

Corso di *Grafica Computazionale*

Rappresentazione di Oggetti Tridimensionali

**Docente:
Massimiliano Corsini**

Laurea Specialistica in Ing. Informatica

Facoltà di Ingegneria

Università degli Studi di Siena

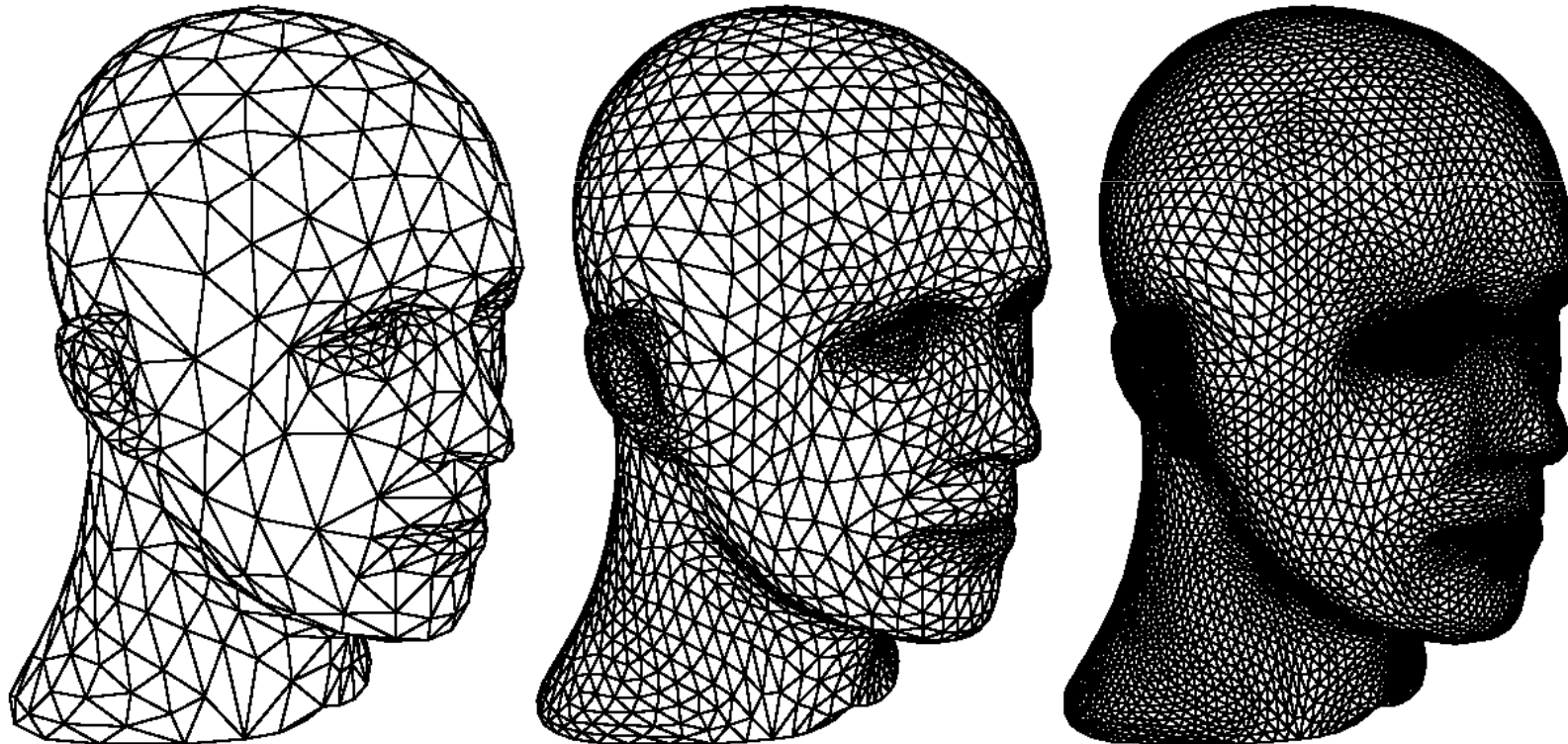


Superfici di Suddivisione



Idea base

- L'idea base è rendere *smooth* le superfici poligonali definite tramite mesh usando delle regole di raffinazione della mesh di partenza.
- Le *tecniche di suddivisione* permettono quindi di colmare il gap tra rappresentazione *discreta* e *continua*.





Gerì's Game (1989)

Facoltà di
Ingegneria



Images from *Gerì's Game*, Pixar Animation Studios

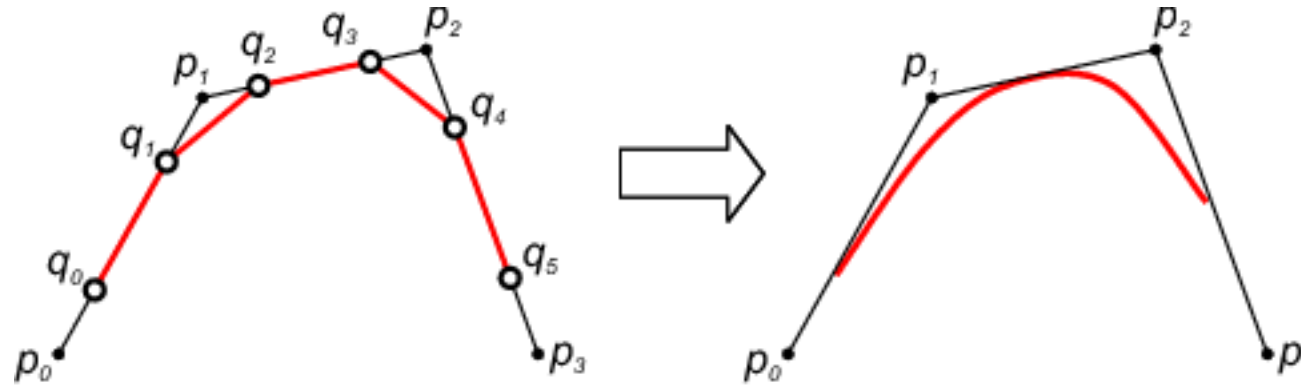


Algoritmo di Chaikin

- La curva iniziale (livello 0) è formata da un insieme di punti di controllo $P_0, P_1, P_2, \dots, P_n$
- Ad ogni passo di suddivisione vengono creati due nuovi vertici tra i punti P_i e P_{i+1} secondo la seguente regola di suddivisione:

$$q_{2i}^{k+1} = \frac{3}{4}p_i^k + \frac{1}{4}p_{i+1}^k$$
$$q_{2i+1}^{k+1} = \frac{1}{4}p_i^k + \frac{3}{4}p_{i+1}^k$$

- Ad ogni passo i vertici “vecchi” (livello k) vengono scartati; la curva di livello $k+1$ è formata soltanto dai nuovi vertici.



First step of subdivision

Limit curve

- La *curva limite* della suddivisione di Chaikin è una B-Spline quadratica.



Superfici di Suddivisione

- Metodi di suddivisione per la generazione di superfici partano da una mesh di controllo M^0 ed applicano iterativamente la regola di suddivisione ottenendo le mesh di livello M^k .
- Noi ci occuperemo di metodi di suddivisioni *stazionari* (la regola di suddivisione rimane la stessa a tutti i livelli di suddivisione).

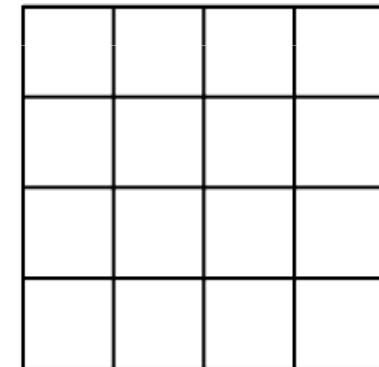
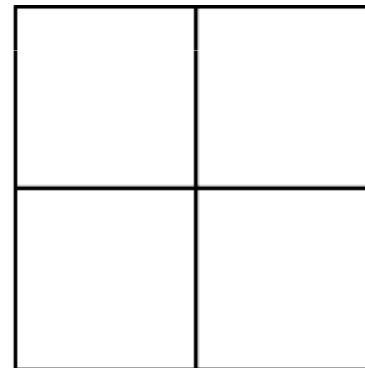
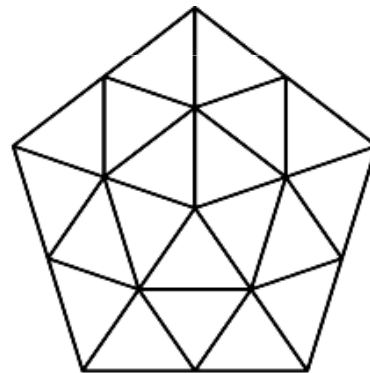
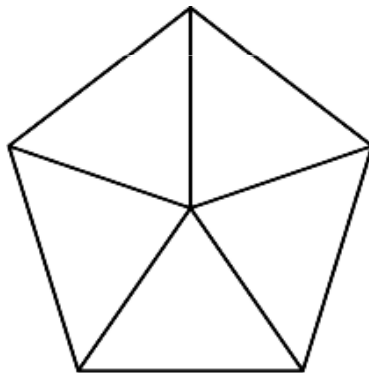


- **Tipo di mesh:** triangolare o quadrangolare.
- **Tipo di regola:** *primale* (face-splitting) o *duale* (vertex-splitting).
- **Approssimante o interpolante.**
- **Smoothness:** proprietà di continuità della superficie limite (C^0 , C^1 , ...).



Schemi Primali

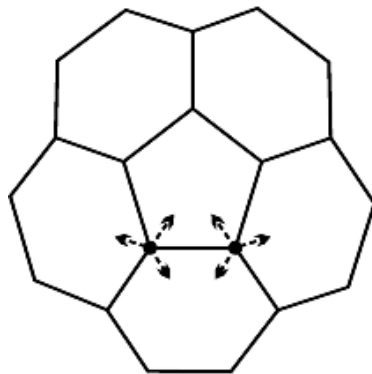
- Per creare le nuove facce della mesh si introduce un nuovo vertice per ogni lato e si uniscono tali vertici in modo da creare le nuove facce.
- Questa operazione viene anche detta 1-to-4 split.



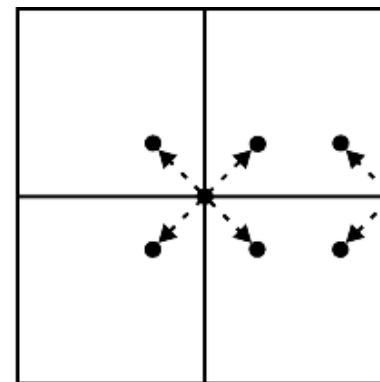
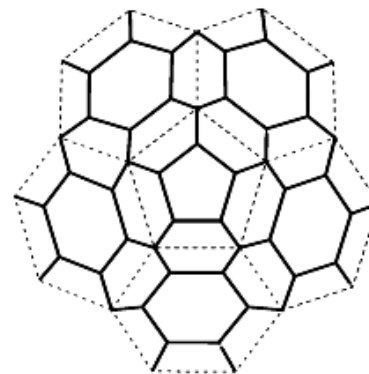
**Schema Primale
per mesh triangolare**

**Schema Primale
per mesh quadrangolare**

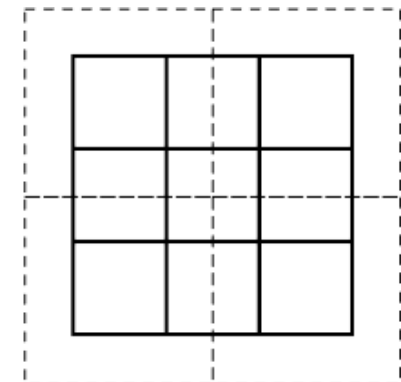
- Ogni vertice viene splittato in modo da creare un nuovo vertice per ogni faccia adiacente a questo. Ogni nuova faccia viene creata all'interno della vecchia faccia.
- Nel caso di mesh triangolari lo split dei vertici risulta in una mesh formata da elementi esagonali.



**Schema Duale
per mesh triangolare**



**Schema Duale
per mesh quadrangolare**





Schemi di Suddivisione

Schema di Suddivisione	Caratteristiche			Continuità
Butterfly	Interp.	Triang.	Primale	C^1
Catmull-Clark	Approx.	Quadr.	Primale	C^2
Doo-Sabin	Approx.	Quadr.	Duale	C^1
Kobbelt	Interp.	Quadr.	Primale	C^1
Kobbelt ($\sqrt{3}$)	Approx.	Triang.	Duale	C^2
Loop	Approx.	Triang.	Primale	C^2



Terminologia

- Per una mesh triangolare i vertici generati dalla suddivisione assumono valenza 6. I vertici di una mesh quadrangolare assumono valenza 4. Tali vertici sono detti vertici *regolari*.
- Una mesh si dice *semi-regolare* se, ad eccezione dei vertici iniziali, tutti i suoi vertici sono regolari.
- I vertici non regolari sono chiamati *vertici straordinari*.
- Moltissimi schemi di suddivisione possono essere descritti tramite maschere di pesi.
- Quasi tutti gli schemi di suddivisione devono essere adattati sui bordi.

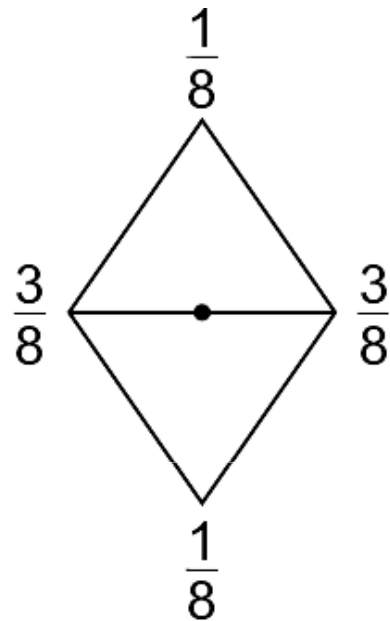


Schema Loop

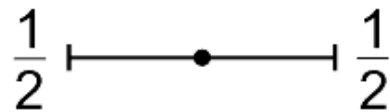
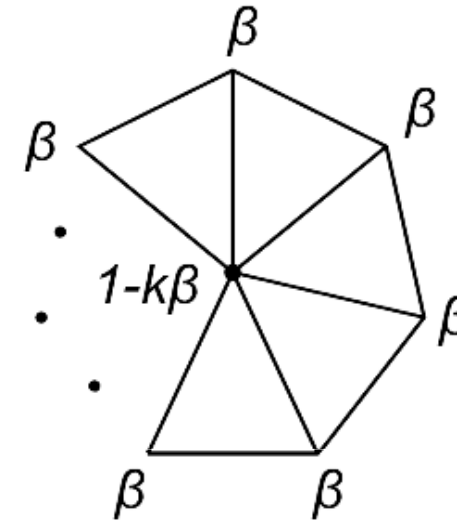
- È uno schema approssimante per mesh triangolari proposto da Charles Loop nel 1987.
- Produce superfici C^2 ovunque eccetto che intorno ai vertici straordinari (C^1).
- È particolarmente semplice calcolare le tangenti (e quindi le normali) della superficie limite così pure la posizione limite dei punti di controllo.



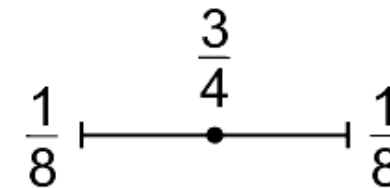
Schema Loop



Interior



Boundary / crease



Masks for odd vertices

Masks for even vertices

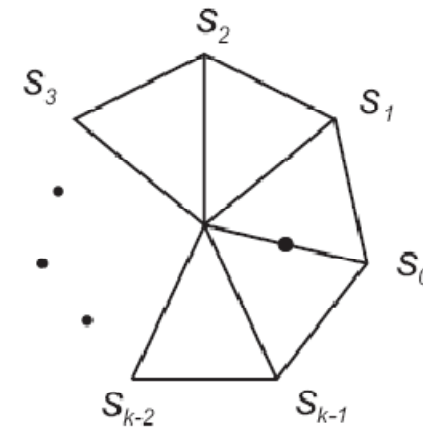
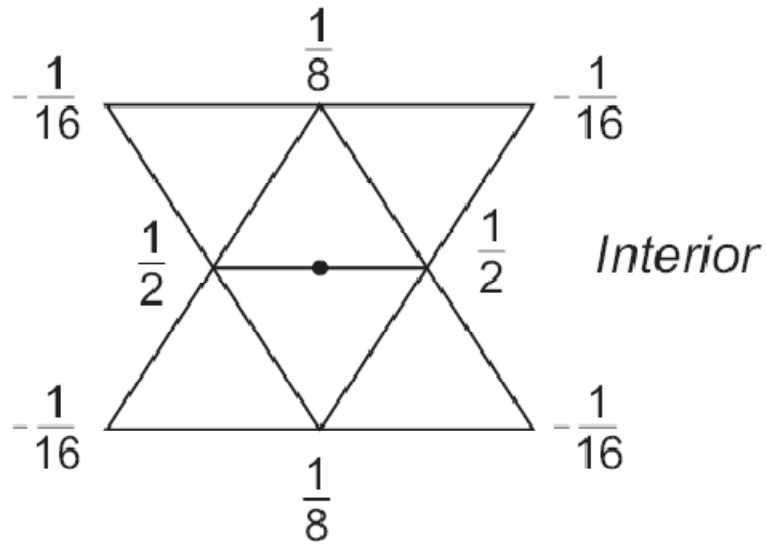


Schema Butterfly

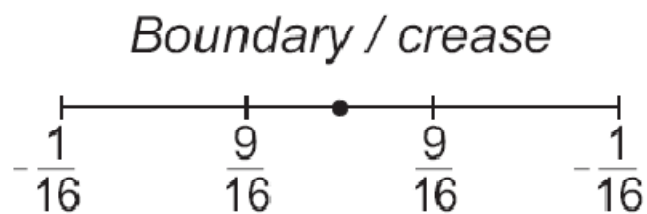
- Inizialmente proposto da Dyn et al. nel 1990 e successivamente modificato da Zorin et al. nel 1996 è uno schema interpolante per mesh triangolari.
- Produce superfici C^1 ovunque eccetto che nei vertici straordinari (la variante di Zorin garantisce continuità C^1 anche nei vertici straordinari).
- È possibile pre-calcolare le tangenti della superficie limite.
- E per quanto riguarda la posizione limite dei punti di controllo...? (indovinate)



Schema Butterfly (modificato)



*Mask for vertices adjacent
to an extraordinary vertex*

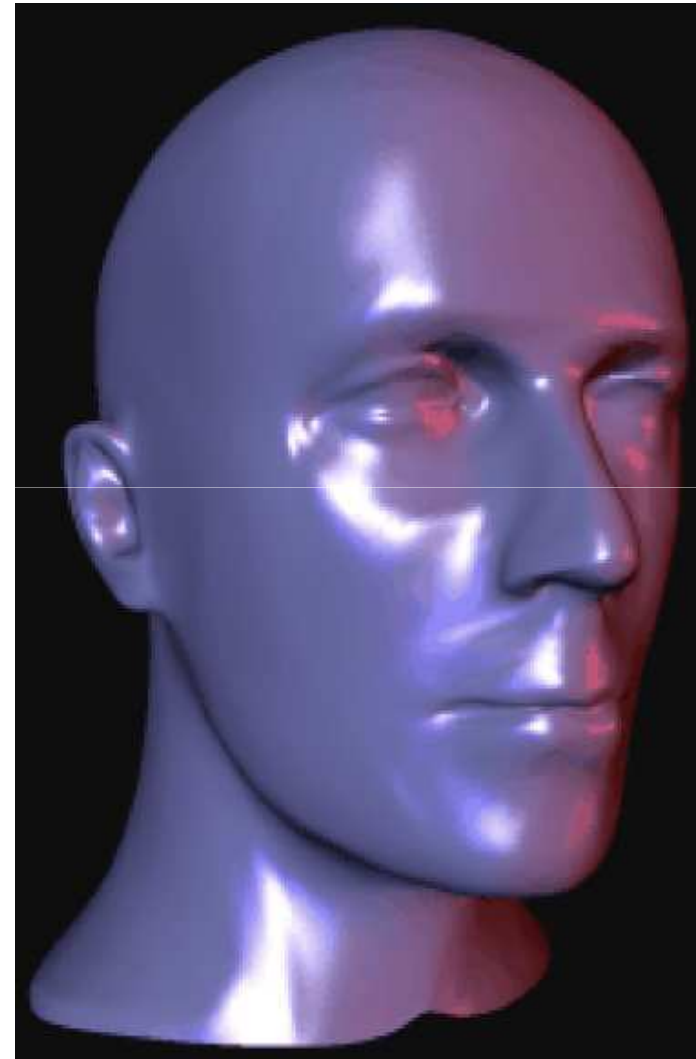


Masks for odd vertices

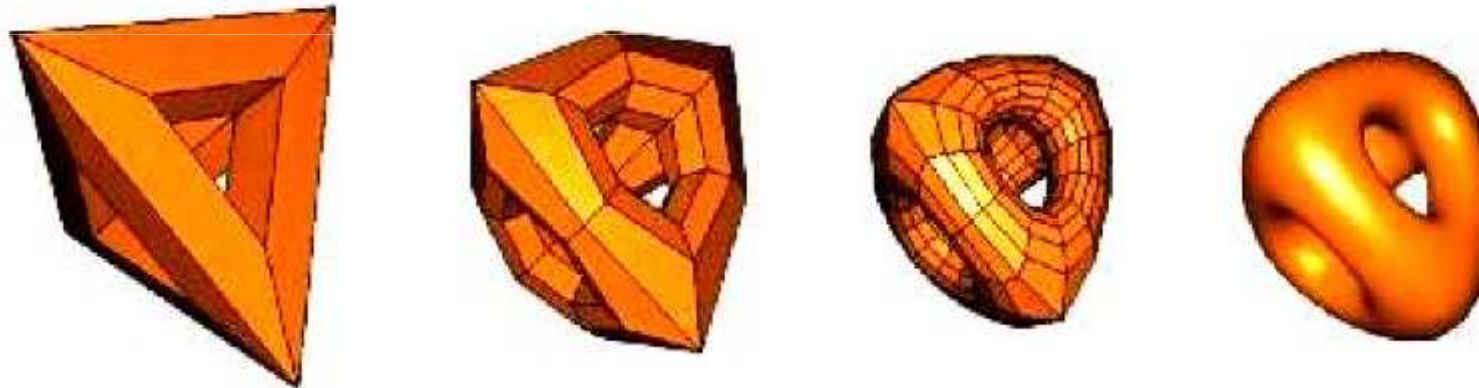


Esempio di Schema Butterfly

Facoltà di
Ingegneria

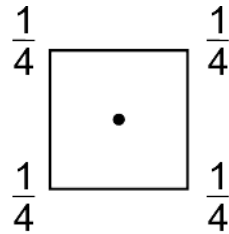


- Sviluppato da E. Catmull e J. Clark nel 1978. E' uno schema approssimante per mesh formate da quadrilateri.
- Produce superfici C^2 in ogni punto eccetto che nei vertici straordinari (C^1).

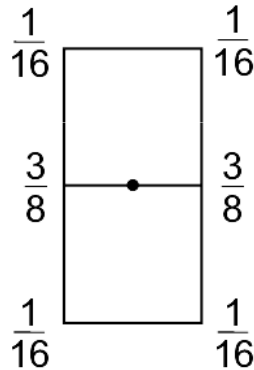




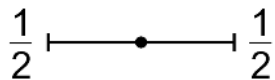
Schema Catmull-Clark



Mask for a face vertex

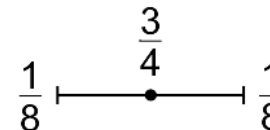


Mask for an edge vertex



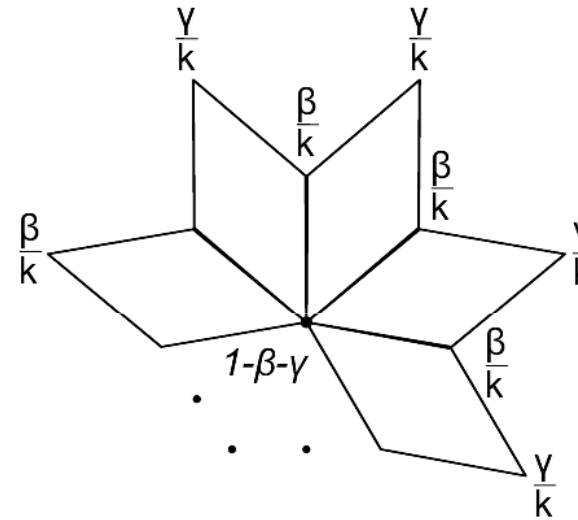
Masks for odd vertices

Boundary / crease



Masks for even vertices

Interior





Approfondimenti sulle mesh poligonali



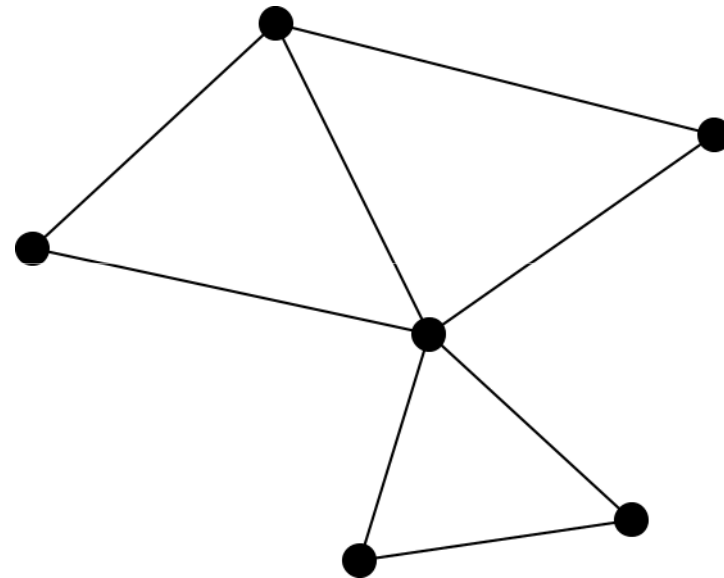
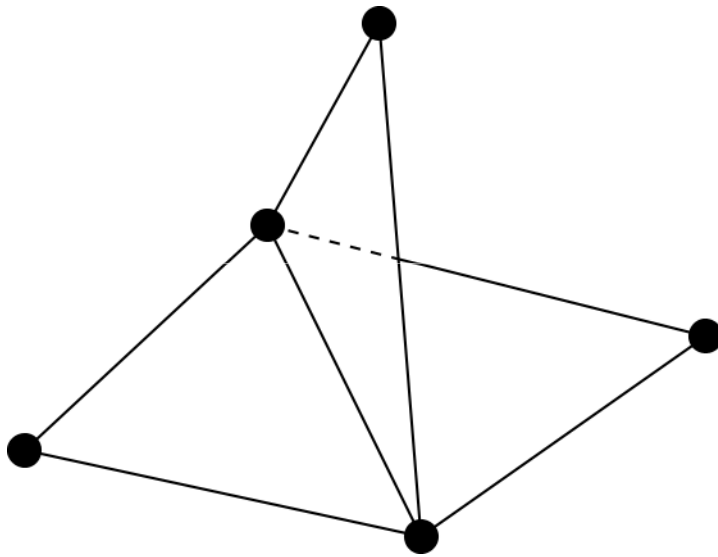
Mesh Two-manifold

- Una mesh si dice *two-manifold* se ogni suo punto è omeomorfo ad un disco.
- Gli algoritmi che lavorano su mesh assumono quasi sempre che la mesh sia two-manifold.
- Alcune caratteristiche di una mesh two-manifold:
 - Ogni lato ha due e solo due vertici incidenti
 - Su un lato incidono sempre una (lato di bordo) o due facce
 - Non devono essere presenti elementi sconnessi (esempio un vertice isolato)



Esempi non-manifold

Facoltà di
Ingegneria





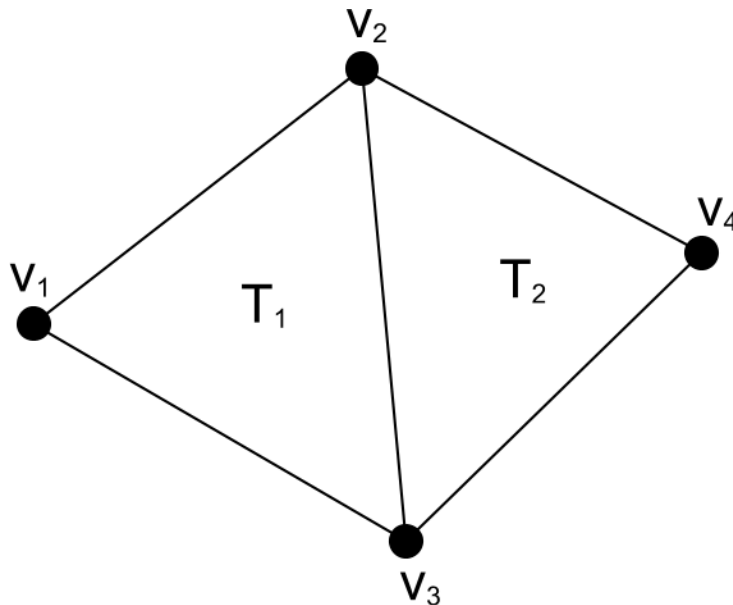
Query su mesh

- Molto spesso è utile effettuare delle query su mesh.
- Esempi:
 - Quali facce incidono su un certo vertice?
 - Quali facce incidono su un certo lato?
 - Quali vertici sono collegati con un dato vertice (1-ring)?
 - ...
- Nota: si dice che un elemento della mesh *incide* su un altro se la loro intersezione è non nulla.



Rappresentazione immediata

- Tutte le facce sono memorizzate come terne di vertici.
- Ad esempio il triangolo T si può rappresentare come $T = \{(V_{1x}, V_{1y}, V_{1z}), (V_{2x}, V_{2y}, V_{2z}), (V_{3x}, V_{3y}, V_{3z})\}$.
- E' semplice ma non è efficiente, ad esempio i vertici sono ripetuti.
- Le query sono particolarmente onerose.

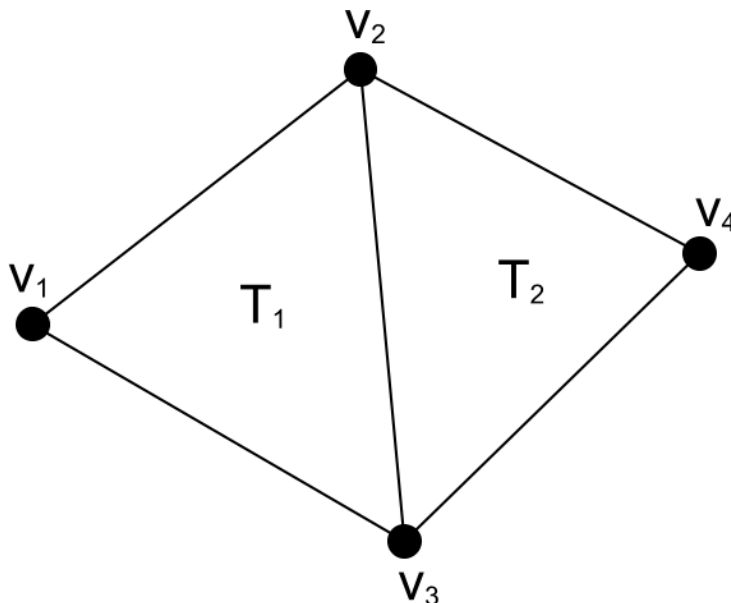


```
typedef struct  
{  
    float v1[3];  
    float v2[3];  
    float v3[3];  
} faccia;
```




Lista dei vertici

- Si utilizza una lista dei vertici senza ripetizione ed una lista delle facce che riferiscano la lista dei vertici.
- La faccia T riferisce (tramite puntatore o indice) ai vertici da cui è composta.
- In questo modo si elimina la duplicazione dei vertici ma non quella dei lati.
- Le query sono onerose anche in questo tipo di rappresentazione.



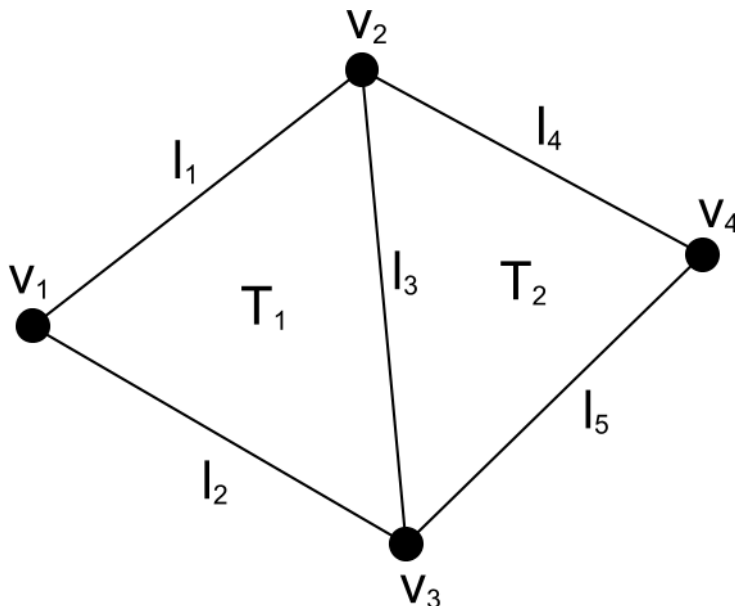
```
Typedef struct {  
    float x,y,z;  
} vertice;
```

```
Typedef struct {  
    vertice *v1, *v2, *v3;  
} faccia;
```



Lista dei lati

- Si utilizza una lista dei vertici (senza ripetizioni) ed una lista dei lati. Ogni lato riferisce i vertici che lo compongono. Le facce sono descritte riferendo i lati che le compongono.
- Vertici e lati non sono ripetuti.
- Le query sulla mesh cominciano a farsi più semplici ed efficienti.

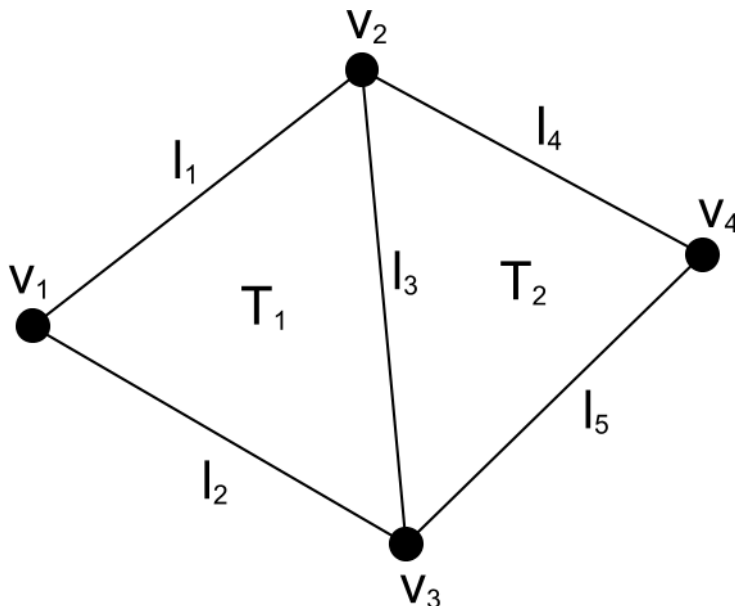


```
Typedef struct {  
    float x,y,z;  
} vertice;  
  
Typedef struct {  
    vertice *v1, *v2;  
} lato;  
  
Typedef struct {  
    lato *L1, *L2, *L3;  
} faccia;
```



Lista dei lati estesa

- Per rendere alcune query più efficienti si inserisce esplicitamente nella rappresentazione le due facce incidenti sul lato.
- Ovviamente a beneficiarne sono le query che sfruttano l'incidenza lato-faccia.



```
Typedef struct {  
    float x,y,z;  
} vertice;
```

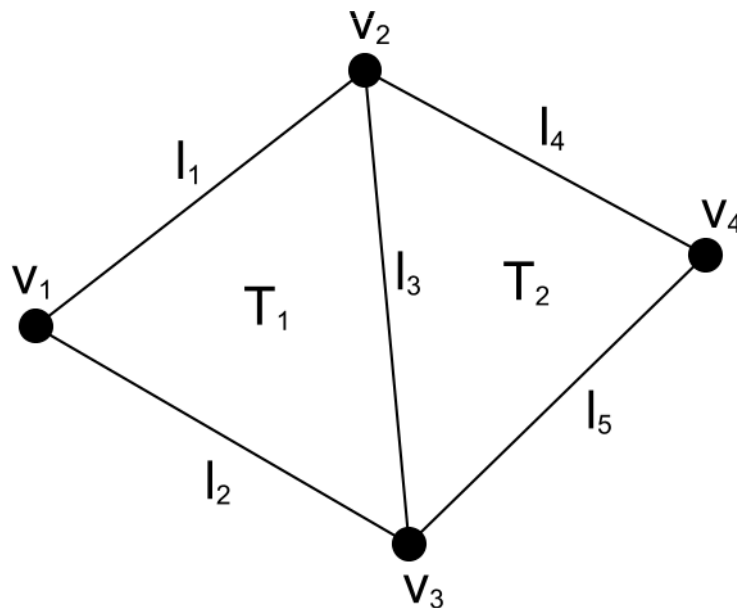
```
Typedef struct {  
    vertice *v1, *v2;  
    faccia *f1, *f2;  
} lato;
```

```
Typedef struct {  
    vertice *L1, *L2, *L3;  
} faccia;
```



Winged-Edge

- I dati di incidenza vanno a far parte ancora più pesantemente della rappresentazione.
- Ogni lato contiene, oltre ai vertici, due puntatori alle facce adiacenti più i puntatori ad i lati uscenti.
- Ogni vertice contiene un puntatore ad uno dei lati che incide su di esso e le sue coordinate.
- Una faccia è definita da un puntatore ad uno dei lati che vi incide.



```
typedef struct {
    float x,y,z;
    lato_we *lato;
} vertice_we;

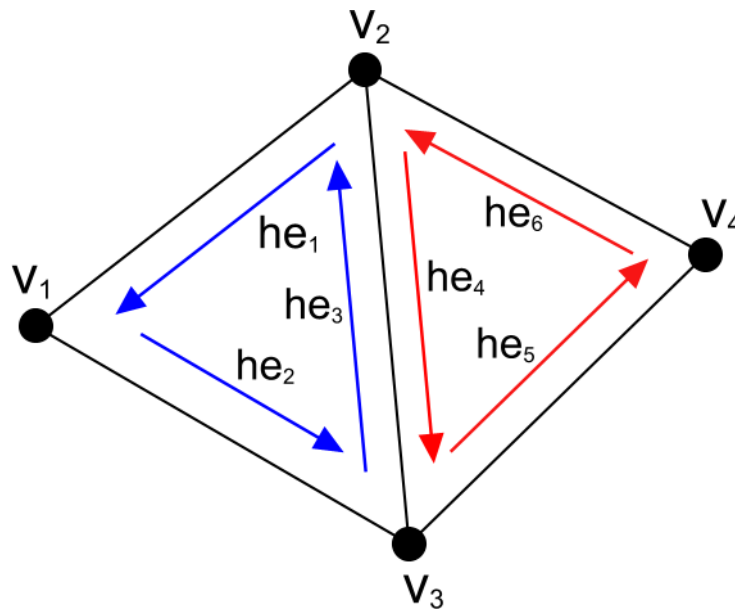
typedef struct {
    vertice_we *v1, *v2;
    lato_we *l1sin, *l1des;
    lato_we *l2sin, *l2des;
    faccia_we *f1, *f2;
} lato_we;

typedef struct {
    lato_we *lato;
} faccia_we;
```



Half-Edge

- In questa rappresentazione ogni lato viene diviso in due *semi-lati* orientati in modo opposto (da cui il nome).
- Ciascun *half-edge* contiene un puntatore al vertice iniziale, al mezzo lato gemello (secondo un ordinamento dato) ed al mezzo-lato associato.
- Ogni vertice contiene un puntatore ad uno qualsiasi dei mezzi lati uscenti e le coordinate.
- Ogni faccia contiene uno dei suoi mezzi lati.



```
Typedef struct {  
    float x,y,z;  
    lato_he *lato;  
} vertice_he;
```

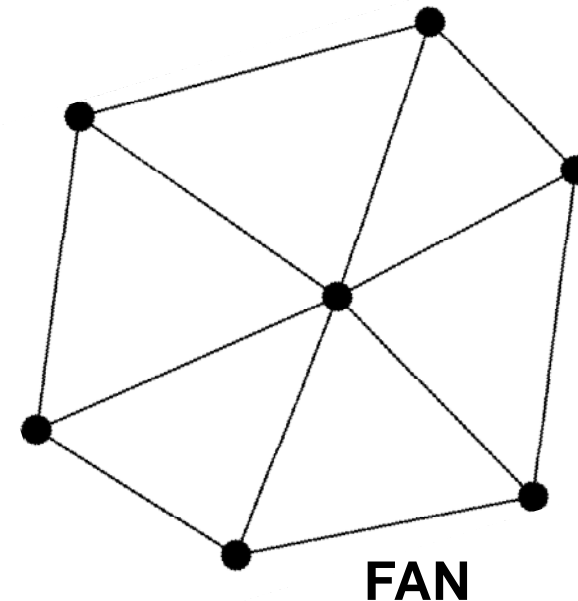
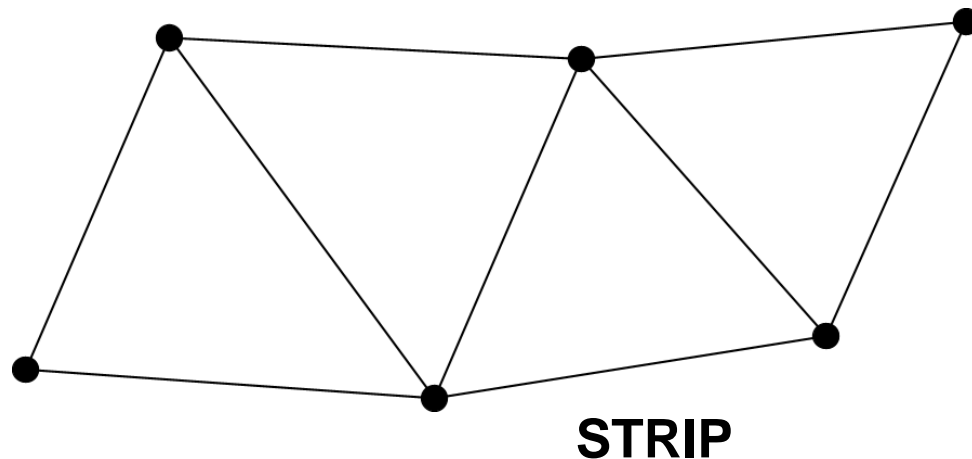
```
Typedef struct {  
    vertice_he *origine;  
    lato_he *gemello;  
    lato_he *successivo;  
    faccia_he *faccia;  
} lato_he;
```

```
Typedef struct {  
    lato_he *lato;  
} faccia_he;
```



Fan e Strip di Triangoli

- Sono particolari gruppi di triangoli utili per ottimizzare le performances di rendering (vedremo).
- *Fan*: è un gruppo di triangoli con un vertice in comune.
- *Strip*: è un gruppo di triangoli con un lato in comune.





Domande?