Triangle Mesh: Halfedge Data Structure

David Gu

Computer Science Department
Stony Brook University

gu@cs.stonybrook.edu

August 11, 2023

Triangle Mesh: Halfedge Data Structure

Discrete Surfaces

Acquired using 3D scanner.

Discrete Surfaces

Our group has developed high speed 3D scanner, which can capture facial surfaces with dynamic expressions.

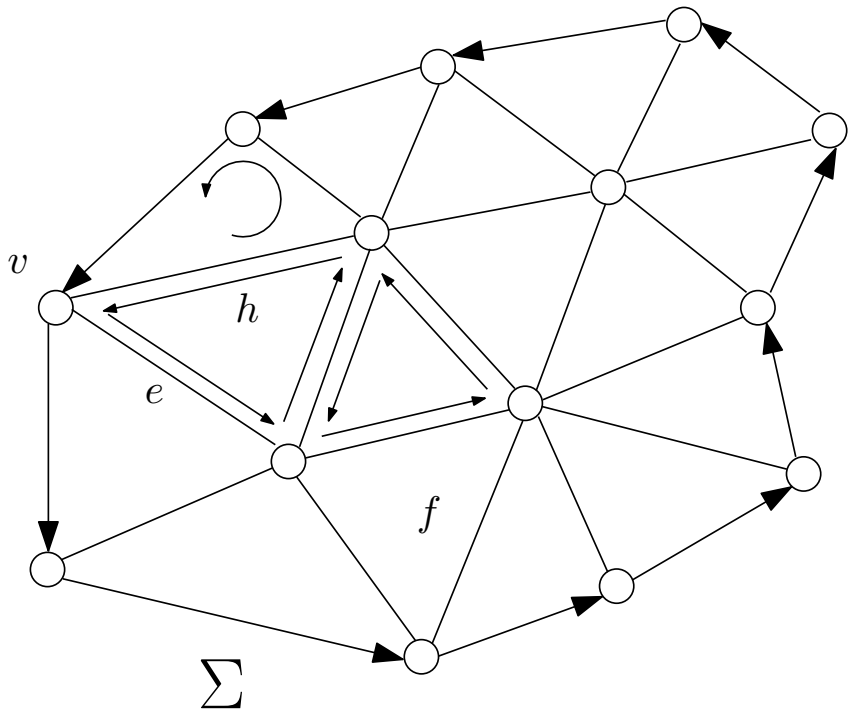
Generic Surface Model - Triangular Mesh

- Surfaces are represented as polyhedron triangular meshes.
- Isometric gluing of triangles in $\mathbb{E}^2$.
- Isometric gluing of triangles in $\mathbb{H}^2, \mathbb{S}^2$.

Discrete structures

- Topology - Simplicial Complex , combinatorics
- Conformal Structure - Corner angles (and other variant definitions)
- Riemannian metrics - Edge lengths
- Embedding - Vertex coordinates

Generic Surface Model - Triangular Mesh



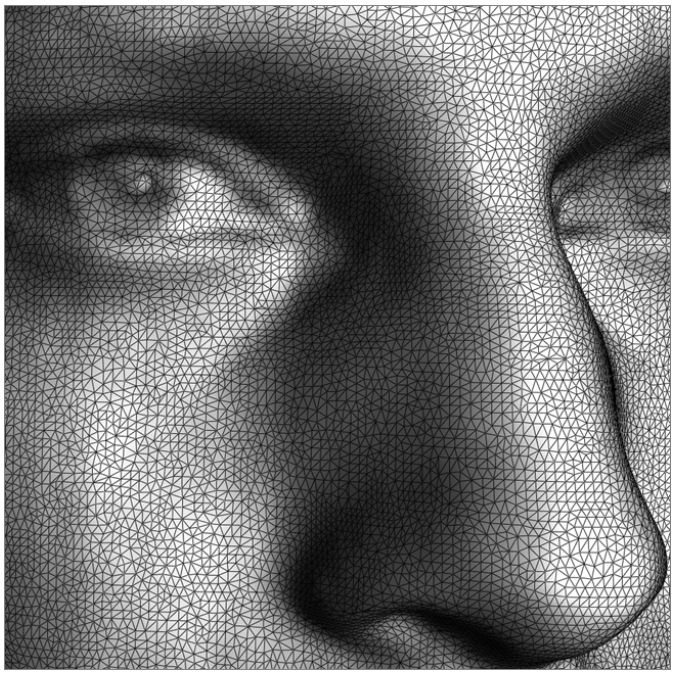
Triangle mesh

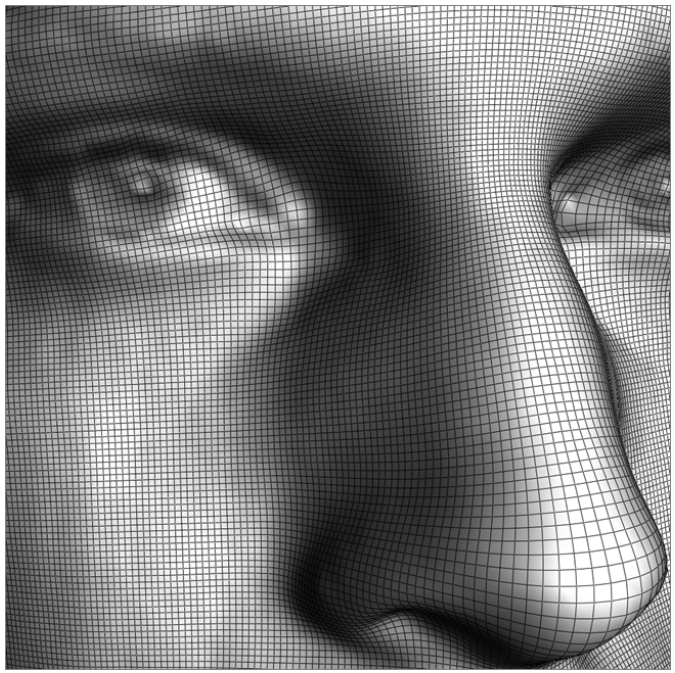
Definition (Triangle Mesh)

A triangle mesh is a oriented two dimensional simplical complex, generally embedded in $\mathbb{R}^3$.

Our goal is to design a data structure to efficiently represent general meshes.

Generic Surface Model - Triangular Mesh





halfedge data structure

fundamental classes

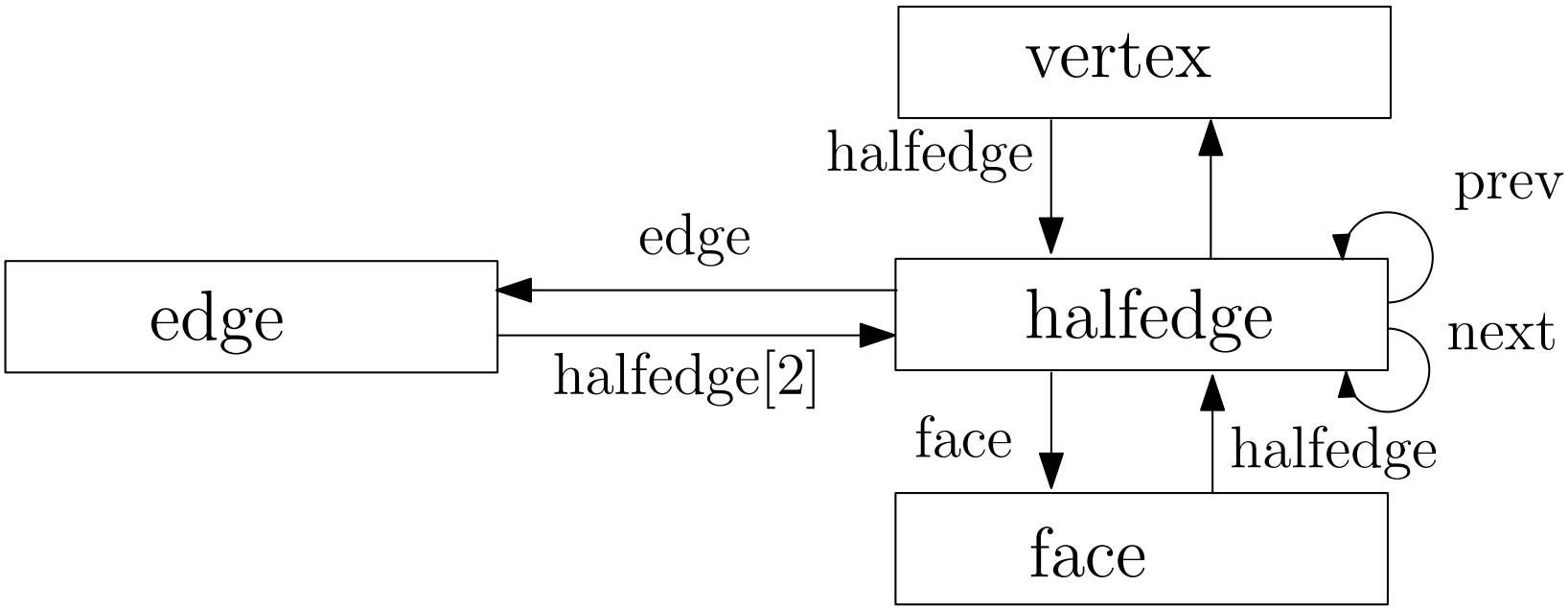
- Vertex
- Halfedge, oriented edge
- Edge, non-oriented edge
- Face, oriented

Links

All objects are linked together through pointers, such that

1. The local Eucler operation can be easily performed
2. The memory cost is minimized

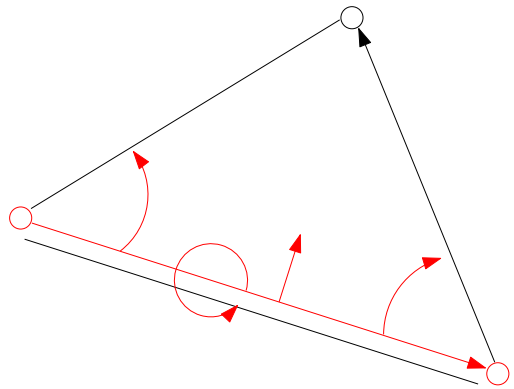
Generic Surface Model - Triangular Mesh



Halfedge class

Pointers

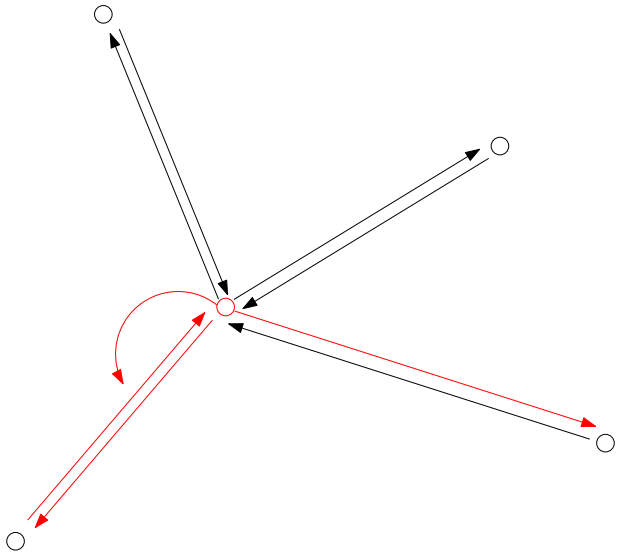
- Halfedge pointers: prev, next halfedge;
- Vertex pointers: target vertex, source vertex;
- Edge pointer: the adjacent edge;
- face pointer: the face it belongs to;



Vertex class

Pointers

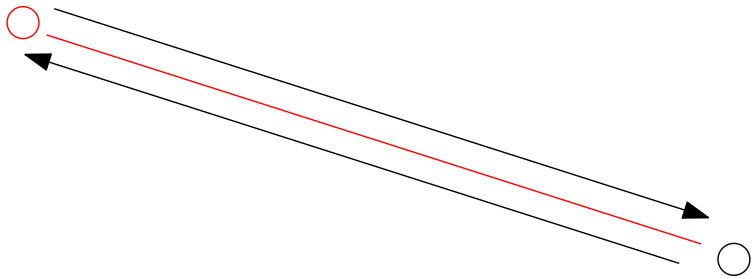
- Halfedge pointers: the first in halfedge



Edge class

Pointers

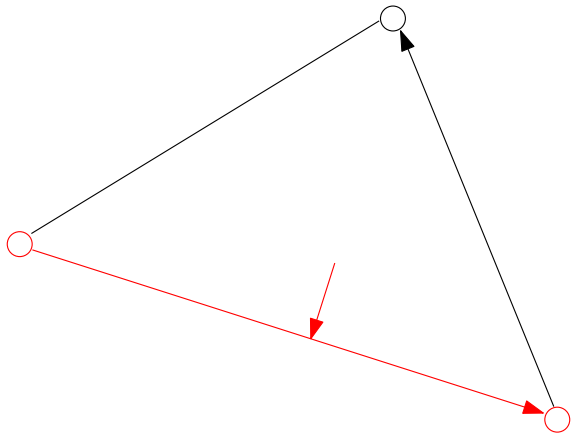
- Halfedge pointers: to the adjacent two halfedges.
- if the edge is on the boundary, then the second halfedge pointer is null.



Face class

Pointers

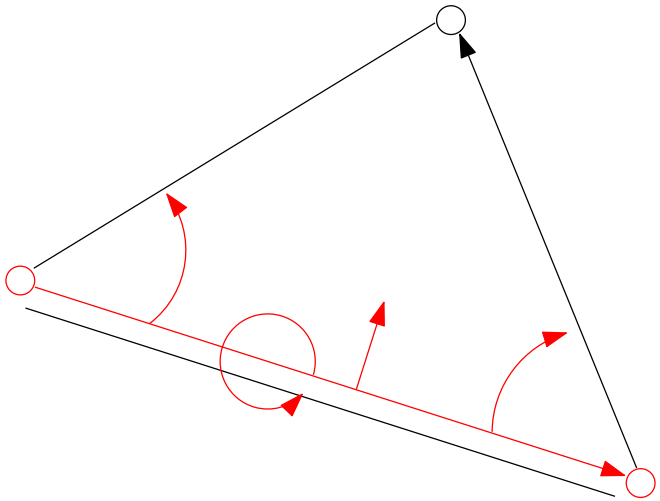
- Halfedge pointers: to the first halfedge.



Mesh class

Data members

- A list of vertices;
- A list of halfedges;
- A list of edges;
- A list of faces;



Euler Operation

circulating neighbors of a vertex $v \rightarrow v/e/f/h$

- iterate out-halfedges counter-clock-wisely
- iterate in-halfedges counter-clock-wisely
- iterate neighboring faces CCWly
- iterate neighboring vertices CCWly

Rotate a halfedge about its target vertex clwly:

$$he = he \rightarrow next() \rightarrow dual();$$

Rotate a halfedge about its target vertex ccwly:

$$he = he \rightarrow dual() \rightarrow prev();$$

Euler Operation

circulating neighbors of a face $f \rightarrow v/e/f/h$

- iterate halfedges ccwly
- iterate edges ccwly
- iterate vertices ccwly
- iterate faces ccwly

Circulate halfedges of a face ccwly:

$$he = he \rightarrow next()$$

circulate halfedge of a face clwly:

$$he = he \rightarrow prev();$$

Attributes

Attributes

Each object stores attributes (traits) which defines other structures on the mesh:

- metric structure: edge length
- angle structure: halfedge
- curvature : vertex
- conformal factor: vertex
- Laplace-Beltrami operator: edge
- Ricci flow edge weight; edge
- holomorphic 1-form: halfedge

Coding Procedure

Define Mesh Class

- define vertex, face, edge, halfedge classes;
- define mesh class with template, including all types of iterators;
- instantiate the mesh class, with the vertex, face, edge, halfedge classes;
- define different methods for the mesh class.

Design Algorithm

- Use the mesh as the main data structure;
- Update the attributes of vertex, edge, halfedge and face;
- Update the connectivity;
- Form numerical linear systems, use linear package to solve it.

```
class CMyVertex : public CVertex
{
public:
  CMyVertex() : m_rgb(1, 1, 1) {};
  ~CMyVertex() {};
  void _from_string();
  CPoint & rgb() { return m_rgb; };
protected:
    CPoint m_rgb;
};
inline void CMyVertex::_from_string()
{
  CParser parser(m_string);
  for (std::list<CToken*>::iterator iter = parser.tokens()
  .begin(); iter != parser.tokens().end(); ++iter)
  {
    CToken * token = *iter;
    if (token->m_key == "rgb") // CPoint
      token->m_value >> m_rgb;
  }
}
```

Listing 1: Vertex Class

```
class CMyEdge : public CEdge
{
public:
  CMyEdge() :m_sharp(false) {};
  ~CMyEdge() {};
  void _from_string();
  bool & sharp() { return m_sharp; };
protected:
  bool m_sharp;
};
inline void CMyEdge::_from_string()
{
  CParser parser(m_string);
  for (std::list<CToken*>::iterator iter = parser.tokens()
  .begin(); iter != parser.tokens().end(); ++iter)
  {
    CToken * token = *iter;
    if (token->m_key == "sharp") // bool
      m_sharp = true;
  }
}
```

Listing 2: Edge Class

```
class CMyFace : public CFace
{
public:

  CPoint & normal() { return m_normal; };
  double & area()   { return m_area; }
protected:
  CPoint m_normal;
  double m_area;
};

  class CMyHalfEdge: public CHalfEdge
  {
  public:
      double angle() { return m_angle; }
  protected:
      double m_angle;
  };
```

Listing 3: Face and HalfEdge Class

```
template<typename V, typename E, typename F, typename H>
class MyMesh : public CDynamicMesh<V, E, F, H>
{
public:
        typedef V V;
        typedef E E;
        typedef F F;
        typedef H H;

    typedef CBoundary<V, E, F, H>          CBoundary;
    typedef CLoop<V, E, F, H>              CLoop;

    typedef MeshVertexIterator<V, E, F, H>
  MeshVertexIterator;
    typedef MeshEdgeIterator<V, E, F, H>
  MeshEdgeIterator;
    typedef MeshFaceIterator<V, E, F, H>
  MeshFaceIterator;
    typedef MeshHalfEdgeIterator<V, E, F, H>
  MeshHalfEdgeIterator;

    typedef VertexVertexIterator<V, E, F, H>
  VertexVertexIterator;
```

```
19      typedef VertexEdgeIterator<V, E, F, H>
        VertexEdgeIterator;
20      typedef VertexFaceIterator<V, E, F, H>
        VertexFaceIterator;
21      typedef VertexInHalfedgeIterator<V, E, F, H>
        VertexInHalfedgeIterator;
22      typedef VertexOutHalfedgeIterator<V, E, F, H>
        VertexOutHalfedgeIterator;
23
24      typedef FaceVertexIterator<V, E, F, H>
        FaceVertexIterator;
25      typedef FaceEdgeIterator<V, E, F, H>
        FaceEdgeIterator;
26      typedef FaceHalfedgeIterator<V, E, F, H>
        FaceHalfedgeIterator;
27
28      void outputMeshInfo();
29      void testIterator();
30  };
31  typedef MyMesh<CMyVertex, CMyEdge, CMyFace, CMyHalfEdge>
      CMyMesh;
```

Listing 4: Mesh Class

```
1  template<typename V, typename E, typename F, typename H>
2  void MyMesh<V, E, F, H>::testIterator()
3  {
4    for (MeshVertexIterator viter(this); !viter.end(); ++
     viter)
5    {
6      V * pV = *viter;
7      // you can do something to the vertex here
8      // ...
9
10     for (VertexVertexIterator vviter(pV); !vviter.end();
     ++vviter)
11     {
12       V * pW = *vviter;
13       // you can do something to the neighboring vertices
     with CCW
14       // ...
15     }
16
17     for (VertexEdgeIterator veiter(pV); !veiter.end(); ++
     veiter)
18     {
19       E * pE = *veiter;
```

```
20        // you can do something to the neighboring edges
    with CCW
21        // ...
22     }
23
24     for (VertexFaceIterator vfiter(pV); !vfiter.end(); ++
    vfiter)
25     {
26        F * pF = *vfiter;
27        // you can do something to the neighboring faces
    with CCW
28        // ...
29     }
30
31     for (VertexInHalfedgeIterator vhiter(this, pV); !
    vhiter.end(); ++vhiter)
32     {
33        H * pH = *vhiter;
34        // you can do something to the incoming halfedges
    with CCW
35        // ...
36     }
37  }
```

```
38
39      for (MeshEdgeIterator eiter(this); !eiter.end(); ++eiter
        )
40      {
41        E * pE = *eiter;
42        // you can do something to the edge here
43        // ...
44      }
45
46      for (MeshFaceIterator fiter(this); !fiter.end(); ++fiter
        )
47      {
48        F * pF = *fiter;
49        // you can do something to the face here
50        // ...
51      }
52
53      //there are some other iterators which you can find them
         in class MyMesh
54
55    }
```

Listing 5: Test different iterators