Geometric Algorithms Fall, 2016

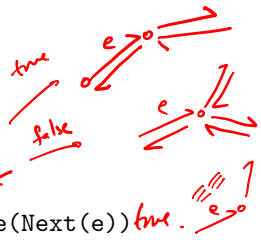
TDR 2 – Meshes

30 November

Questions

1. Which of the following are true:

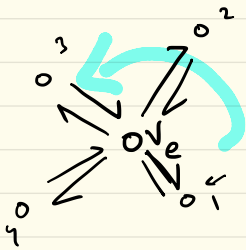
   - `Twin(Twin(e)) = e` true.
   - `Next(Prev(e)) = e` true.
   - `Twin(Prev(Twin(e))) = Next(e)`
   - `IncidentFace(e) = IncidentFace(Next(e))` true.



2. Give an algorithm that lists all vertices adjacent to a given vertex v in a halfedge structure.
3. Write the algorithm to compute the normal of a vertex v by averaging the normals of the faces around v.
4. Write the algorithm to compute the set of silhouette edges of the mesh.
5. Write the algorithm to triangulate a mesh – break each face of the mesh into triangles. In the end, all faces of the mesh should be triangles.
6. Write a method to verify that the mesh has been read and the half-edge data structure computed correctly. What are the possible tests that you can think of to check the correctness of the half-edge data structure?

Give an algorithm that lists all vertices adjacent to a given vertex v in a halfedge structure.

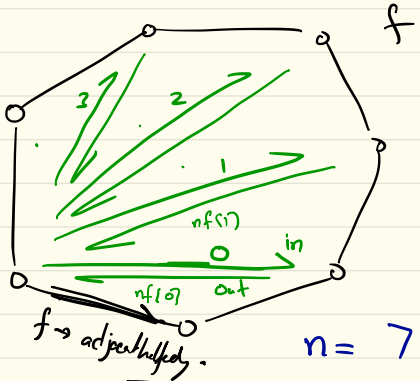
```
myHalfedge *e = V->halfedge;
do {
    output  e->next->source;
    e = e->prev->twin;
}
while ( e != V->halfedge );
```



. Write the algorithm to triangulate a mesh – break each face of the mesh into triangles. In the end, all faces of the mesh should be triangles.

Given: myFace * f.
Output: break f into triangles.

myhalfedge *e = f→adjacenthalfedge;



n ← # of vertices around f.

vector < myhalfedge * > in;
vector < myhalfedge * > out;

```
for ( int i=0 ; i < n-3 ; i++ )
{
    in.push_back ( new myhalfedge() ) ;
    out.push_back ( new ... () ) ;
}
```

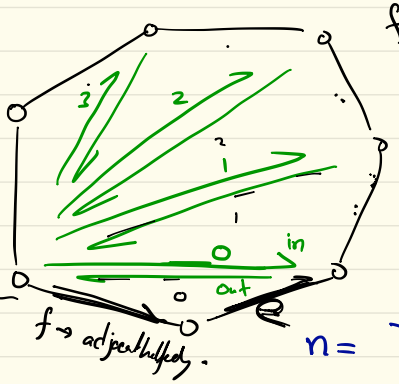
```
vector < myFace * > nf;
for ( i=0 ; i < n-2 ; i++ )
    nf.push_back ( new myFace() );
```

myhalfedge *e = f→adjacenthalfedge→next;

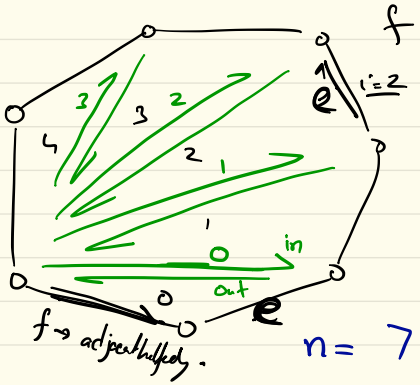
```
for ( int i=0 ; i<n-3 ; i++ )
{
    in[i]→next = e→next ;
    in[i]→prev = out[i+1] ;
    in[i]→twin = out[i] ;
    in[i]→source = f→adjhalf→source ;
    in[i]→adjface = nf[i+1] ;

    out[i]→next = in[i-1] ;
    out[i]→prev = e ;
    out[i]→twin = in[i] ;
    out[i]→source = e→next→source ;
    out[i]→adjface = nf[i] ;

    e = e→next ;
}
```



```
for (i=0; i<n-2; i++)
{
    nf[i]->adjacentHf = out[i];
}
e = f->adja.->next;
for (i=0; i<n-3; i++)
{
    myHalfedge * tmp = e->next;
    e->next = out[i];
    e->prev = out[i]->next;
    e->twin
    e->adjacentFc = nf[i];
}
```

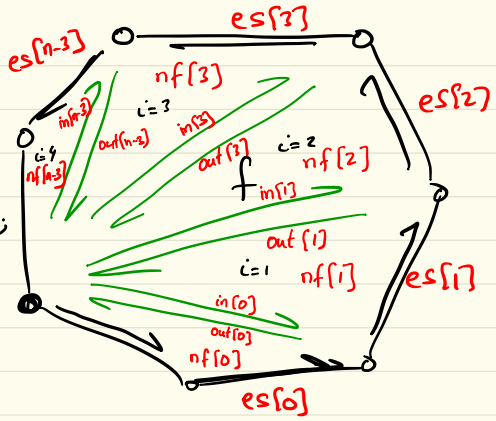


Fix the bourday errors.

1. Compute n.

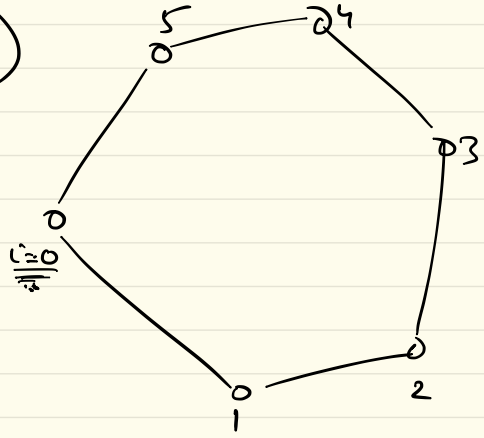
2.

```
vector<myhalfedge *> es;
myhalfedge *t = f->adjacenthalfedge->next;
for (int i=0; i<n-3; i++)
{
    es[i] = e;
    e = e->next;
}
```



Key: remember to fix boundary problems.

```
for ( int i ... )
{
    iplusone= (i+1) % n;
    iminusone= (i-1+n) % n;
    .
    .
    .
```



Geometric Algorithms Fall, 2016

TDR 2 – Meshes

30 November

Questions

1. Which of the following are true:

   - `Twin(Twin(e)) = e` ← true
   - `Next(Prev(e)) = e` ← true
   - `Twin(Prev(Twin(e))) = Next(e)` ← false
   - `IncidentFace(e) = IncidentFace(Next(e))` ← true.

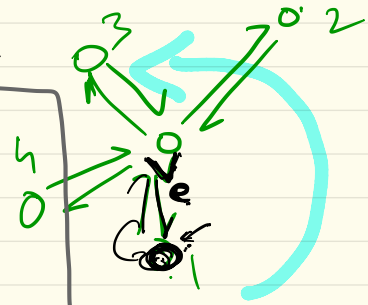
2. Give an algorithm that lists all vertices adjacent to a given vertex v in a halfedge structure.
3. Write the algorithm to compute the normal of a vertex v by averaging the normals of the faces around v.
4. Write the algorithm to compute the set of silhouette edges of the mesh.
5. Write the algorithm to triangulate a mesh – break each face of the mesh into triangles. In the end, all faces of the mesh should be triangles.
6. Write a method to verify that the mesh has been read and the half-edge data structure computed correctly. What are the possible tests that you can think of to check the correctness of the half-edge data structure?

↖ true.

Give an algorithm that lists all vertices adjacent to a given vertex v in a halfedge structure.

Given: myVertex * v

Output: all the neighbor vertices of v.



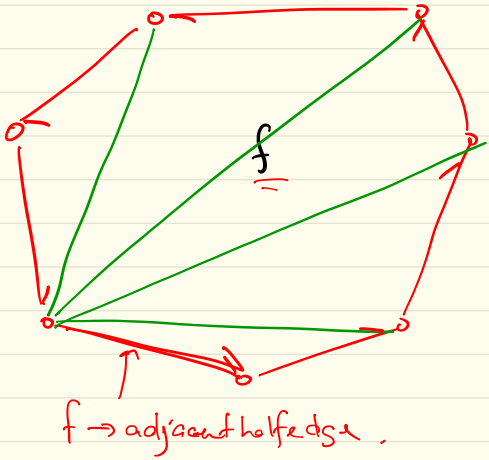

```
myHalfedge *e = v->originof;
do
{
    output  e->twin->source;
    e =     e->prev->twin;
}
while ( e != v->originof );
```

5. Write the algorithm to triangulate a mesh – break each face of the mesh into triangles. In the end, all faces of the mesh should be triangles.

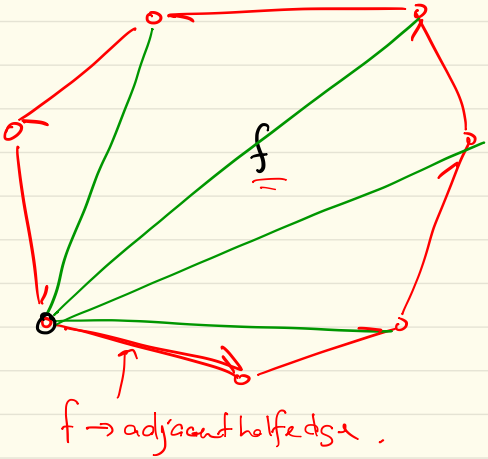
myFace *f

Step1.: Count # of vertices of f.

```
int n=0;
myHalfedge *e = f->adjacentedge;
do {
    n = n+1;
    e = e->next;
} while (e != f->adjacentedge);
```



If (n == 3) we go home.



named e[]'s.

```
int i = 1;
myHalfedge *t = f->adjacentedge->next;
myHalfedge **e = new myHalfedge*[n-2];
do {
    e[i] = t;
    t = t->next;
    i++;
} while ( t != f->adjacenthalfedge->prev );
```