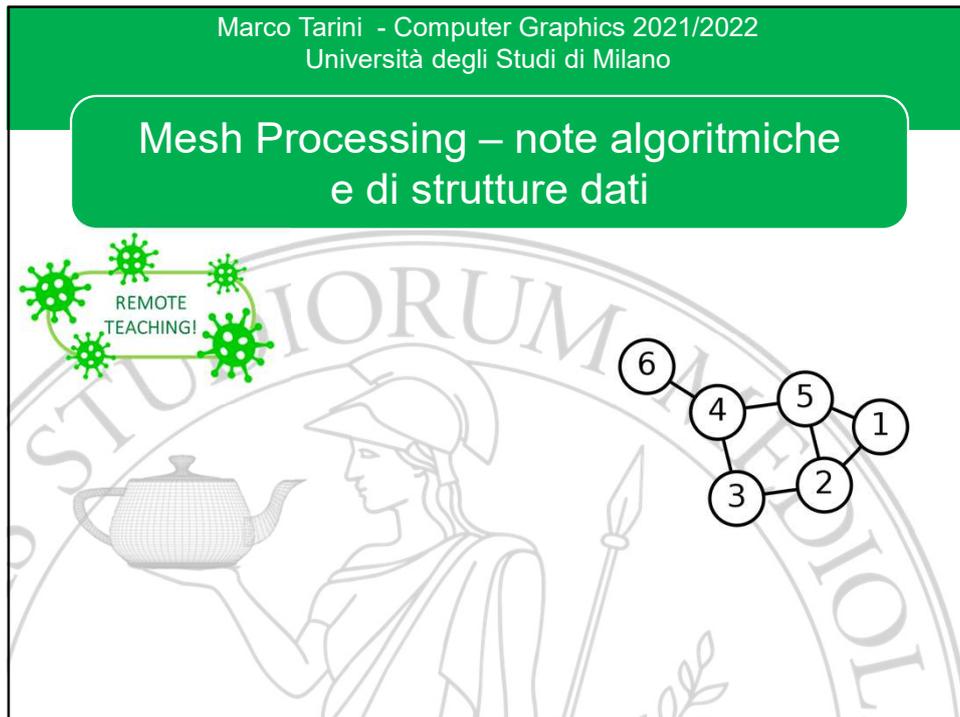


Marco Tarini - Computer Graphics 2021/2022
Università degli Studi di Milano

Mesh Processing – note algoritmiche e di strutture dati

REMOTE TEACHING!



155

Geometry Processing: le basi algoritmiche

- ✓ Gli esempi visti sono solo alcuni dei molti esempi di task affrontati dal geometry processing.
- ✓ Gli algoritmi che risolvono questi task necessitano (quasi tutti) operazioni base sulla connettività come:
 - ⇒ Data una faccia, enumera le facce adiacenti
 - ⇒ Dato un vertice, elenca tutte le facce che includono quel vertice
 - ⇒ Dato un vertice, enumera tutti i vertici che sono connessi a quel vertice da un edge (la «stella» del vertice)
 - ⇒ Dato una faccia, elenca tutti i vertici che fanno parte di quella faccia
 - ⇒ Dato un edge, scopri se è un edge di bordo.
 - ⇒ Dato un edge di bordo, enumera tutti gli altri edge che fanno parte di quel bordo



156

Mesh Processing: le basi algoritmiche

- ✓ Gli esempi visti sono solo alcuni dei molti esempi di task affrontati nel contesto del mesh processing.
- ✓ Gli algoritmi che risolvono questi task necessitano (quasi tutti) operazioni base sulla connettività come per esempio:
(operazioni di navigazione su mesh)
 - ⇒ Data una faccia, enumera tutte le facce adiacenti (separate da un edge)
 - ⇒ Dato una faccia ed un edge, trova la faccia (se esiste) che sta dall'altra faccia di quell'edge
 - ⇒ Dato un vertice, elenca tutte le facce che includono quel vertice (detta la «stella» o «stella-1» del vertice)
 - ⇒ Dato un vertice, enumera tutti i vertici che sono connessi a quel vertice da un edge
 - ⇒ Dato un edge, scopri se è un edge di bordo.
 - ⇒ Dato un edge di bordo, enumera tutti gli altri edge che fanno parte di quel bordo
 - ⇒ Dato una faccia, elenca tutti i vertici che fanno parte di quella faccia
- ✓ E' necessario che queste operazioni base siano effettuate efficientemente (idealmente, in tempo costante)



157

Strutture dati per Mesh Processing

- ✓ La struttura «lista facce» della mesh indexed, usata per memorizzare la connettività non consente di effettuare queste operazioni
 - ⇒ (se non attraverso una scansione dell'intero vettore delle facce, che richiede ovviamente un tempo lineare col numero di facce)
 - ⇒ (eccetto: «data una faccia, elenca tutti i vertici che fanno parte di quella faccia»)
- ✓ Per effettuare mesh processing, dobbiamo dotarci di strutture più adatte per memorizzare la connettività di una mesh
 - ⇒ svantaggio: più prolisse, onerose da mantenere durante le modifiche
 - ⇒ ma consentono di navigare sulla mesh molto più agevolmente
- ✓ Vediamo una di queste strutture



158

Struttura dati: connettività di una indexed mesh

$F_0 \rightarrow$	0, 2, 1
$F_1 \rightarrow$	1, 2, 5, 4
$F_2 \rightarrow$	2, 3, 5

CONNETTIVITA'
(come lista facce)

159

Struttura dati: half edges

	V	F	Opp
$H_0 \rightarrow$	2	-	1
$H_1 \rightarrow$	0	0	0

161

Struttura dati: half edges

	V	F	Next	Opp
H ₀ →	2	-	2	1
H ₁ →	0	0	3	0
H ₂ →	0	-		
H ₃ →	2	0		

162

Struttura dati: half edges

	V	F	Next	Opp
H ₀ →	2	-	2	1
H ₁ →	0	0	3	0
H ₂ →	0	-		5
H ₃ →	2	0	5	4
H ₄ →	1	1		3
H ₅ →	1	0	1	2

163

Struttura dati: half edges

	V	F	Next	Opp
H ₀ →	2	-	2	1
H ₁ →	0	0	3	0
H ₂ →	0	-	6	5
H ₃ →	2	0	5	4
H ₄ →	1	1	11	3
H ₅ →	1	0	1	2
H ₆ →	1	-	7	12
H ₇ →	4	-	10	15
H ₈ →	5	2	14	11
H ₉ →	3	2	8	10
H ₁₀ →	5	-	13	9
H ₁₁ →	2	1	15	8
H ₁₂ →	4	1	4	6
H ₁₃ →	3	-	0	14
H ₁₄ →	2	2	9	13
H ₁₅ →	5	1	12	7

164

Lista di half edge

- ✓ La connettività è rappresentata da un vettore di Half Edge
- ✓ Per ogni half edge memorizzo i campi (sono tutti indici):
 - ⇒ Indice di Vertice: da quale vertice parte
 - ⇒ Indice di Faccia: di quale faccia è un bordo
 - ⇒ Next: l'indice dell'half-edge che incontro proseguendo nella direzione dell'half edge (senza cambiare faccia o bordo della mesh)
 - ⇒ Opposite: indice all'altro half-edge che condivide lo stesso edge
- ✓ Strutture a contorno:
 - ⇒ In ogni vertice, posso memorizzare l'indice di un Half-Edge che parte da quell vertice (uno qualsiasi)
 - ⇒ Posso avere un vettore di facce, per ogni faccia l'indice di un half edge appartenente a quella faccia (uno qualsiasi)



166

HalfEdge: pseudocodice Java

```
class HalfEdge {  
    int vi;    // indice di vertice  
    int fi;    // indice di faccia (o -1)  
    int next; // indice di halfHedge  
    int opp;   // indice di halfHedge  
}  
  
// la tabella  
HalfEdge[] he = new HalfEdge( .... );
```

ed es, il valore *opposite*
del halfedge di indice 12 è

```
he[12].opp;
```

ed es, l'half edge di indice
7 fa parte della faccia

```
he[7].fi;
```

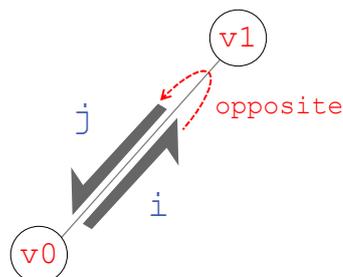


167

Esempio di uso base

- ✓ dato un halfedge di indice i ,
quali sono i due vertici v_0 e v_1 che delimitano l'edge
corrispondente?

```
int v0 = he[i].vi;  
int j = he[i].opp;  
int v1 = he[j].vi;
```



168

Strutture dati per connettività a confronto

- ✓ Lista facce:
 - ⇒ Compatta
(quindi, adatta a storing, ad esempio su disco o streaming)
 - ⇒ Sufficiente per ad alcuni task di processing
(e in questo caso, preferibile)
 - ⇒ Il redering classico eseguito dalle GPU
è pensato per questa struttura dati
 - ⇒ E' generale: non richiede ad esempio two-manifoldness o orientamento
consistente delle facce
(quindi: capace di rappresentare strutture inconsistenti – è un vantaggio
e uno svantaggio)
 - ⇒ Lista di elementi non omogenea, (alcune facce hanno un numero di
vertici diverso da altre).
eccetto che per tri-mesh o pure quad meshes: per loro, la lista facce è
una matrice $3 \times N$ o $4 \times N$ (comodo)



169

Strutture dati per connettività a confronto

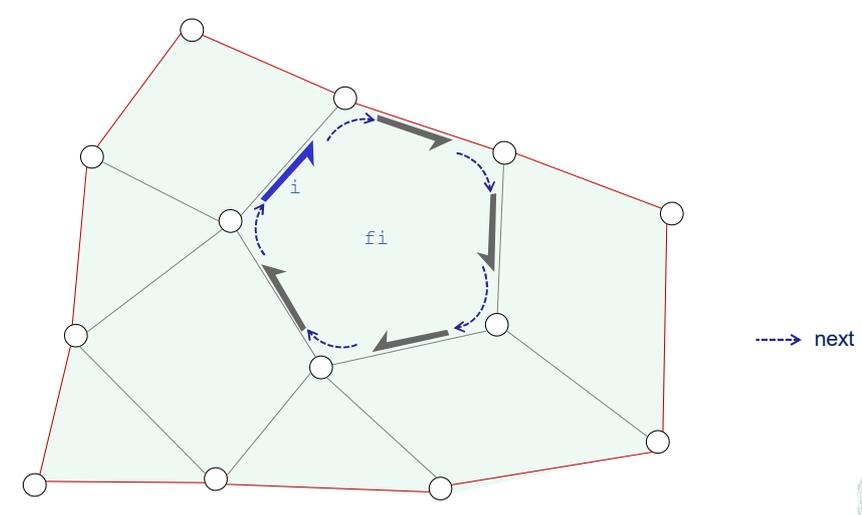
- ✓ Lista di half hedge:
 - ⇒ Prolissa
(calcolo: quanti interi è necessario memorizzare in media, rispetto ad
una lista facce?
Ipotesi: in una tri mesh, ho vertici, facce, edge tipicamente in
proporzione 1x, 2x, 3x. In una quad mesh: 1x, 1x, 2x)
 - ⇒ Più complicata da mantenere coerente durante le operazioni di modifica
della connettività
 - ⇒ Non adatta per il rendering su GPU
 - ⇒ Vantaggio: lista di elementi sempre omogenea: 4 elementi per half-edge
(nella variante che abbiamo visto), anche su mesh poligonali miste
 - ⇒ Consente di «navigare sulla mesh», con salti all'elemento adiacente
in tempo costante (consentendo algoritmi di mesh processing in tempo
lineare o pseudolineare piuttosto che quadratico)
 - ⇒ Richiede adattamenti se la mesh non è two-manifold e ben orientata
- ✓ Nota: sono due rappresentazioni alternative di una stessa cosa
(la connettività della mesh).
 - ⇒ Una si può costruire a partire dall'altra



170

Esempio: conta quanti lati ha una faccia

✓ Dato un half-edge di indice i , corrispondente ad una data faccia, trova quanti lati ha questa faccia



-----> next



171

Esempio: conta quanti lati ha una faccia

✓ Dato un half-edge di indice i , corrispondente ad una data faccia, trova quanti lati ha questa faccia

```
int fi = he[i].fi;
if (fi == -1) ... /* non esiste la faccia */
int start = i; // half edge di partenza
int lati = 0;
do {
    lati ++;
    i = he[i].next;
} while (i!=start);
```

(era un half edge di bordo)

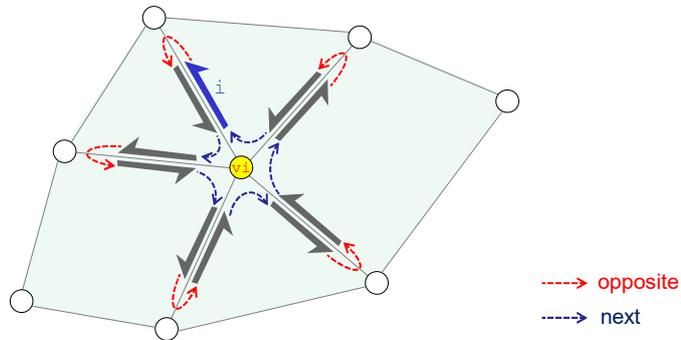
(nota: il ciclo finisce quando torno al punto di partenza)



172

Esempio: scansione di vertici attorno ad un vert

- ✓ Dato un half-edge di indice i , che emana da un vertice v_i , trova tutti i vertici v_j nella stella di v_i



(nota: il ciclo finisce quando torno al punto di partenza)



173

Esempio: scansione di vertici attorno ad un vert

- ✓ Dato un half-edge di indice i , che emana da un vertice v_i , trova tutti i vertici v_j nella stella di v_i

```
int vi = he[i].vi;  
  
int start = i; // half edge di partenza  
  
do {  
    i = he[i].opp;  
    int vj = he[i].vi;  
  
    /* qui: fai qualcosa con vertice vj */  
  
    i = he[i].next;  
} while (i!=start);
```

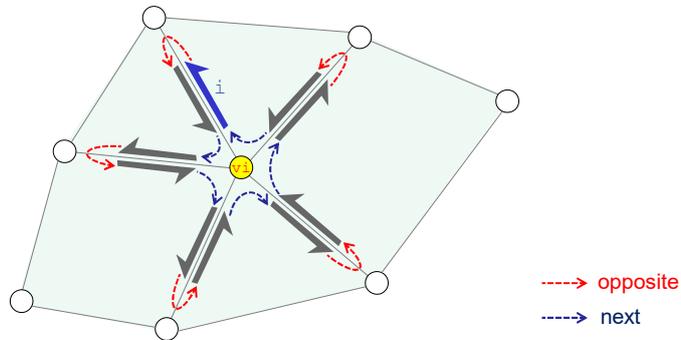
(nota: il ciclo finisce quando torno al punto di partenza)



174

Scansione di facce attorno ad un vert

- ✓ Dato un half-edge di indice i , che emana da un vertice v_i , trova tutte le faccie f_j adiacenti a v_i



(nota: il ciclo finisce quando torno al punto di partenza)



175

Esempio: scansione di facce attorno ad un vert

- ✓ Dato un half-edge di indice i , che emana da un vertice v_i , trova tutte le faccie f_j adiacenti a v_i

```
int vi = he[i].v;  
  
int start = i; // half edge di partenza  
  
do {  
    int fi = he[i].fi;  
    /* qui: fai qualcosa con faccia fi */  
    /* se esiste: potrebbe essere -1 */  
  
    i = he[i].opp;  
    i = he[i].next;  
} while (i!=start);
```

(nota: il ciclo finisce quando torno al punto di partenza)



176

Esempio: contare la faccie adiacenti da una faccia

✓ dato un indice di halfedge i , se confina con una faccia f_i , allora conta quante_facce adiacenti a f_i ci sono

-----> opposite
-----> next

177

Esempio: contare la faccie adiacenti da una faccia

✓ dato un indice di halfedge i , se confina con una faccia f_i , allora conta quante_facce adiacenti a f_i ci sono

```
int fi = he[i].f;  
if (fi == -1) ... /* non esiste la faccia */  
int start = i;  
int quante_facce = 0;  
do {  
    int j = he[i].opp;  
    if (he[j].fi != -1) quante_facce ++;  
    i = he[i].next;  
} while (i != start);
```

178

Esempio: visita di un bordo (insieme di edge)

✓ Sia dato un indice di halfedge i , confinante con una faccia f_i ;
se l'edge corrispondente è di bordo, allora scandisci tutti gli edge che fanno parte dello stesso bordo:

-----> opposite
-----> next



179

Esempio: visita di un bordo (insieme di edge)

✓ Sia dato un indice di halfedge i , confinante con una faccia f_i ;
se l'edge corrispondente è di bordo, allora scandisci tutti gli edge che fanno parte dello stesso bordo:

```
i = he[i].opp;  
int fi = he[i].f;  
if (fi == -1) {  
    int start = i;  
    do {  
        int i = he[i].next;  
        /* fa qualcosa con i... */  
    } while (i!=start);  
}
```



180

Esempio: visita di un bordo (insieme di edge)

Funziona anche su mesh con "dangling edges" (edges non adiacenti ad alcuna faccia)

campo next

181

Strutture per connettività a confronto: sommario

INDEXED	HALF-EDGES
✓ HW friendly	✓ Rendering... complicato
✓ Navigazione... complicata ⇒ richiede strutture dati ulteriori ("di adiacenza")	✓ Navigazione semplice ⇒ es: trovare tutti i vertici nella "1-star" di un vertice
✓ Ideale solo per mesh "pure" ⇒ tri-meshes, pure quad meshes	✓ poligoni misti a piacere
✓ Compatta: 3 indici x tri	✓ Prolissa: 12 indici per tri
✓ > adatta per rendering	✓ > adatta per geometry processing

182