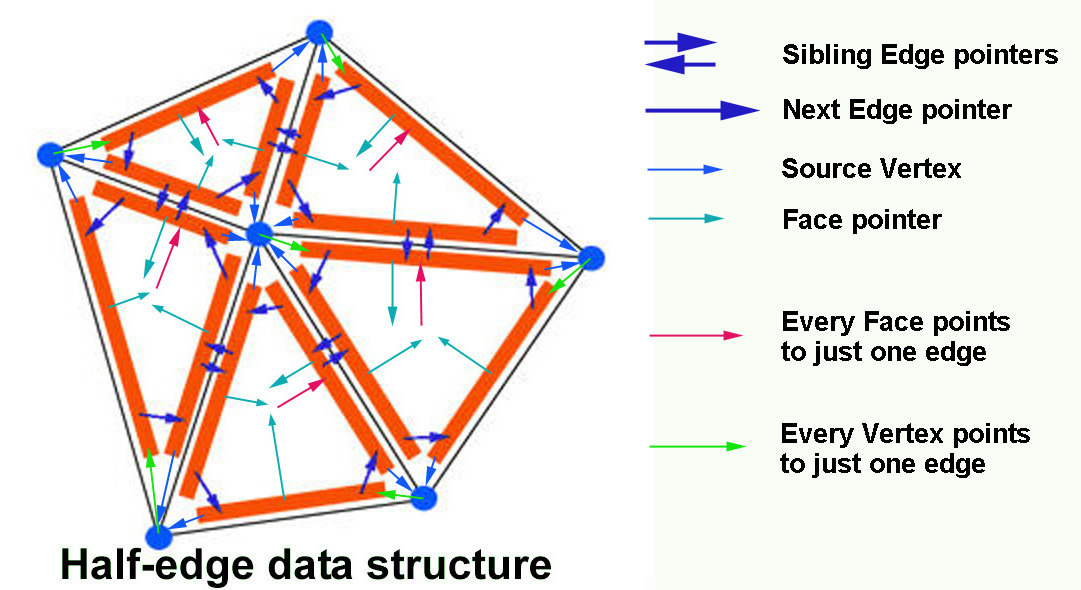
[**半边数据结构**](http://blog.csdn.net/sheila_1988/article/details/6739960)

表示一个空间多边形网格的常用方法是使用一个顶点列表和一个面列表，在面结构中存储属于该面的顶点的指针。这种方式是非常便捷高效的，但是它没有存储顶点、边和面之间的从属、包含及邻接关系。如当我们想要删除某个点，或者比如想要找到某个点的1-邻域点，这时我们往往要对点列表和面列表整体进行遍历搜索，这样开销是很大的，会直接影响到整个系统的效率。

为了提高点线面的查询或增删改操作的效率，半边数据结构中增加了“半边”的概念，为了方便表达拓扑关系，它将一条边表示成拓扑意义上方向相反的两条“半边”，所以称为半边数据结构。

下图为用半边数据结构表示的三角形网格的一部分。



半边结构由三个对象组成：顶点、半边、面。从上图中，我们可以看到半边（用橙黄色代表）会围绕着一个面的边缘形成一个循环列表，其环绕方向我们可以预先实现给定，这个面为半边所对应的面。每个半边存储一个指向它离开的顶点的指针、一个指向它相对的半边的指针、一个指向它对应的面的指针以及一个指向它的下一个半边的指针，类似如下：

struct HE\_edge

{  
 HE\_vert\* vert; // vertex at the end of the half-edge

HE\_edge\* pair; // oppositely oriented adjacent half-edge

HE\_face\* face; // face the half-edge borders

HE\_edge\* next; // next half-edge around the face

};

每个顶点存储其点的位置，以及一个从此点出发的半边的指针。从每个顶点出发的半边可能会有多个，我们可以从中随便挑选出一个，在后面可以看出这里只须存储一个半边即可。顶点结构类似如下：

struct HE\_vert

{  
 float x;

float y;

float z;  
 HE\_edge\* edge; // one of the half-edges emantating from the vertex

};

每个面存储一个属于该面的半边的指针，同样此处只须随意选择一个即可。面结构类似如下：

struct HE\_face

{  
 HE\_edge\* edge; // one of the half-edges bordering the face

};

半边数据结构中，点线面的邻接关系就直接存储在这些点线面中，我们可以很方便快捷的进行查询。

我们可以找到半边两端的顶点，以及找到交于半边所在边的两个面。

HE\_vert\* vert1 = edge->vert;

HE\_vert\* vert2 = edge->pair->vert;  
 HE\_face\* face1 = edge->face;

HE\_face\* face2 = edge->pair->face;

我们可以找到属于一个面的所有半边。

HE\_edge\* edge = face->edge;  
 do {  
 // do something with edge

edge = edge->next;

} while (edge != face->edge);

我们可以找到从一个点出发的所有半边。

HE\_edge\* edge = vert->edge;  
 do {  
 // do something with edge, edge->pair or edge->face

edge = edge->pair->next;  
 } while (edge != vert->edge);

参考文献：

http://www.flipcode.com/archives/The\_Half-Edge\_Data\_Structure.shtml