Mesh representations and data structures

Luca Castelli Aleardi

ÉCOLE POLYTECHNIQUE

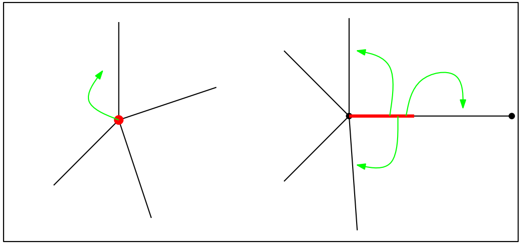
Shared vertex representation

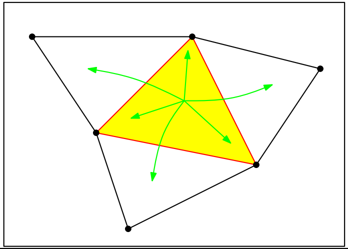
Half-edge DS

Winged edge

Triangle based DS

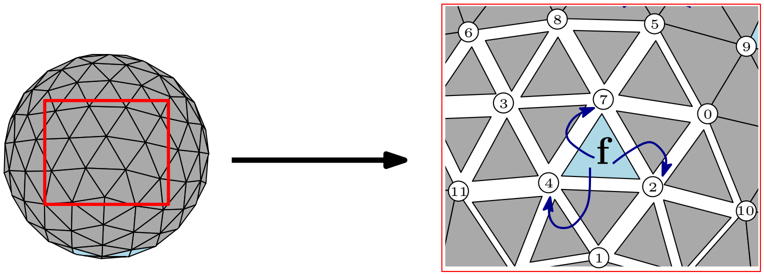
Corner Table





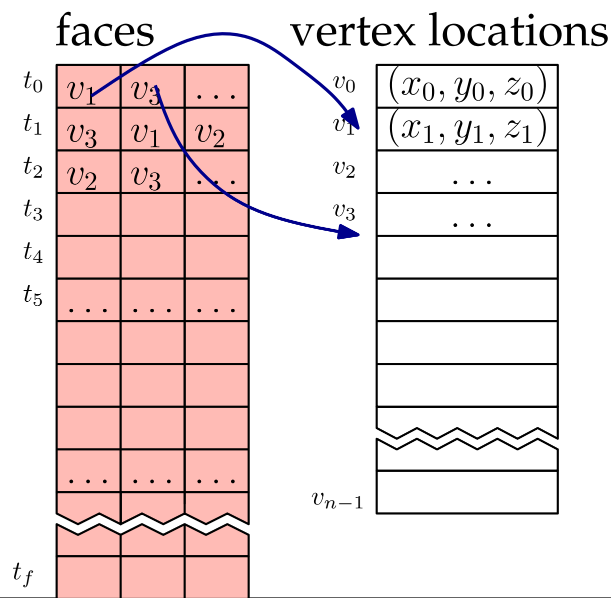
Shared vertex representation

easy to implement
quite compact
not efficient for traversal



for each face (of degree d), store:
- d references to incident vertices

for each vertex, store:
- 1 reference to its coordinates




```
class Point{
  double x;
  double y;
}
```

geometric information

```
class Vertex{
 Point p;
}

class Face{
Vertex[]  vertices;
}
```

combinatorial information

Memory cost

$$3 \times f = 6n$$

Size (number of references)

Queries/Operations

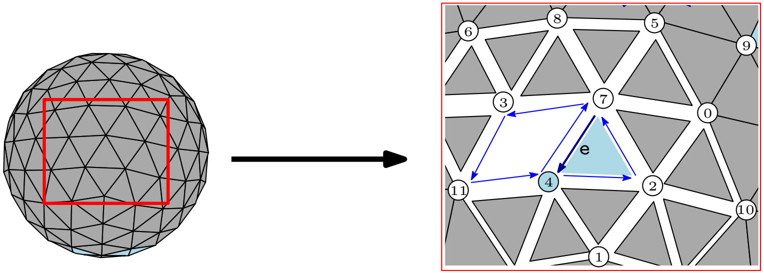
List all vertices or faces

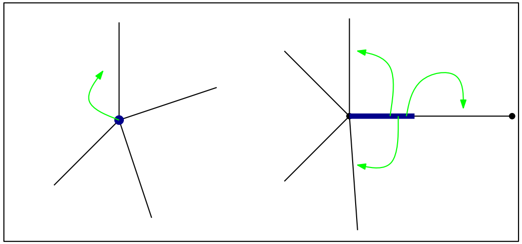
Test adjacency between u and v

Find the 3 neighboring faces of f

List the neighbors of vertex v

Half-edge data structure: polygonal (orientable) meshes



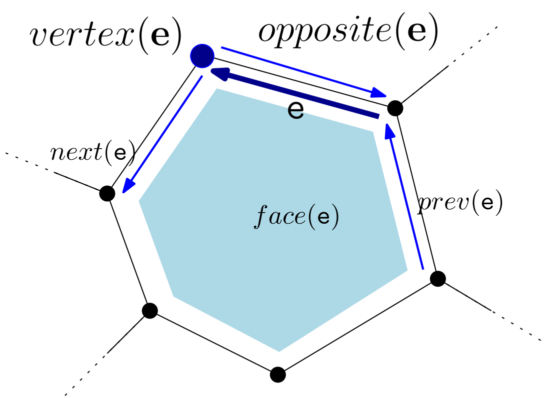


$$f + 5 \times h + n \approx 2n + 5 \times (2e) + n = 32n + n$$

Size (number of references)

```
class Point{
  double x;
  double y;
}
```

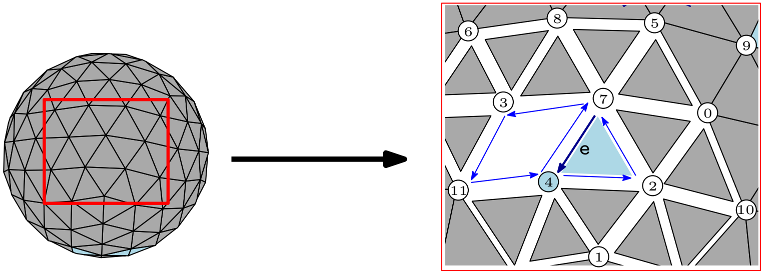
geometric information




```
class Halfedge{
  Halfedge  prev, next, opposite;
  Vertex v;
  Face f;
}
class Vertex{
  Halfedge  e;
  Point p;
}
class Face{
  Halfedge  e;
}
```

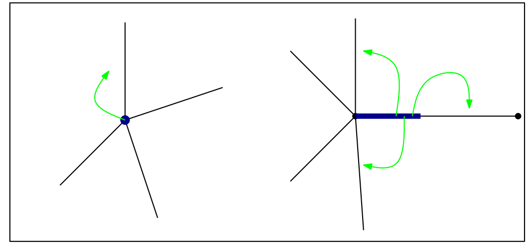
combinatorial information

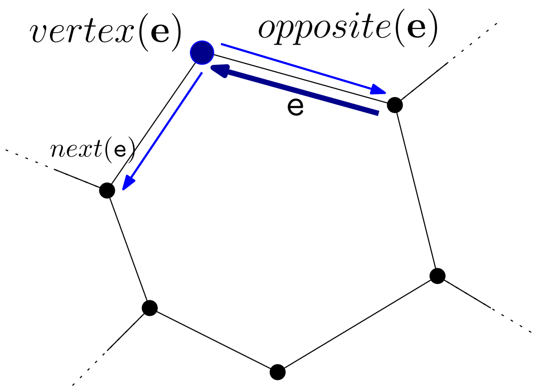
Half-edge data structure: polygonal (orientable) meshes



$$3 \times h + n \approx 3 \times (2e) + n = 18n + n$$

Size (number of references)





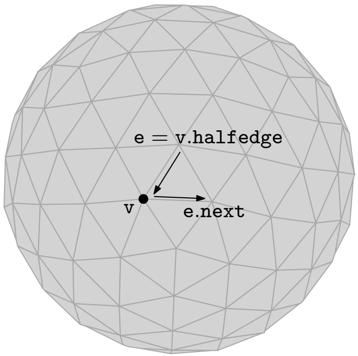

```
class Point{
  double x;
  double y;
}
```

geometric information

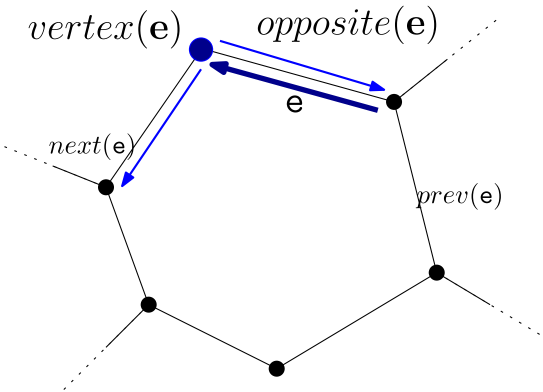
```
class Halfedge{
  Halfedge  prev, next, opposite;
  Vertex v;
  Face f;
}
class Vertex{
  Halfedge  e;
  Point p;
}
class Face{
  Halfedge  e;
}
```

combinatorial information

Half-edge data structure: efficient traversal

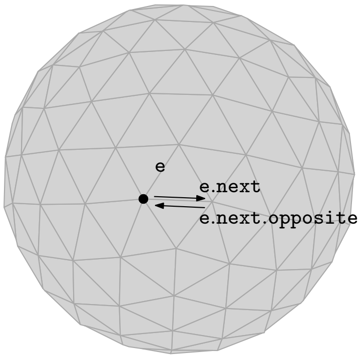



```
public int vertexDegree(Vertex<X> v) {
    int result=0;
    Halfedge<X> e=v.getHalfedge();

    Halfedge<X> pEdge=e.getNext().getOpposite();
    while(pEdge!=e) {
        pEdge=pEdge.getNext().getOpposite();
        result++;
    }

    return result+1;
}
```

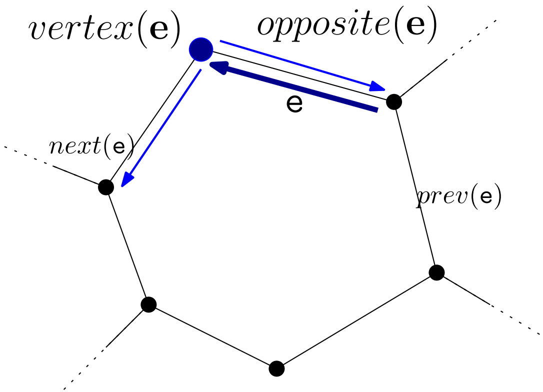



```
public int degree() {
    Halfedge<X> e,p;
    if(this.halfedge==null) return 0;

    e=halfedge; p=halfedge.next;
    int cont=1;
    while(p!=e) {
        cont++;
        p=p.next;
    }
    return cont;
}
```

Half-edge data structure: efficient traversal

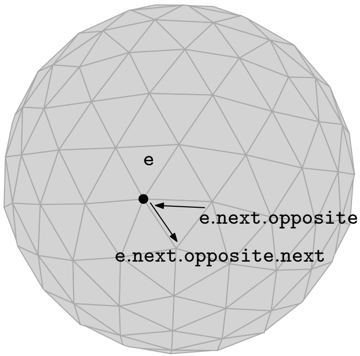



```java
public int vertexDegree(Vertex<X> v) {
    int result=0;
    Halfedge<X> e=v.getHalfedge();

    Halfedge<X> pEdge=e.getNext().getOpposite();
    while(pEdge!=e) {
        pEdge=pEdge.getNext().getOpposite();
        result++;
    }

    return result+1;
}
```

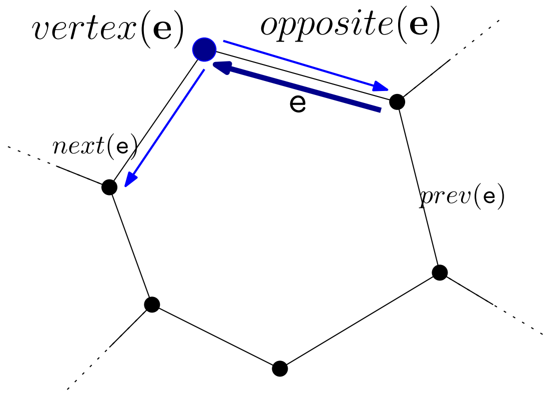



```java
public int degree() {
    Halfedge<X> e,p;
    if(this.halfedge==null) return 0;

    e=halfedge; p=halfedge.next;
    int cont=1;
    while(p!=e) {
        cont++;
        p=p.next;
    }
    return cont;
}
```

Half-edge data structure: efficient traversal

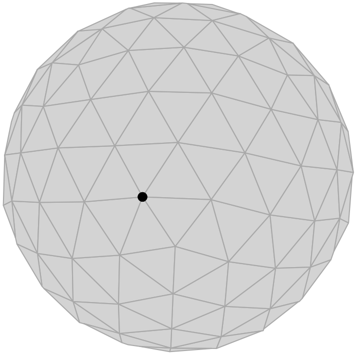


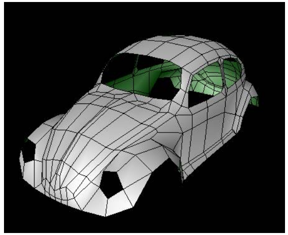

```
public int vertexDegree(Vertex<X> v) {
    int result=0;
    Halfedge<X> e=v.getHalfedge();

    Halfedge<X> pEdge=e.getNext().getOpposite();
    while(pEdge!=e) {
        pEdge=pEdge.getNext().getOpposite();
        result++;
    }

    return result+1;
}
```

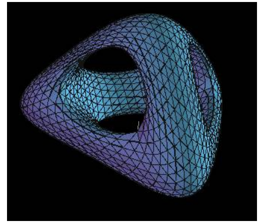


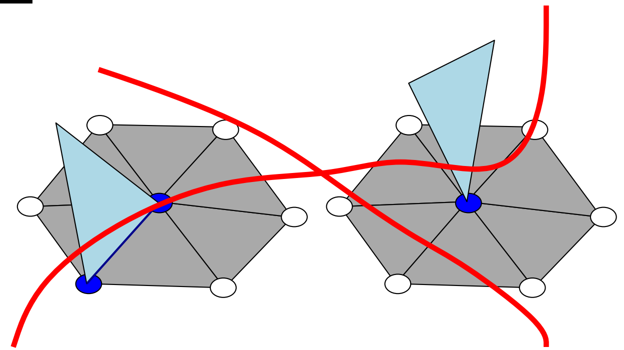

```
public int degree() {
    Halfedge<X> e,p;
    if(this.halfedge==null) return 0;

    e=halfedge; p=halfedge.next;
    int cont=1;
    while(p!=e) {
        cont++;
        p=p.next;
    }
    return cont;
}
```

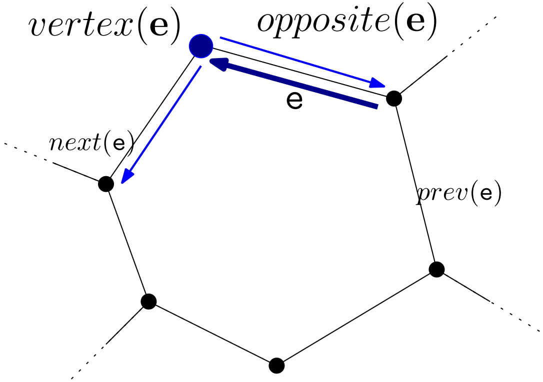
Half-edge data structure: polygonal manifold meshes





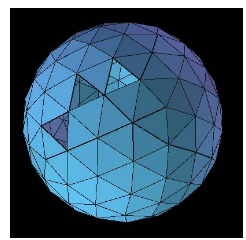


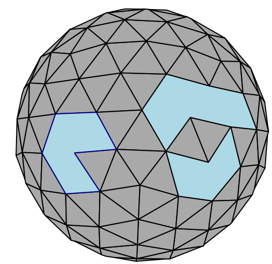




can we represent them?

yes





Triangle based DS: for triangle meshes (used in CGAL)

```
class Point{
  float x;
  float y;
  float z;
}
```

```
class Triangle{
  Triangle t1, t2, t3;
  Vertex v1, v2, v3;
}
 class Vertex{
  Triangle  root;
  Point p;
}
```

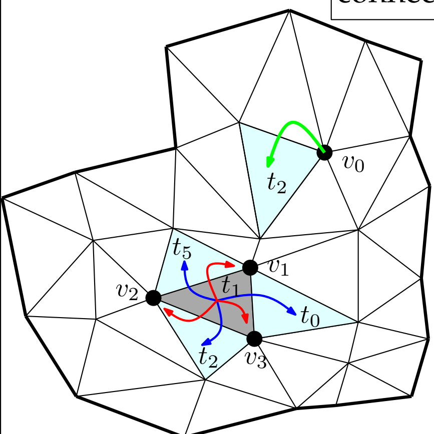
connectivity

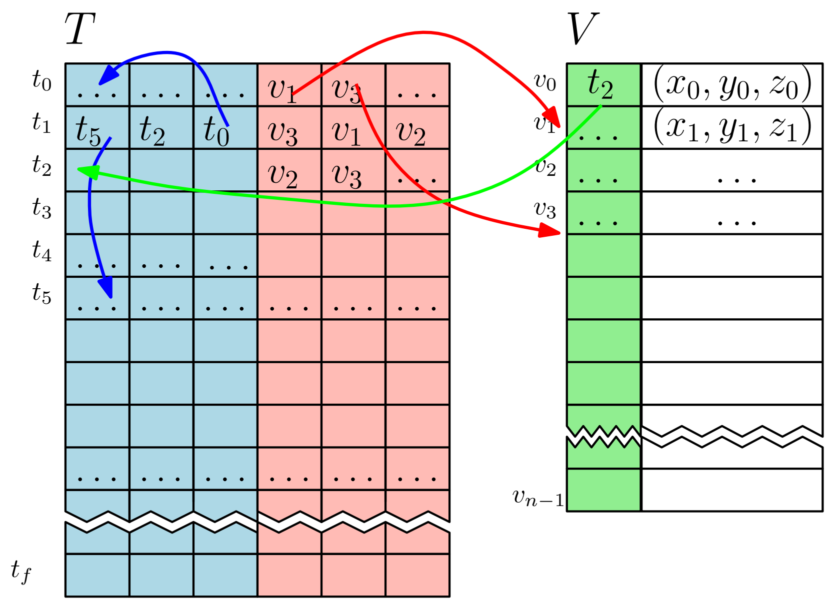
for each triangle, store:

- 3 references to neighboring faces
- 3 references to incident vertices

for each vertex, store:

- 1 reference to an incident face





$$(3+3)\times f + n = 6\times 2n + n = 13n$$

Size (number of references)

Triangle based DS: mesh traversal operators

```
class Point{
  float x;
  float y;
  float z;
}
```

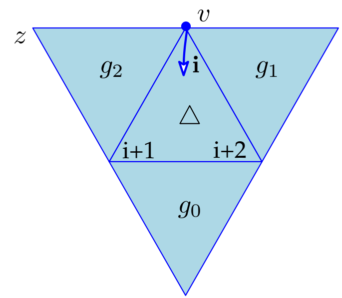
```
class Triangle{
  Triangle t1, t2, t3;
  Vertex v1, v2, v3;
}
 class Vertex{
  Triangle  root;
  Point p;
}
```

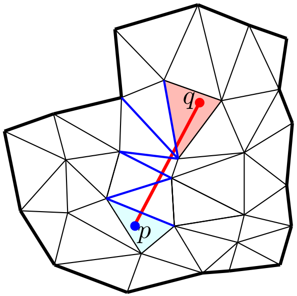
connectivity

the data structure supports the following operators

$$v = \texttt{vertex}(\triangle, i)$$
$$\triangle = \texttt{face}(v)$$
$$i = \texttt{vertexIndex}(v, \triangle)$$
$$g_0 = \texttt{neighbor}(\triangle, i)$$
$$g_1 = \texttt{neighbor}(\triangle, ccw(i))$$
$$g_2 = \texttt{neighbor}(\triangle, cw(i))$$
$$z = \texttt{vertex}(g_2, \texttt{faceIndex}(g_2, \triangle))$$

```
int cw(int i) {return (i + 2)%3; }
int ccw(int i) {return (i + 1)%3; }
```





we can locate a point, by performing a walk in the triangulation

```
int degree(int v) {
   int d = 1;
   int f = face(v);
   int g = neighbor(f, cw(vertexIndex(v, f)));
   while (g != f) {
     int next = neighbor(g, cw(faceIndex(f, g)));
     int i = faceIndex(g, next);
     g = next;
     d++;
   }
   return d;
}
```

we can turn around a vertex, by combining the operators above

Triangle based DS: mesh update operators

```
class Point{
  float x;
  float y;
  float z;
}
```

```
class Triangle{
  Triangle t1, t2, t3;
  Vertex v1, v2, v3;
}
 class Vertex{
  Triangle  root;
  Point p;
}
```

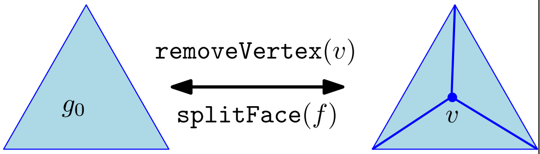
connectivity

the data structure supports the following operators

$\mathtt{removeVertex}(v)$

$\mathtt{splitFace}(f)$

$\mathtt{edgeFlip}(e)$



the data structure is **modifiable**

all these operators can be performed in $O(1)$ time