge);
}
```
The one-ring is the set of entities surrounding a vertex.

```cpp
void get_onering( const HalfVertex* v, std::vector<HalfVertex*>& ring ) {

    HalfEdge* edge = v->edge;
    HalfEdge* first = edge;
    do {

        ring.push_back( edge->twin->vertex );
        edge = edge->twin->next;

    } while (first != edge);
}
```

exercise: practice extracting the one-ring of edges and faces!
Your TODO list . . .

- examine src/flux-base/src/halfedges.[h,cpp],
- add unit tests for src/flux-base/src/halfedges.[h,cpp],
- work on Project 1!
Don’t forget to **commit** and **push** your changes!

**Host:** (assuming you are in top-level *flux directory*)

```sh
$ git add exercises/class07
$ git commit -a -m "added exercises from lecture 7"
$ git push
```

If you are in a build directory, the first command would be: $ git add ../../exercises/class07

**Client:**

```sh
$ git pull
```