

# Lezione 6

## Triangolazioni e relative strutture dati

1

## Triangolazione

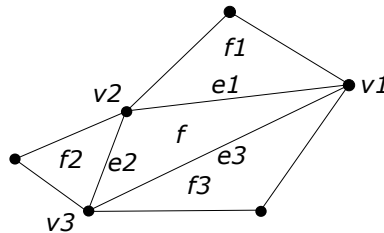
- ❖ Una triangolazione è una suddivisione piana in cui tutte le facce finite sono triangoli
- ❖ Non è necessariamente una suddivisione massimale perché il contorno della faccia infinita può essere anche un poligono non convesso

2

## Relazioni topologiche in una triangolazione

Sono le stesse che abbiamo in una suddivisione piana generica, ma:

- ❖ Le relazioni basate su facce FV, FE, FF sono *costanti* su tutte le facce finite (tre elementi)
- ❖ Le uniche relazioni non costanti sono quelle basate su vertici
- ❖ La faccia infinita si tratta come caso a parte



3

## Operazioni di editing sulle triangolazioni

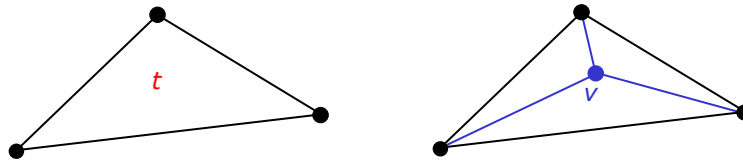
- ❖ È sempre possibile applicare operatori di Eulero
- ❖ Ci sono alcuni operatori di modifica interessanti:
  - ❖ Operatori di raffinamento:
    - ❖ producono una triangolazione con più vertici/spigoli/facce
  - ❖ Operatori di semplificazione:
    - ❖ producono una triangolazione con meno vertici/spigoli/facce
- ❖ Potrebbero essere prodotti mediante sequenze di operatori di Eulero, ma conviene implementarli direttamente
- ❖ Inoltre questi operatori sono chiusi rispetto alle triangolazioni (salvo casi particolari)

4

## Operatori di raffinamento

### Triangle split:

- ❖ Si inserisce un nuovo vertice  $v$  in un triangolo  $t$  e lo si divide in tre collegando  $v$  ai vertici di  $t$



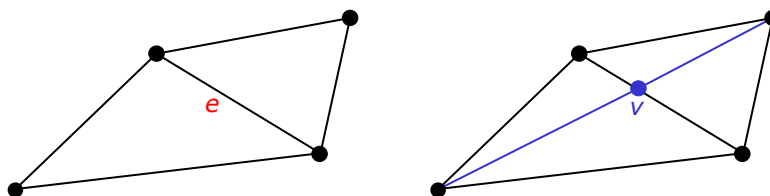
Nota: un operazione di questo tipo si può fare in qualunque suddivisione con facce stellate e si chiama *starring*

5

## Operatori di raffinamento

### Edge split:

- ❖ Si inserisce un nuovo vertice  $v$  su di uno spigolo  $e$  e si dividono a metà i due triangoli adiacenti ad  $e$  collegando  $v$  ai vertici opposti ad  $e$

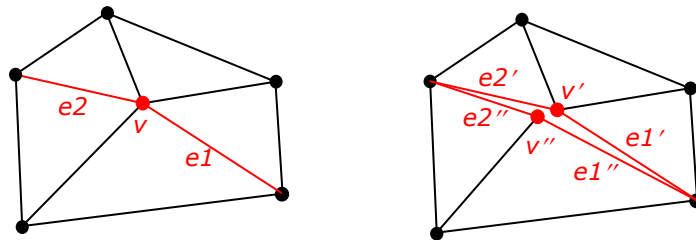


6

## Operatori di raffinamento

Vertex split:

- ❖ Si considera un vertice  $v$  e due dei suoi spigoli incidenti  $e1$  ed  $e2$
- ❖ Si "taglia" la triangolazione lungo  $e1$  ed  $e2$
- ❖ Si duplicano  $v$ ,  $e1$  ed  $e2$  ciascuno in due copie
- ❖ ...

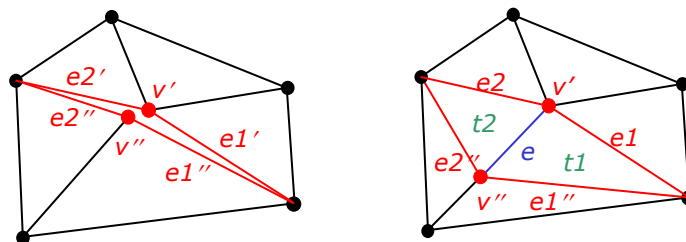


7

## Operatori di raffinamento

Vertex split:

- ❖ ...
- ❖ Si sposta una o entrambe le copie di  $v$
- ❖ Si collegano le due copie di  $v$  con un nuovo spigolo  $e$ , generando due nuove facce  $t1$  e  $t2$

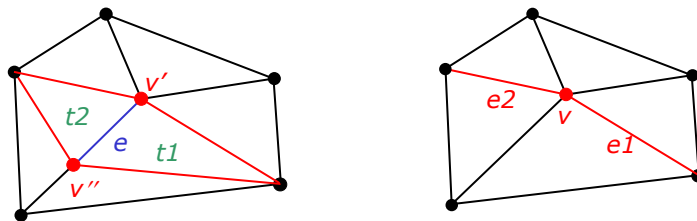


8

## Operatori di semplificazione

Edge collapse (inverso di vertex split):

- ❖ Si prende uno spigolo  $e$  e si elimina facendo collassare i suoi vertici estremi in un solo vertice
- ❖ Le due facce incidenti su  $e$  vengono eliminate

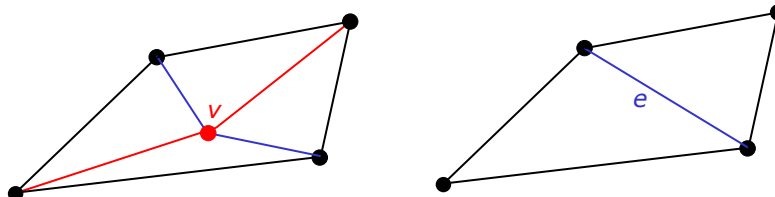


9

## Operatori di semplificazione

Edge merge (inverso di edge split):

- ❖ si considera un vertice  $v$  interno con 4 triangoli incidenti
- ❖ si cancellano i 4 triangoli e si creano 2 triangoli tracciando uno spigolo  $e$  coincidente con una delle diagonali del quadrilatero

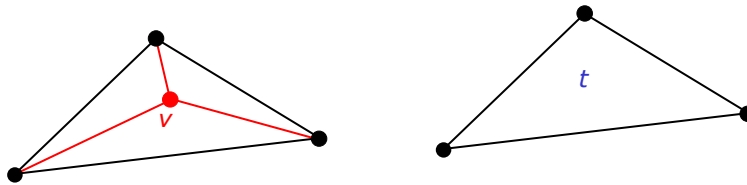


10

## Operatori di semplificazione

Delete vertex (inverso di triangle split):

- ❖ si considera un vertice  $v$  interno con 3 triangoli incidenti
- ❖ si fondono i 3 triangoli in un unico triangolo  $t$

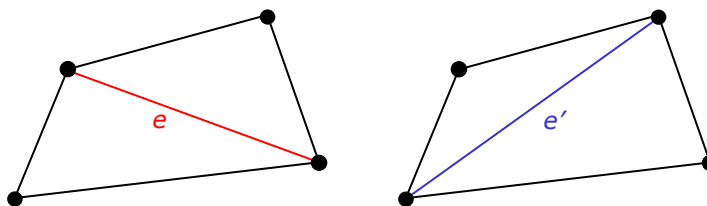


11

## Altro Operatore di modifica

Edge swap:

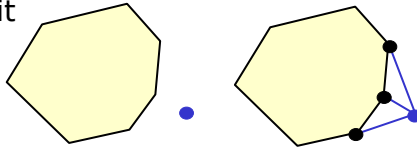
- ❖ si considera uno spigolo  $e$  su cui incidono due triangoli la cui unione forma un quadrilatero convesso  $Q$
- ❖ si sostituisce lo spigolo  $e$  con l'altra diagonale  $e'$  di  $Q$



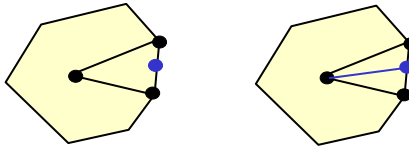
12

## Casi particolari (bordo)

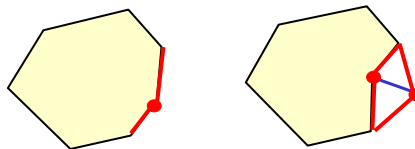
❖ Triangle split



❖ Edge split



❖ Vertex split



❖ Edge swap: NO

13

## Strutture dati per triangolazioni

Sono strutture che modificano o estendono quelle viste per le suddivisioni, in modo da sfruttare le caratteristiche specifiche delle triangolazioni:

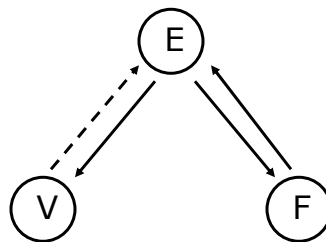
- ❖ facce con tre spigoli/vertici/vicini
- ❖ alcune relazioni costanti in più

14

## La struttura *simmetrica semplificata*

Idea:

- ❖ Manteniamo tutte le relazioni costanti della simmetrica
- ❖ Manteniamo la relazione VE (non costante) solo in modo parziale



15

## La struttura *simmetrica semplificata*

Implementazione dinamica

```
typedef struct vertex
{
    float x,y;
    V_attr attr;
    struct edge * VE;
} Vertex;

typedef struct edge
{
    E_attr attr;
    struct vertex * EV[2];
    struct face * EF[2];
} Edge;
```

16

## La struttura *simmetrica semplificata*

### Implementazione

```
typedef struct face
{
    F_attrib attr;
    struct edge FE[3];
} Face;
```

```
Vertex V[n];
Edge E[e];
Face F[f];
```

17

## La struttura *simmetrica semplificata*

Complessità spaziale struttura dinamica  
(tralasciando il costo degli attributi):

- ❖ Ogni record Vertex ha costo 3 (2 geometria + 1 topol.)
  - ❖ Abbiamo un array di n Vertex – costo 3n
- ❖ Ogni record Edge ha costo 4
  - ❖ Abbiamo un array di e Edge – costo 4e
- ❖ Ogni record Face ha costo 3
  - ❖ Abbiamo un array di f Face – costo 3f
- ❖ In totale abbiamo  $3n + 4e + 3f \approx 21n$  (di cui 2n per la geometria e 19n per la topologia)
  - ❖ Circa la metà della simmetrica standard!

18

## La struttura *simmetrica semplificata*

### Valutazione delle relazioni topologiche:

- ❖ EV, EF, FE: codificate – ottimali
- ❖  $VE = VE^* + EF + FE + EV$  in tempo ottimale:
  - ❖ Dato  $v$ , si calcola  $e := VE^*(v)$
  - ❖  $(f1, f2) := EF(e)$
  - ❖  $(v1, v2) := EV(e)$
  - ❖ if  $(v = v1)$  then  $f := f1$  else  $f := f2$
  - ❖  $(e1, e2, e3) := FE(f)$
  - ❖ lo spigolo che segue  $e$  in ordine ciclico nella terna  $(e1, e2, e3)$  è il prossimo spigolo di  $VE(v)$
  - ❖ si ripete a partire dallo spigolo trovato finché si torna su  $e$
- ❖ Notare che EE si calcola con lo stesso algoritmo della simmetrica standard ma adesso in tempo ottimale

19

## La struttura *simmetrica semplificata*

### Supporto alle altre operazioni di analisi:

- ❖ Come la simmetrica standard

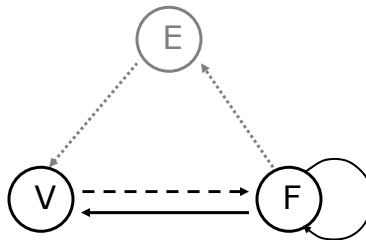
### Supporto alle operazioni di editing:

ALLA LAVAGNA

20

## La struttura *indicizzata con adiacenze*

- ❖ Stessa base della struttura indicizzata:
  - ❖ array di vertici
  - ❖ array di facce, ogni faccia punta ai suoi tre vertici (FV)
- ❖ In più:
  - ❖ ogni faccia punta alle facce ad essa adiacente (FF)
  - ❖ ogni vertice punta ad una faccia incidente (VF\*)



21

## La struttura *indicizzata con adiacenze*

Complessità spaziale:

- ❖ n celle di costo  $2+1$  per l'array V
- ❖ f celle di costo 6
  
- ❖ In totale abbiamo  $3n + 6f \approx 15n$  (di cui  $2n$  per la geometria)
- ❖ Circa  $3/4$  dello spazio occupato dalla simmetrica semplificata

22

## La struttura *indicizzata con adiacenze*

Valutazione delle relazioni topologiche:

- ❖ FV, FF codificate – ottimale
- ❖ FE implicita
- ❖  $VF = VF^* + FF + FV$  ottimale (per esercizio, simile alla simmetrica, più semplice)
- ❖ VV, VE: simili alla VF, ottimali considerando che tutte le facce sono triangoli
- ❖ Relazioni basate su spigoli (gli spigoli esistono solo in modo implicito come coppie di vertici):
  - ❖ EV implicita
  - ❖ EF/EE: bisogna intersecare le relazioni VF dei due vertici estremi..... - non ottimale – lineare nel numero di spigoli/facce incidenti nei due vertici

23

## La struttura *indicizzata*

Operazioni di editing:

ALLA LAVAGNA

24

## La struttura *indicizzata con adiacenze*

### Valutazione globale:

- ❖ Struttura più compatta della simmetrica semplificata
- ❖ Non supporta gli spigoli (attributi) e la valutazione delle relazioni basate su spigoli (EE ed EF)
- ❖ Conviene se non si vogliono mantenere attributi sugli spigoli e si naviga per adiacenze tra triangoli

25

## Riferimenti

- ❖ Dispense:
  - ❖ Capitolo 3
- ❖ Preparata-Shamos:
  - ❖ Capitolo 1
- ❖ de Berg, Van Kreveld, Overmars, Schwarzkopf:
  - ❖ Capitolo 2

26