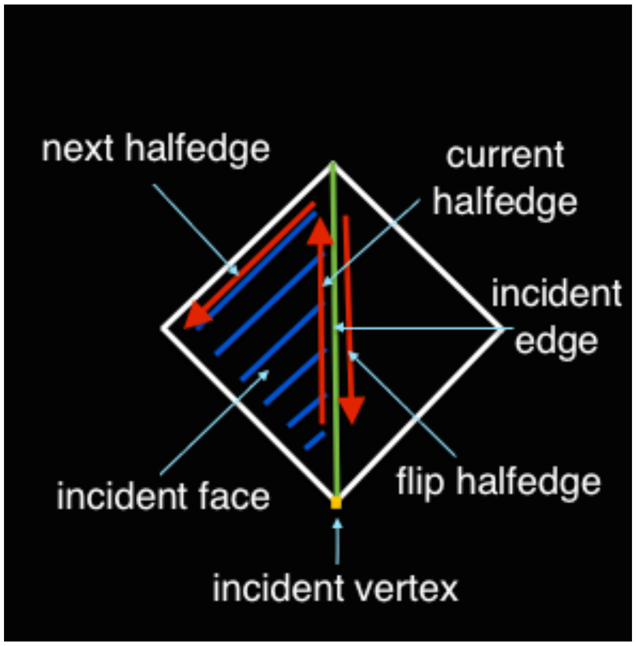
The Half-Edge Data Structure

In geometry processing algorithms, very often both global and local traversal of a mesh is required. This means that connectivity information relating vertices, edges and faces must be provided. The minimal set of operations exploiting this connectivity information usually include:

1. Enumeration of individual vertices, edges and faces in an unspecified order.
2. Oriented traversal of the edges of a face and access to its vertices.
3. Access to the incident faces of an edge (left or right based on convention) and to its two endpoint vertices.
4. Access to an incident face or edge of a vertex. This enables enumeration of the one-ring neighborhood of the vertex.

Since connectivity information primarily relates to mesh edges, most data structures for polygonal meshes tend to be edge based.



The half edge data structure splits each edge into two oriented half edges. A consistent counter clockwise orientation is assumed around each face and along each boundary from the input vertex and face indices. Furthermore edges are assumed to be non-manifold (no more than 2 faces per edge). Each half edge stores a reference to:

1. the vertex it points to
2. the edge it lies on
3. its adjacent face
4. the next half edge
5. the opposite half edge

Enumeration of the edges of a face can be achieved as follows:

```
HalfEdgeRef he = face->he
do {
        EdgeRef e = he->e;
        // do something with e

        he = he->next;
} while (he != face->he);
```

Enumeration of the one ring neighborhood of a vertex can be achieved as follows:

```
HalfEdgeRef he = vertex->he
do {
        // do something with the element(s) adjacent to this HalfEdgeRef

        he = he->flip->next; // here flip is the opposite HalfEdgeRef
} while (he != face->he);
```

A *Ref object can be internally implemented as a pointer.

Implementation: https://github.com/rohan-sawhney/halfedge-mesh