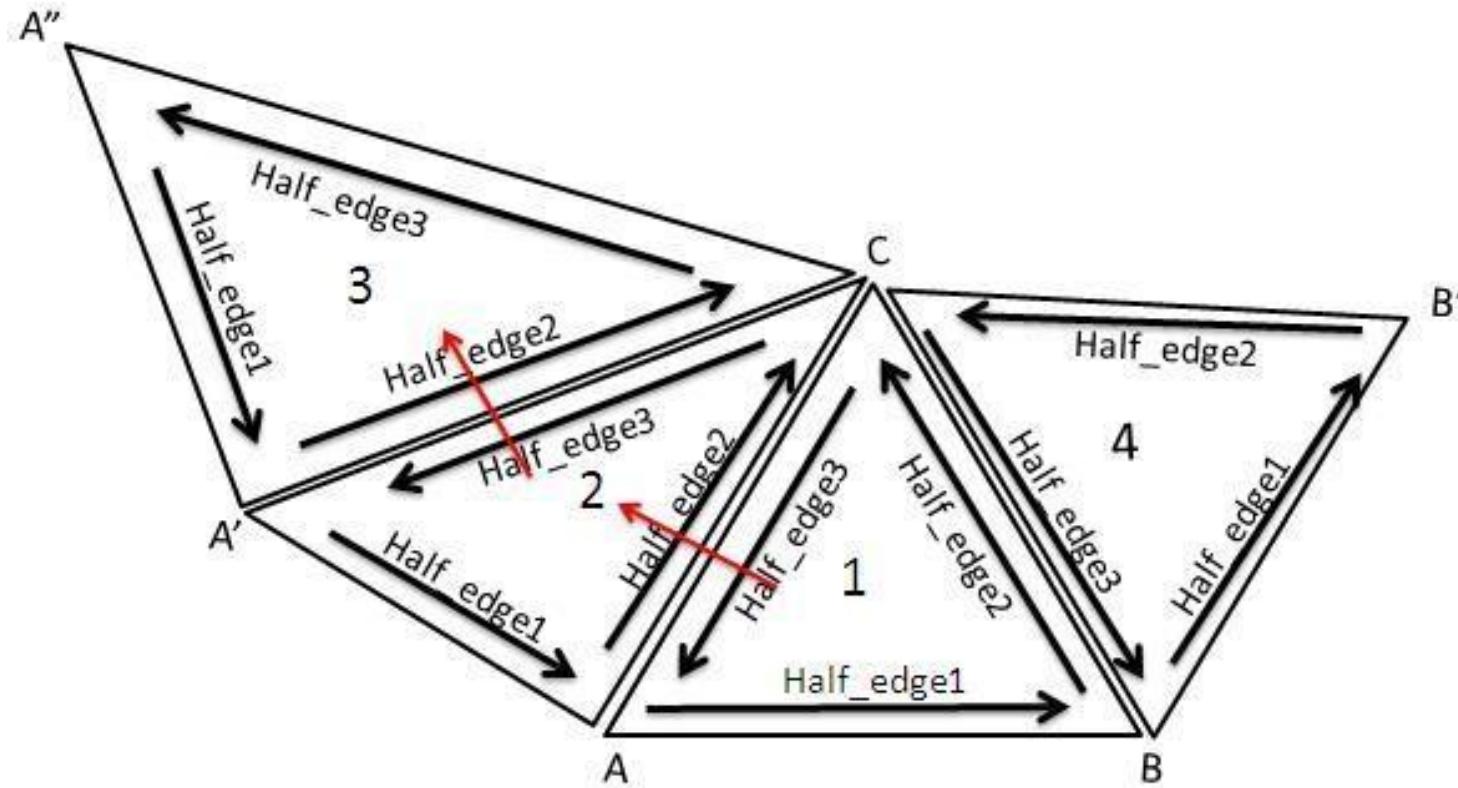


OpenMesh使用简介

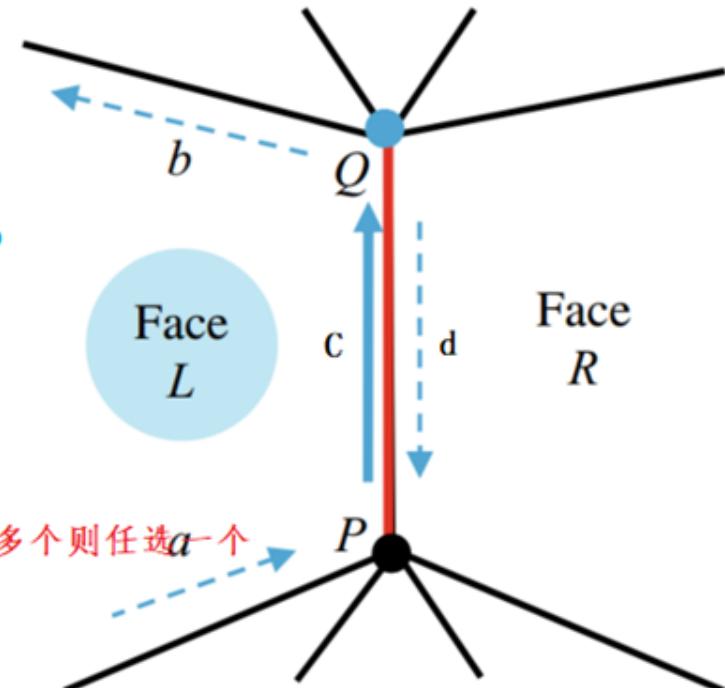
吴文明
计算机与信息学院



半边结构



```
struct H_edge
{
    Vertex *vert; //指向自身的出发点 P
    Face   *face; //指向自身相邻的面片 L
    H_edge *prev, *next; //指向前/后一个半边 a与b
    H_edge *pair; //指向Twins半边 d
}; // c
struct Vertex
{
    float x, y, z;
    H_edge  *edge; //指向一个以它为出发点的半边，若多个则任选一个
}; // p
struct Face
{
    H_edge *edge; //指向任意一个环绕它的半边 c
}; // L
```



Half-edge data structure

- 使用准备：
 - 加入Include并定义MyMesh:

```
#include <OpenMesh/Core/IO/MeshIO.hh>
#include <OpenMesh/Core/Mesh/TriMesh_ArrayKernelT.hh>
#include "OpenMesh/Core/Mesh/Handles.hh"

typedef OpenMesh::TriMesh_ArrayKernelT<> MyMesh;
```



- 读

```
//从文件中读取网格模型
MyMesh mesh;
bool result = OpenMesh::IO::read_mesh(mesh,
    "e:/MyProjects/MeshProcessor/ModelData/trunk/干净模型/Eight.ply");
if (result == false)
{
    return 1;
}
```

- 写

```
//将网格模型保存到文件中
bool result = OpenMesh::IO::write_mesh(mesh, "C:/Hello.ply");
```



- 得到顶点的总数量

```
//得到网格模型mesh中的顶点的总数量
cout << mesh.n_vertices();
```

- 遍历所有顶点

```
//遍历mesh中所有的顶点。
for (auto it = mesh.vertices_begin(); it != mesh.vertices_end(); ++it)
{
    //输出这些顶点的坐标
    auto point = mesh.point(it.handle());
    cout << "x: " << point.data()[0] << "  y:" << point.data()[1] << "  z:"
        << point.data()[2] << endl;
}
```



- 遍历所有的面

```
//遍历mesh中所有的面。  
for (auto it = mesh.faces_begin(); it != mesh.faces_end(); ++it)  
{  
    //对it进行一些操作  
}
```



- 遍历所有的半边

```
//遍历mesh中所有的half edge。  
for (auto it = mesh.halfedges_begin(); it != mesh.halfedges_end(); ++it)  
{  
    //得到每条half edge的起始点和终点。  
    auto fromVertex = mesh.from_vertex_handle(it.handle());  
    auto toVertex = mesh.to_vertex_handle(it.handle());  
}
```



- 得到半边周围的几何元素

```
MyMesh::HalfedgeHandle he;
//得到半边he所对应的面。
MyMesh::FaceHandle face = mesh.face_handle(he);

//得到半边he的下一条半边。
MyMesh::Halfedge he2 = mesh.next_halfedge_handle(he);

//得到半边he相对的半边。
MyMesh::Halfedge he3 = mesh.opposite_halfedge_handle(he);

//得到半边he的起点和终点。
MyMesh::VertexHandle fromV = mesh.from_vertex_handle(he);
MyMesh::VertexHandle toV = mesh.to_vertex_handle(he);
```

- 遍历顶点周围的1-邻域面

```
//遍历顶点vertex周围的1-邻域面
OpenMesh::VertexHandle vertex;
for (auto it = mesh.vf_begin(vertex); it != mesh.vf_end(vertex); ++it)
{
}
```



- 遍历顶点周围的1-邻域顶点

```
//遍历顶点vertex周围的1-邻域顶点
OpenMesh::VertexHandle vertex;
for (auto it = mesh.vv_begin(vertex); it != mesh.vv_end(vertex); ++it)
{
    auto vertex = it.handle();
}
```



- 面周围的1-邻域顶点，即属于该面的所有顶点（3个）

```
//遍历属于face的所有顶点
OpenMesh::FaceHandle face;
for (auto it = mesh.fv_begin(face); it != mesh.fv_end(face); ++it)
{
    auto vertex = it.handle();
}
```



- 遍历面的所有半边（3条）

```
//遍历属于face的所有半边
OpenMesh::FaceHandle face;
for (auto it = mesh.fh_begin(face); it != mesh.fh_end(face); ++it)
{
    auto he = it.handle();
}
```



- 遍历面周围的1-邻域面

```
//遍历属于face的所有顶点
OpenMesh::FaceHandle face;
for (auto it = mesh.fv_begin(face); it != mesh.fv_end(face); ++it)
{
    auto vertex = it.handle();
}
```

- 计算、遍历所有面上的法向

```
//计算并遍历所有面上的法向
mesh.request_face_normals();
mesh.update_face_normals();
for (auto it = mesh.faces_begin(); it != mesh.faces_end(); ++it)
{
    auto face = it.handle();
    MyMesh::Normal normal = mesh.normal(face);
    double x = normal.data()[0];
    double y = normal.data()[1];
    double z = normal.data()[2];
}
```



- 计算、遍历所有点上的法向（用1-邻域面上的法向平均）

```
//计算并遍历所有点上的法向
mesh.request_face_normals();
mesh.request_vertex_normals();
mesh.update_normals();
for (auto it = mesh.vertices_begin(); it != mesh.vertices_end(); ++it)
{
    auto vertex = it.handle();
    MyMesh::Normal normal = mesh.normal(vertex);
    double x = normal.data()[0];
    double y = normal.data()[1];
    double z = normal.data()[2];
}
```



- 向量的基本几何操作

```
MyMesh::Normal v1(2, 2, 2), v2;  
//v1单位化  
v1 = v1 / v1.length();  
//或者  
v1.normalize();  
//两个向量的内积  
double innerProduct = v1 * v2;  
//两个向量的叉积  
MyMesh::Normal v3 = v1 % v2;
```