Modelllazione geometrica

Dove si delineano le principali tecniche per descrivere e rappresentare oggetti in uno spazio tridimensionale.

- **Rappresentazione poligonale**
- **Curve e superfici**
- **Geometria Costruttiva Solida**
- **Partizionamento spaziale**

- Ora che abbiamo a disposizione gli strumenti matematici, geometrici ed algoritmici necessari, iniziamo a occuparci con questo capitolo di grafica 3D vera e propria, cominciando da come si descrive un oggetto in uno spazio 3D
- La **modellazione**, sia manuale che procedurale, di oggetti 3D è fondamentale nell'ambito della grafica al calcolatore, come fondamentali sono i modi in cui un modello può essere rappresentato
- In questa prima parte vedremo i fondamenti, nel prossimo capitolo ci addentreremo in alcuni aspetti avanzati della problematica
- Una volta che sapremo come descrivere un oggetto 3D, affronteremo lo studio del **rendering** per capire come passare dalle descrizioni descritte in questo capitolo all'immagine finale

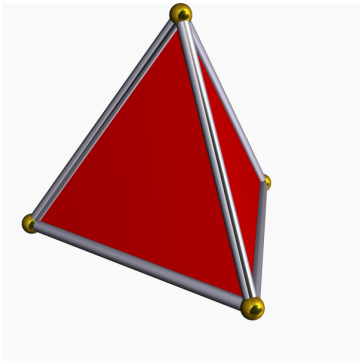
- Gli oggetti che si vogliono rappresentare in una applicazione grafica hanno di solito caratteristiche particolari
  1. Sono **finiti**
  2. Sono **chiusi** (non sempre)
  3. Sono **continui**
- Le rappresentazioni di oggetti (regioni dello spazio, in generale) si suddividono in
  - basate sul contorno (boundary): descrivono una regione in termini della superficie che la delimita (**boundary representation**, o **b-rep**).
  - basate sullo spazio occupato (o volumetriche).
- Inizieremo dalle rappresentazioni basate sul contorno, ed in particolare da quella poligonale.

Rappresentazione poligonale

- Per semplificare la descrizione ai fini della grafica 3D al calcolatore si usa spesso una **approssimazione poligonale** degli oggetti (del loro contorno).
- Si tratta di approssimare una superficie bidimensionale con un insieme di poligoni convessi opportunamente connessi gli uni agli altri.

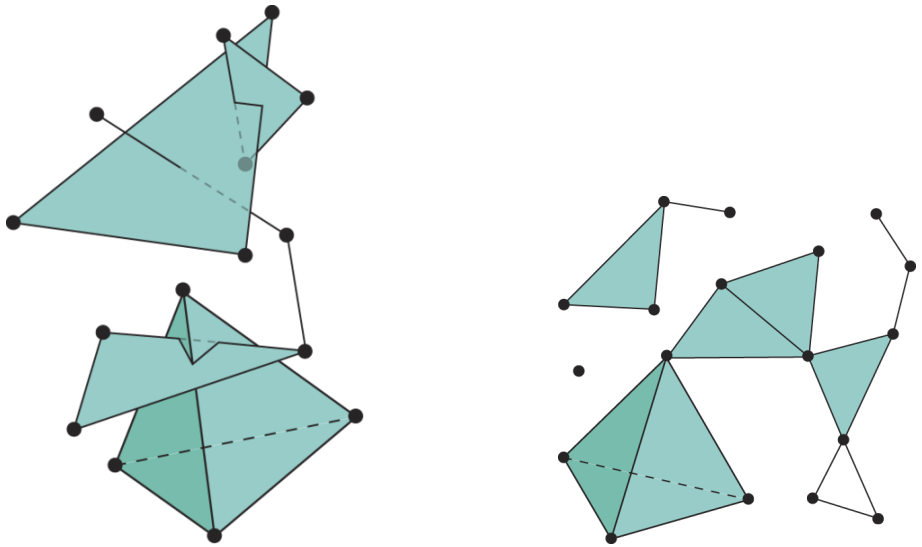
Complesso simpliciale

- Un **simplesso** è l'analogo n-dimensionale del triangolo.
- Specificamente, un simplesso di ordine n (o n-simplesso) è il guscio convesso di $n+1$ punti affinemente indipendenti in $\mathbb{R}^d$ $(d \geq n)$.
- Per esempio,
  uno 0-simplesso è un punto, un 1-simplesso è un segmento,
  un 2-simplesso è un triangolo, un 3-simplesso è un tetraedro.



- Il guscio convesso di un qualunque sottoinsieme degli $n+1$ punti punti che definiscono il n-simplesso si chiama **faccia** del simplesso. Le facce sono a loro volta simplessi (di ordine $\leq n$). Se il sottoinsieme è proprio, anche la faccia si dice **propria**.
- Le facce di ordine 0 sono i punti stessi, chiamati **vertici**. Le facce di ordine 1 si chiamano **spigoli**. La faccia di ordine n è il n-simplesso stesso.

- Un **complesso simpliciale** $\mathcal{K}$ è un insieme di simplessi che soddisfano le seguenti condizioni:
  1. Ogni faccia di un simplesso in $\mathcal{K}$ appartiene a sua volta a $\mathcal{K}$.
  2. L'intersezione di due simplessi σ_1 e σ_2 è una faccia comune a σ_1 e σ_2 oppure è vuota.
- Se l'ordine massimo dei simplessi è k, $\mathcal{K}$ prende il nome di k-complesso simpliciale.
- Per esempio, un 2-complesso simpliciale deve contenere almeno un triangolo e nessun tetraedro.
- Un k-complesso simpliciale è **puro** se ogni simplesso di ordine $< k$ è la faccia di un k-simplesso.



- Per esempio, un 2-complesso simpliciale puro è fatto solo di triangoli (non ci sono vertici o spigoli "orfani").

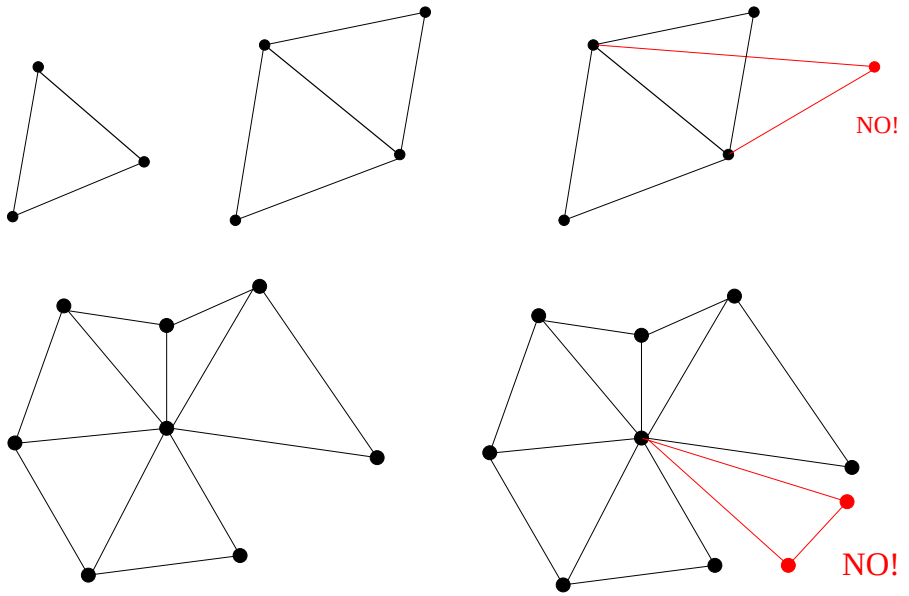
- Due simplessi σ_1 e σ_2 sono **incidenti** se σ_1 è una faccia propria di σ_2 o vale il viceversa.
- Due k-simplessi sono $(k-1)$**-adiacenti** se esiste un $(k-1)$-simplesso che è una faccia propria di entrambi.

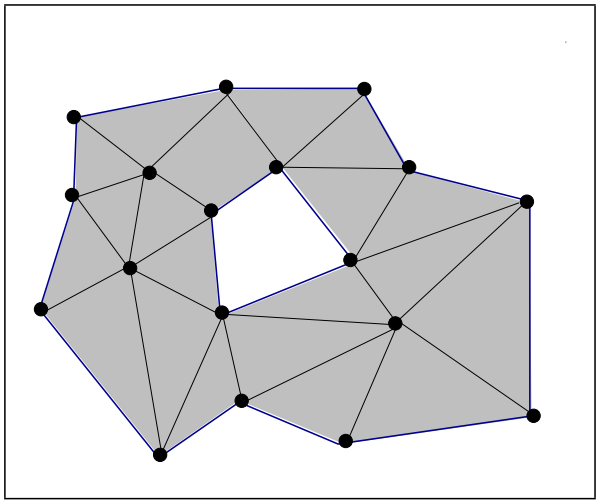
Varietà

- Una **varietà k-dimensionale** $\mathcal{X}$ è un sottoinsieme di $\mathbb{R}^d$ in cui ogni punto ha un intorno omeomorfo alla sfera aperta di $\mathbb{R}^k$.
- Per esempio, la superficie della sfera, o di un poliedro, è una varietà bidimensionale. In generale le superifici degli oggetti solidi sono varietà bidimensionali.
- **Omeomorfismo**: applicazione biiettiva, continua, con inversa continua. Intuizione: trasformazione senza "strappi".
- Una varietà k-dimensionale con **bordo** è una varietà in cui ogni punto ha un intorno omeomorfo alla sfera aperta o alla semisfera aperta di $\mathbb{R}^k$.
- Il **bordo** di $\mathcal{X}$ è l'insieme dei punti che hanno un intorno omeomorfo alla semisfera aperta.
- Una varietà è sempre una varietà con bordo, eventualmente vuoto.
- Il bordo, se non è vuoto, è a sua volta una varietà k-1 dimensionale senza bordo.

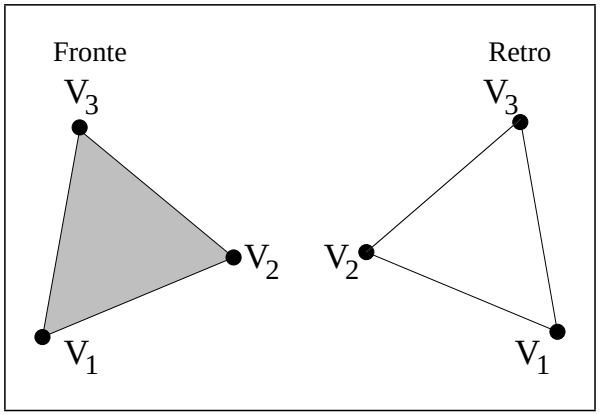
Maglie poligonali

- Una **maglia** (*mesh*) triangolare è 2-complesso simpliciale puro che è anche una varietà bidimensionale con bordo.
- I triangoli della maglia si chiamano anche **facce**.
- La condizione di essere varietà si traduce nei seguenti vincoli ulteriori sulla struttura del complesso simpliciale:
  - uno spigolo appartiene al massimo a due triangoli (quelli eventuali che appartengono ad uno solo formano il **bordo** della maglia)
  - se due triangoli incidono sullo stesso vertice allora devono appartenere alla chiusura transitiva della relazione di 1-adiacenza, ovvero devono formare un ventaglio o un ombrello.



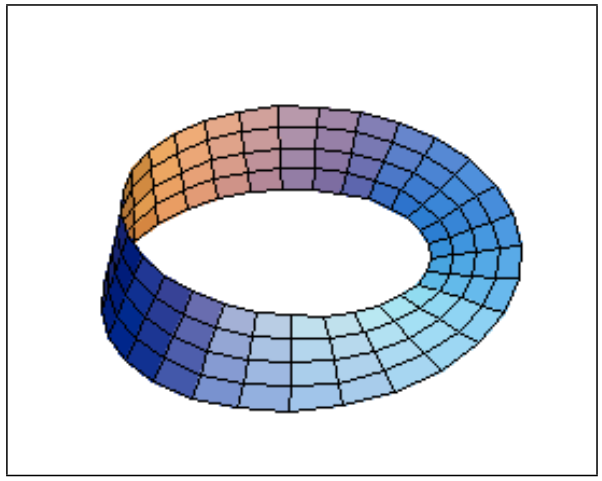


- Il bordo della maglia consiste di uno o più anelli (sequenza chiusa di spigoli) o *loop*.
- Se non esistono spigoli di bordo la maglia è **chiusa** (come quelle che rappresentano la superficie di una sfera).



- L'**orientazione** di una faccia è data dall'ordine ciclico (orario o antiorario) dei suoi vertici incidenti. L'orientazione determina il fronte ed il retro della faccia. La convenzione (usata anche da OpenGL) è che la faccia mostra il fronte se i vertici sono disposti in senso antiorario.

- L'orientazione di due facce adiacenti è **compatibile** se i due vertici del loro spigolo in comune sono in ordine inverso. Vuol dire che l'orientazione non cambia attraversando lo spigolo in comune.
- La maglia si dice **orientabile** se esiste una scelta dell'orientazione delle facce che rende compatibili tutte le coppie di facce adiacenti.



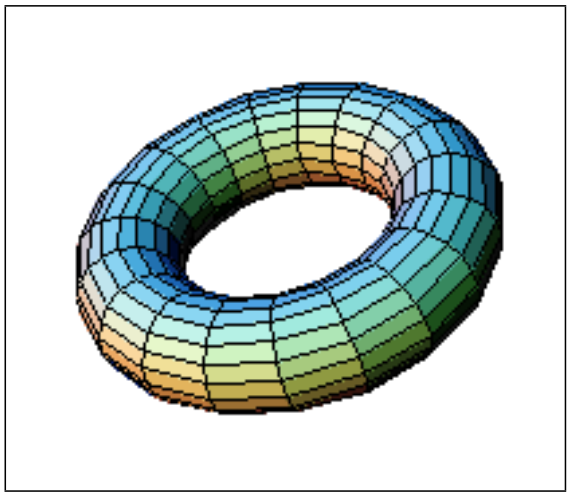
- **Maglie poligonali** Abbiamo definito la maglia triangolare. In maniera analoga si può estendere la definizione a maglie poligonali, ovvero composte da poligoni qualunque (poco usate comunque).
- Si noti comunque che un qualuque poligono può sempre essere scomposto in triangoli, tracciando le sue diagonali.

Equazione di Eulero

- Se V è il numero di vertici, L il numero di spigoli ed F il numero di facce della maglia poligonale orientabile *chiusa* di genere G, allora vale la **Formula di Eulero**

$$V - L + F = 2 - 2G$$

- Una superficie ha **genere** G se può essere tagliata lungo $2G$ anelli chiusi senza disconnetterla
- Il genere di una superficie dipende (e determina) la sua topologia; per una sfera, per esempio, $G = 0$, mentre per un toro (una ciambella) $G = 1$.
- In ogni caso la maggior parte degli oggetti utilizzati in computer graphics sono poliedri semplici, topologicamente equivalenti a sfere.



- Più in generale, per una maglia poligonale orientabile (e varietà bidimensionale) vale la formula:

$$V - L + F = 2(S - G) - B$$

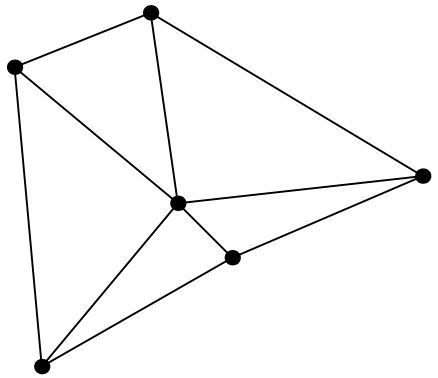
dove S è il numero di componenti connesse e B è il numero di anelli di bordo.

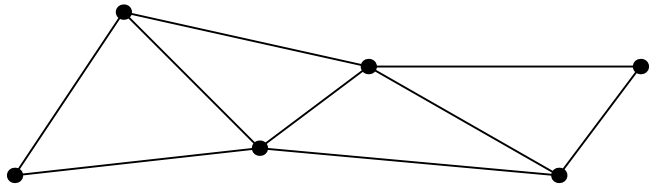
Tipologie di maglia poligonale

- Le più comuni tipologie di maglia poligonale sono:
  - **Generiche:** i poligoni possono avere qualsiasi numero di spigoli e non è detto che ci sia un solo tipo di poligono. Sono raramente utilizzate in grafica al calcolatore
  - **Quadrangolari:** gli elementi poligonali sono **tutti** quadrilateri. Sono alle volte usate quando i dati da rappresentare posseggono una simmetria spaziale particolare, ad esempio se si vuole fare il rendering di un terreno descritto da un array di altezze. In una maglia quadrangolare bisogna imporre un vincolo aggiuntivo di planarità per ogni quadrilatero che la compone.
  - **Triangolari:** sono le più diffuse. Questo deriva dal fatto che, tre punti giacciono sempre su un piano, dunque non servono vincoli aggiuntivi sulla planarità dei triangoli.
- OpenGL consente di descrivere maglie poligonali generiche, ma per disegnarle li suddivide in triangoli.

Fan e strip di triangoli

- Quando si devono disegnare due triangoli che hanno uno spigolo in comune, tale spigolo in realtà viene disegnato due volte. Questo introduce un certo grado di **overdraw**, ovvero di ridondanza che può incidere sulle prestazioni dell'applicazioni.
- Si preferisce quindi raggruppare i triangoli di una maglia in opportuni gruppi che possono essere elaborati in maniere più efficiente. Per le OpenGL si hanno due possibili gruppi di triangoli:
  1. **Fan di triangoli:** è un gruppo di triangoli che hanno in comune un vertice. Il primo triangolo viene specificato completamente, per i successivi basta dare il nuovo vertice Questo raggruppamento è piuttosto efficiente, ma in genere i triangoli che incidono su un vertice sono pochi
  2. **Strip di triangoli:** è un gruppo di triangoli che posseggono a due a due uno spigolo in comune. Di nuovo il primo triangolo viene specificato normalmente, per i successivi basta specificare il nuovo vertice. È meno efficiente, ma le strip in genere contengono più triangoli delle fan





Trasformazioni affini e proiezioni

- Per determinare l'effetto di una qualsiasi trasformazione affine su un oggetto (traslazione, rotazione, scalatura, composizioni varie di queste), **basta applicare la trasformazione ai vertici** (che sono punti); le informazioni connettive date dagli spigoli non cambiano in questo tipo di trasformazioni
- Questo rende piuttosto semplice il rendering di oggetti descritti in termini di maglie poligonali .. qualsiasi trasformazione viene eseguita sui vertici, cioè si tratta di applicare trasformazioni affini su punti, cosa che sappiamo bene come fare
- L'affermazione precedente è vera anche per la proiezione; per vedere come si proietta la forma di una maglia su un piano (l'immagine), basta seguire la proiezione dei vertici.

Alcuni esempi di maglie poligonali

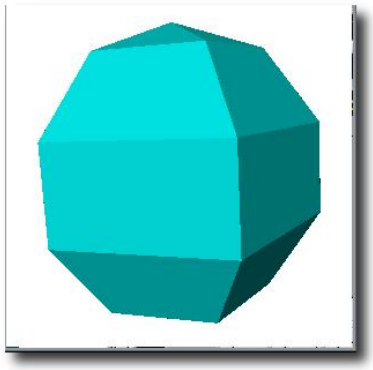
Solidi platonici (alcuni)

- Un **poliedro** è un solido la cui superficie è formata da un insieme finito di facce poligonali. Si dice **semplice** se non ha buchi. Più precisamente, la superficie di un poliedro è una maglia poligonale orientabile chiusa di genere 0.
- Un poliedro semplice è **regolare** se le facce sono poligoni regolari congruenti (stesso numero degli spigoli) ed in ogni vertice incide lo stesso numero di spigoli.
- I **solidi platonici** sono tutti e soli i poliedri **regolari** in tre dimensioni: tetraedro, ottaedro, icosaedro, esaedro (o cubo), penta-dodecaedro (dodici facce pentagonali).

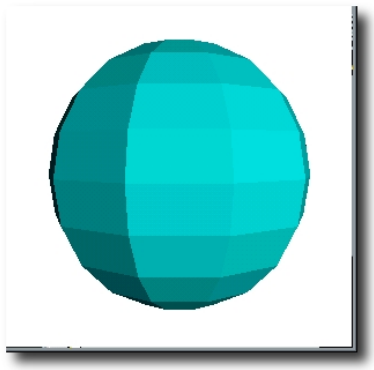
(1) Tetraedro (v=4, f=4) (2) Ottaedro (v=6, f=8) (3) Icosaedro (v=12, f=20)

Varie approssimazioni di una sfera

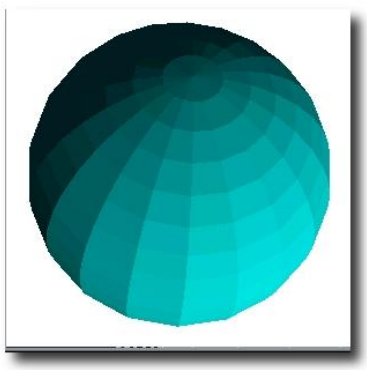
Approssimazioni di una sfera con poliedri; si vede come più il numero di vertici è alto più l'approssimazione risulta fine.



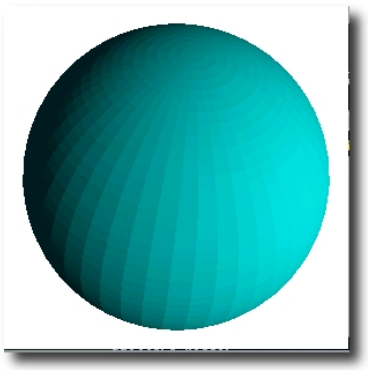
(4) (v=36, f=50)



(5) (v=121, f=200)

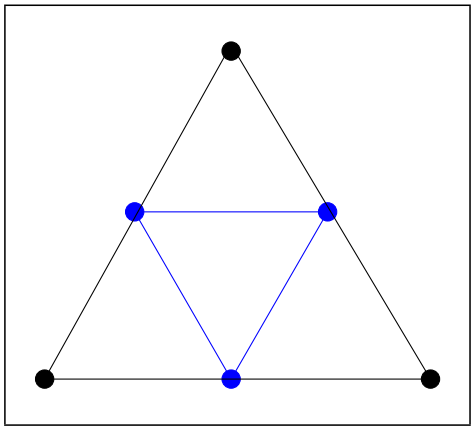


(6) (v=441, f=800)



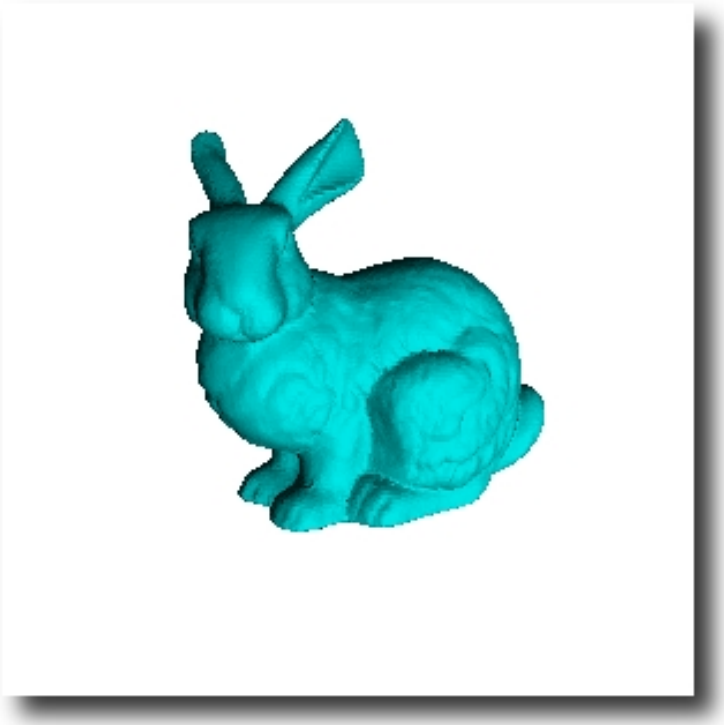
(7) (v=2601, f=5000)

Come si costruisce? Con un procedimento iterativo: si parte dal tetraedro inscritto nella sfera e si suddivide ciascuna faccia in 4 triangoli bisecando gli spigoli. I nuovi vertici vengono spostati sulla superficie della sfera e si procede.



Esempi complessi

Il coniglio di **Stanford** è ottenuto dalla **fusione** di varie immagini di profondità ottenute con apparecchiature laser avanzate. L'originale è fatto di terracotta. Il caccia dell'Alleanza Ribelle deriva da modellazione manuale.

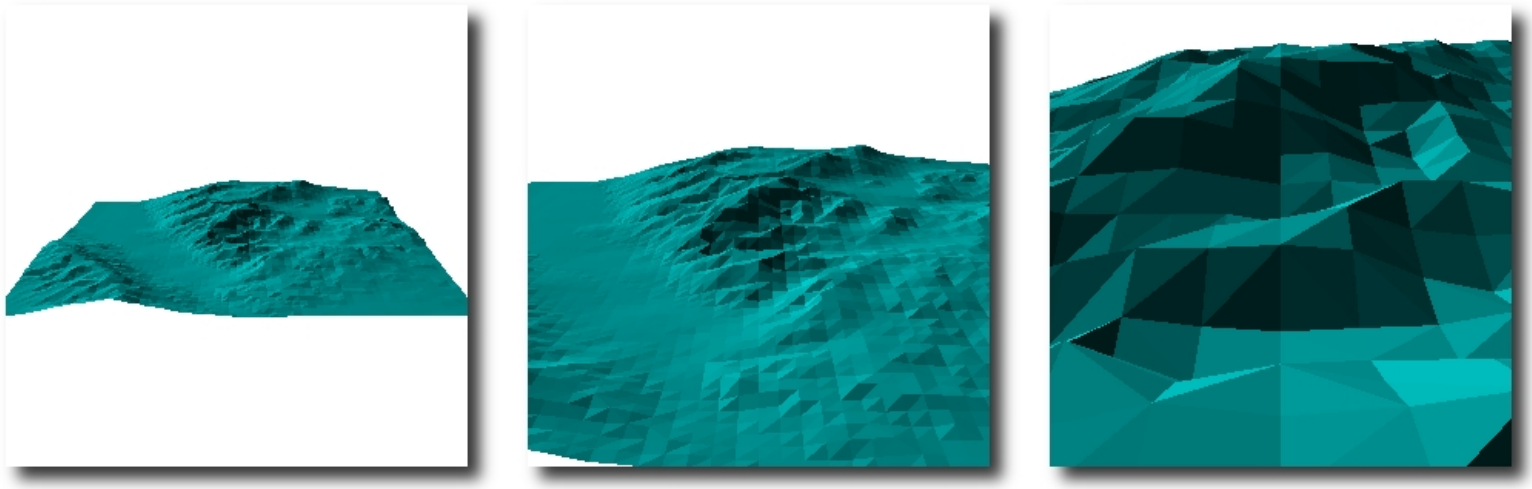


(8) Bunny (v=35947, f=69451)

(9) X-Wing (v=3104, f=6084)

Heightfield

In questa sequenza si può vedere un'approssimazione poligonale per le altezze di un terreno.



(11) Terreno (v=3721, f=7200)

Rappresentazione

- Vediamo ora quali strutture dati si usano per descrivere nella pratica delle maglie poligonali
- Da notare, prima di tutto, che nella stessa applicazione grafica si utilizzano spesso più strutture dati diverse per rappresentare lo stesso oggetto
- Vedremo in un attimo, ad esempio, come sia comodo descrivere una maglia poligonale in un certo modo in un file esterno, ma sia più efficiente usare una descrizione diversa all'interno del programma
- Sono quindi necessarie delle procedure per convertire una rappresentazione in un'altra
- Progettare ed implementare tali procedure è un ottimo modo per capire nel dettaglio le varie rappresentazioni utilizzate per descrivere maglie poligonali
- Nei disegni e negli esempi ci concentreremo sul caso di maglia triangolare, ma il discorso è valido in generale per tutti i poligoni convessi

Elementi fondamentali

- Gli elementi fondamentali di una rappresentazione poligonale (a maglia) di una superficie bidimensionale sono
  1. **Vertici:** sono gli elementi 0 dimensionali e sono identificabili con punti nello spazio $3D$ (essenzialmente tre coordinate); alle volte può essere utile associare ai vertici altre caratteristiche oltre alla posizione (tipo il colore)
  2. **Spigoli:** sono elementi 1 dimensionali e rappresentano un segmento che unisce due vertici. Di solito non contengono altre informazioni.
  3. **Facce:** sono i poligoni bidimensionali, formati da un certo numero di spigoli e di vertici (dimostrare che sono in numero uguale). I vertici o gli spigoli si usano per identificare la faccia; possono contenere altre informazioni (tipo il colore)
  4. **Normali:** è fondamentale sapere quale è l'esterno della superficie e quale l'interno; a tal scopo si associa spesso ad una maglia poligonale anche l'informazione sulla normale uscente. Come vedremo questa informazione può essere associata alle facce, come sarebbe naturale, oppure ai ai vertici (per ragioni che saranno chiare nel seguito).

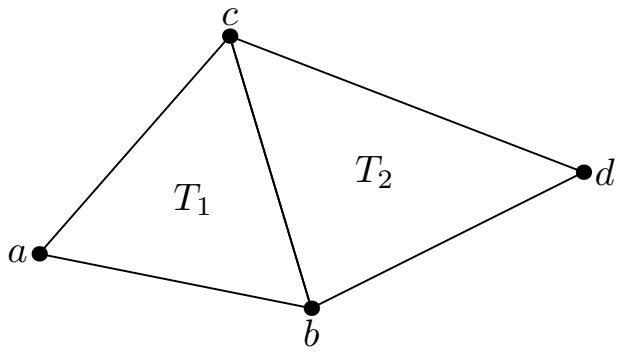
- Da tali definizioni si vede che esistono due elementi importanti; i **vertici** danno informazioni di tipo **posizionale**, gli **spigoli** informazioni di tipo **connettivo**
- Ovvero, i vertici sono gli unici elementi a cui è associata una posizione; la disposizione dei vertici determina la forma dell'oggetto
- Gli spigoli connettono i vertici, permettendo di introdurre un concetto di "vicinanza" tra vertici e dando le informazioni di tipo topologico (ovvero definiscono un **grafo**)
- Le facce sono determinate una volta dati i vertici e gli spigoli, quindi non introducono nulla di nuovo dal punto di vista posizionale e topologico; al più possono avere associate informazioni utili per lo shading, ma come vedremo spesso non è il caso
- La normale $\mathbf{n}$ ad una faccia è data dal prodotto vettore di due suoi spigoli consecutivi non collineari (bisogna stare attenti al verso: la normale è *uscente* dal *fronte* della faccia). Per un triangolo (V_1, V_2, V_3) si ha:

$$\mathbf{n} = (V_3 - V_2) \times (V_1 - V_2)$$

- Una **ricerca di incidenza** è una procedura che determina tutti gli elementi di un dato tipo che incidono su un certo elemento
- Ad esempio può essere interessante e utile sapere, data una faccia, quali siano i vertici della maglia che incidono su tale faccia.

Rappresentazione semplice

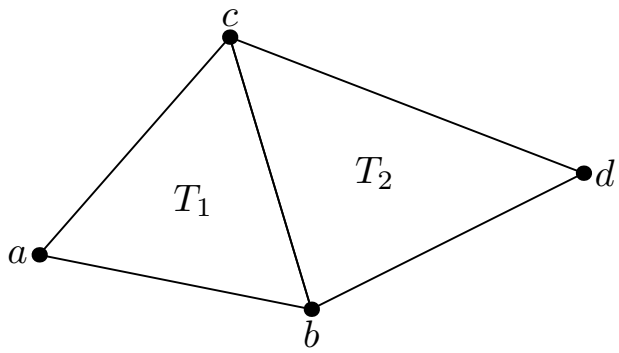
- Il modo più immediato per rappresentare una maglia poligonale è specificare tutte le facce della maglia come terne di triplette di coordinate cartesiane
- Ad esempio il triangolo T_1 si può rappresentare come $T_1 = \{(a_x, a_y, a_z), (b_x, b_y, b_z), (c_x, c_y, c_z)\}$.
- È chiaro che sebbene la rappresentazione sia semplice e compatta, non è efficiente; ad esempio i vertici vengono ripetuti nella lista dei poligoni.
- Ricerche di incidenza sono inoltre particolarmente onerose (e a volte non definibili)




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

Lista dei vertici

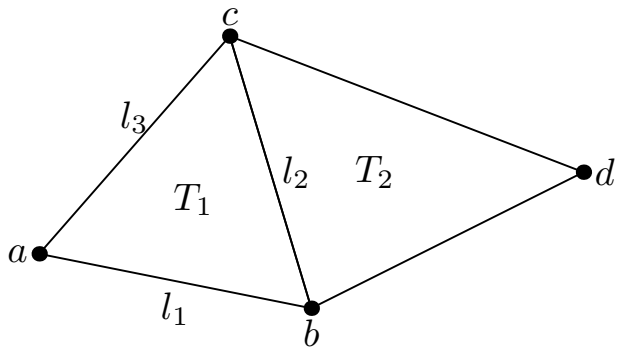
- Un primo miglioramento può essere ottenuto introducendo esplicitamente i vertici
- Questo può essere fatto costruendo una lista dei vertici (senza ripetizioni) e dando quindi una lista di facce, ciascuna descritta dai puntatori ai vertici che la compongono (in genere in senso antiorario)
- Ad esempio la faccia $T1$ punta ai tre vertici a, b e c
- In questo modo si elimina la duplicazione dei vertici, ma non quello sugli spigoli; uno spigolo appartenente a due triangoli viene immagazzinato due volte
- Le ricerche di incidenza continuano ad essere complesse




```
typedef struct {
   float x,y,z;
} vertice;
typedef struct {
   vertice* v1,v2,v3;
} faccia;
```

Lista degli spigoli

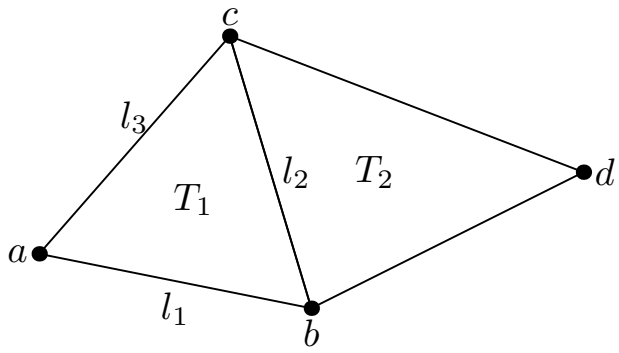
- Un secondo passo può essere ottenuto introducendo esplicitamente gli spigoli
- Questo può essere fatto costruendo una lista dei vertici (senza ripetizioni) ed una lista degli spigoli, ciascuno composto dai due puntatori ai vertici incidenti sullo spigolo; ciascuna faccia viene descritta infine dai puntatori degli spigoli che la compongono (in genere in senso antiorario)
- Ad esempio lo spigolo l_1 punta ai due vertici a e b, mentre la faccia T_1 punta ai due spigoli l_1, l_2 ed l_3.
- In questo modo si elimina la ripetizione sui vertici e sugli spigoli
- Le ricerche di incidenza sono più semplici, ma non tutte




```
typedef struct {
   float x,y,z;
} vertice;
typedef struct {
   vertice* in, fin;
} spigolo;
typedef struct {
   spigolo* l_1,l_2,l_3;
} faccia;
```

Lista degli spigoli estesa

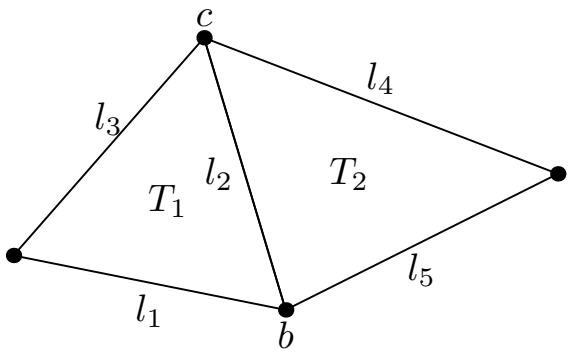
- Un terzo passo può essere ottenuto aggiungendo alla struttura dati degli spigoli anche i due puntatori alle facce incidenti sullo spigolo
- Ad esempio lo spigolo l_2 punta a T_1 e T_2, oltre che a b e c come prima.
- In questo modo si semplificano di molto le ricerche di incidenza spigolo-faccia




```
typedef struct {
   float x,y,z;
} vertice;
typedef struct {
   vertice* in, fin;
   faccia* sin, dest;
} spigolo;
typedef struct {
   spigolo* a,b,c;
} faccia;
```

Rappresentazione Winged-Edge

- In tale rappresentazione (Baugmart 1975) si aggiungono dei puntatori allo spigolo per rendere più semplice l'analisi delle incidenze.
- L'elemento base è lo spigolo (edge) con le sue due facce incidenti (wings)
- Lo spigolo l_2 contiene un puntatore ad i due vertici su cui incide b, c, alle due facce su cui incide T_1, T_2 ed ai due spigoli uscenti da ciascun vertice che contornano le due facce l_1, l_5, l_3, l_4
- Un vertice contiene un puntatore ad uno degli spigoli che incide su di esso, più le coordinate (ed altro)
- La faccia contiene un puntatore ad uno degli spigoli che vi incide (ed altro).



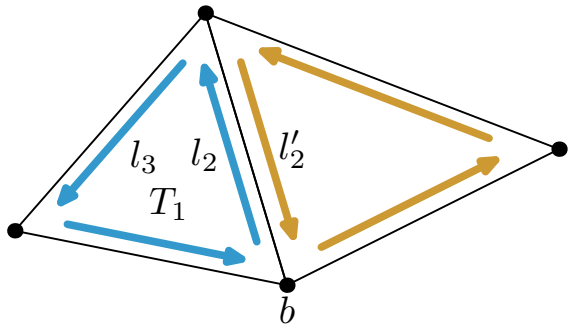

```
typedef struct {
   we_vertice* v_ini, v_fin;
   we_spigolo* vi_sin, vi_dstr;
   we_spigolo* vf_sin, vf_dstr;
   we_faccia* f_sin, f_dstr;
} we_spigolo;
typedef struct {
   float x, y, z;
   we_spigolo* spigolo;
} we_vertice;
typedef struct {
   we_spigolo* spigolo;
} we_faccia;
```

Esercizio 1: descrivere una procedura che, data una rappresentazione winged-edge di una maglia triangolare, "stampi" tutti le facce incidenti su un dato vertice v, assumendo data una procedura `Stampa_Faccia(we_faccia *f)`

```
l = v->spigolo;

do{
   f = l->f_sin;
   Stampa_Faccia(f);
   l = l->vi_sin;
} while (l != v->spigolo)
```

Rappresentazione Half-Edge

- In tale rappresentazione ogni spigolo viene diviso in due spigoli orientati in modo opposto
- Ciascun mezzo spigolo contiene un puntatore al vertice iniziale, alla faccia a cui "appartiene", al mezzo spigolo successivo (seguendo l'ordinamento) ed al mezzo spigolo gemello
- Ogni vertice, oltre alle coordinate (e possibilmente altre caratteristiche come il colore) contiene un puntatore ad uno qualsiasi dei mezzi spigoli che esce da tale vertice
- Ogni faccia contiene uno dei suoi mezzi spigoli (oltre ad altre caratteristiche quali, ad esempio, la normale)
- Per esempio, seguendo il disegno, si hanno i seguenti puntatori: $l_2(b, l_2', T_1, l_3)$, $T_1(l_2)$, $b(l_2)$.




```
typedef struct {
   he_vertice* origine;
   he_spigolo* gemello;
   he_faccia* faccia;
   he_spigolo* successivo;
} he_spigolo;
typedef struct {
   float x, y, z;
   he_spigolo* spigolo;
} he_vertice;
typedef struct {
   he_spigolo* spigolo;
} he_faccia;
```

Esercizio 2: descrivere una procedura che, data una rappresentazione half-edge di una maglia triangolare, "stampi" tutti i mezzi-spigoli uscenti dal vertice v con una procedura data `Stampa_Spigolo(he_spigolo *l)`

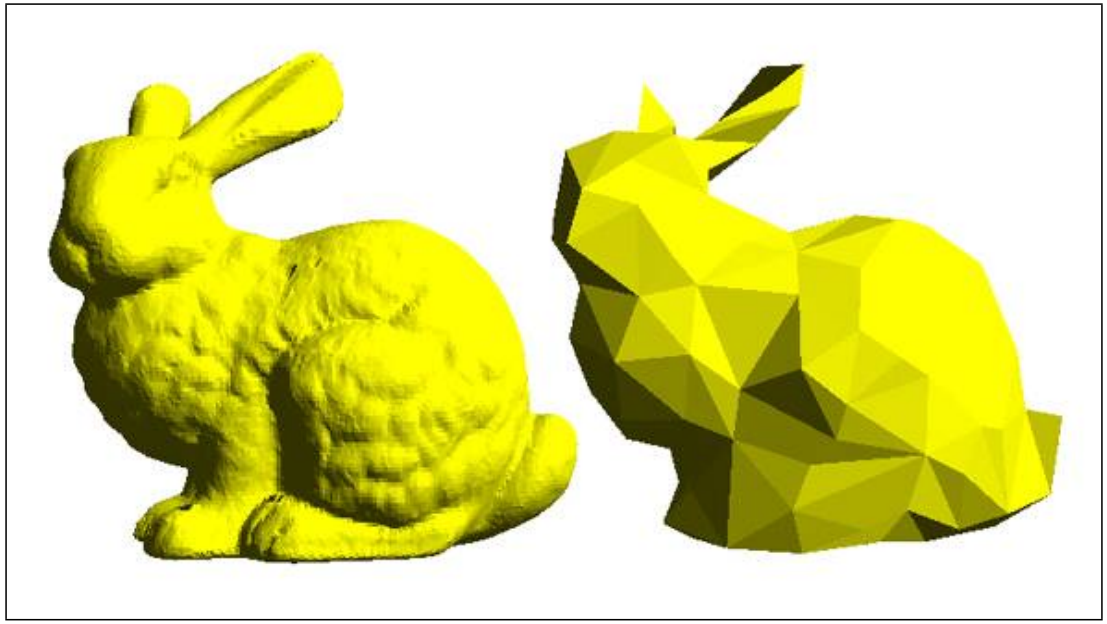
```
l = v->spigolo;

do{
   Stampa_Spigolo(l);
   e = l->gemello;
   l = e->successivo;
} while (l != v->spigolo)
```

- Alcune considerazioni finali sulle strutture dati utilizzate per rappresentare le maglie poligonali
- Come già detto, la stessa applicazione grafica può far uso di più di una struttura dati
- Ad esempio è molto comune, poiché è semplice ed occupa poco spazio, la rappresentazione con la lista di vertici come formato per i file contenenti la geometria di oggetti
- Le applicazioni grafiche in genere caricano tali file ed usano l'informazione contenuta in essi per riempire una struttura dati più utile ai fini algoritmici (per esempio la half-edge)
- Ecco un esempio di file (formato simile a OFF) che descrive un tetraedro con una maglia di triangoli.
- Lo header dice che ci sono 4 vertici e 4 facce. Segue lista dei vertici e poi la lista delle facce, specificate tramite indice dei vertici. Ogni linea che descrive una faccia inizia con il numero di vertici che compongono la faccia (3).

```
4 4
-1 -1 -1
1 1 -1
1 -1 1
-1 1 1
3 1 2 3
3 1 0 2
3 3 2 0
3 0 1 3
```

Semplificazione

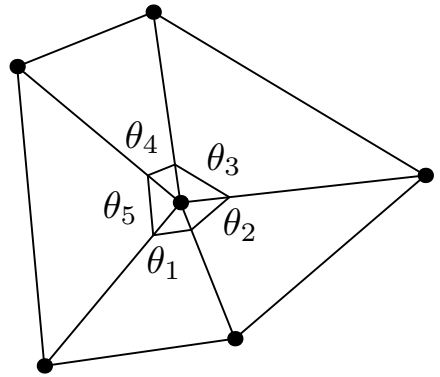
- Le maglie poligonali, se molto accurate, possano essere computazionalmente troppo onerose da gestire.
- È quindi importante poterle semplificare, se necessario.
- Di solito si applica algoritmo incrementale che rimuove un vertice alla volta e ripara il buco lasciato.
- Idealmente vogliamo rimuovere il maggior numero possibile di vertici per cui la risultante maglia semplificata sia una buona approssimazione della maglia fine originale.



- Molte applicazioni richiedono modelli complessi, con un elevato dettaglio, al fine di mantenere un livello di realismo convincente. Per questo motivo i modelli vengono creati o acquisiti ad una risoluzione molto elevata.
- Tuttavia, tanta complessità non è sempre richiesta, e poiché il costo per disegnare un modello è direttamente legato alla sua complessità, è utile possedere una versione semplificata del modello.
- Naturalmente vorremmo che la semplificazione avvenisse in modo automatico.
- Gli algoritmi di semplificazione mirano a questo. Spefificamente, un algoritmo di semplificazione prende in ingresso una maglia triangolare e ne produce una versione versione approssimata ma più semplice, in termini di numero di triangoli.
- Vi sono due categorie di algoritmi:
  - Decimazione dei vertici.
  - Contrazione iterativa degli spigoli.

Decimazione dei vertici

- Si selezionano vertici "poco importanti", basandosi su euristiche locali, si rimuovono e si ri-triangola la cavità risultante.
- Ad ogni passo di rimozione si deve determinare il miglior vertice candidato ad essere rimosso, in base a criteri euristici.
- Un criterio può essere quello di diradare le zone a bassa **curvatura**.
- Una approssimazione della curvatura (Gaussiana), che prende il nome di *angle deficit* si calcola nel seguente modo: ad ogni vertice si associa un numero pari a 2π meno la somma degli angoli interni di tutti i triangoli incidenti sul punto



$$\varepsilon(v) = 2\pi - \sum_i \theta_i(v)$$

- È dimostrato che se $\varepsilon(v) = 0$ allora il vertice v e tutti i vertici ad esso connessi giacciono su un piano.
- Si vede che vertici con ε nullo (o molto piccolo) sono essenzialmente "inutili"

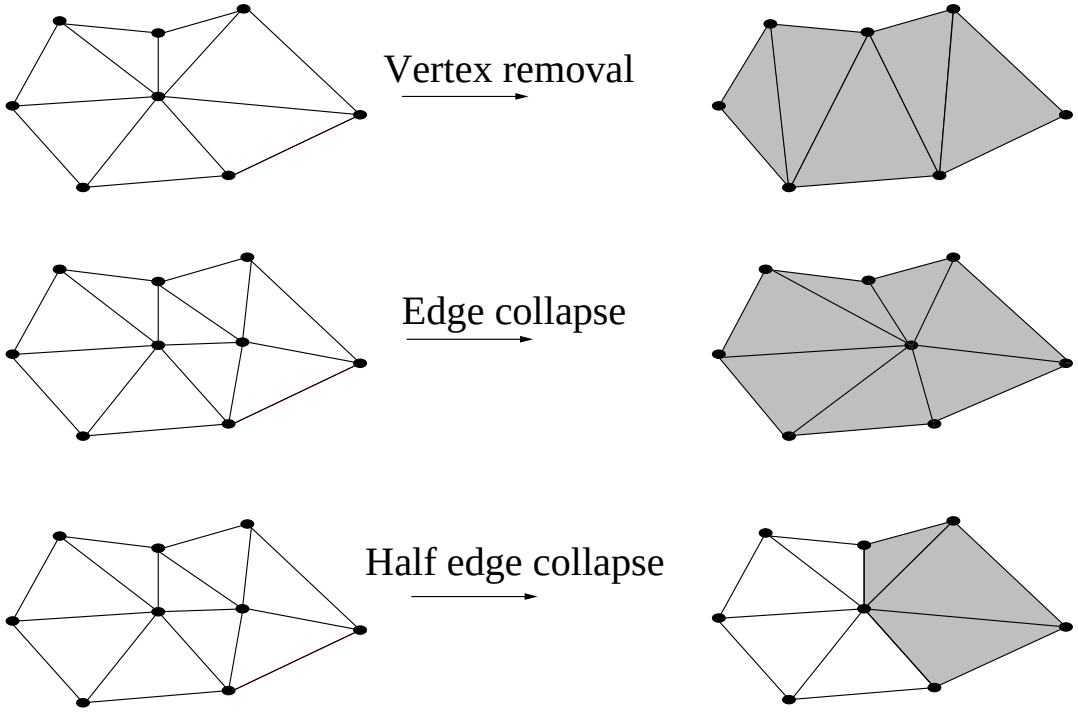
Contrazione iterativa degli spigoli

- Agli spigoli viene associato un costo, tipicamente legato all'errore introdotto dalla contrazione dello spigolo stesso. Ad ogni iterazione viene eliminato per contrazione lo spigolo di costo minore, ed i costi dei vicini vengono aggiornati.
- I vari metodi differiscono per la metrica di errore impiegata.
- Una metrica semplice:

$$\varepsilon(\overline{P_1P_2}) = \frac{\|P_2 - P_1\|}{|\mathbf{n}_\ell \cdot \mathbf{n}_r|}$$

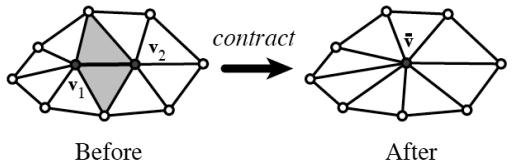
  dove P_1 e P_2 sono i vertici incidenti sullo spigolo e $\mathbf{n}_\ell$ e $\mathbf{n}_r$ le normali delle facce incidenti.

- Vi sono diversi modi per rimuovere un elemento (vertice o spigolo) da una maglia:
  - Il **vertex removal** rimuove un vertice e triangola la cavità risultante.
  - L' **edge collapse** rimuove uno spigolo fondendo due vertici in uno *nuovo*, mentre nel **half edge collapse** uno dei due vertici rimane fermo.



In grigio i triangoli nuovi generati dall'operazione.

Algoritmo di Garland-Heckbert



- L'algoritmo
  di Garland-Heckbert è un algoritmo di semplificazione
  che procede per contrazione iterativa degli spigoli.
- A ciascun vertice $\mathbf{v}$ è associato in insieme di piani, $\Pi(\mathbf{v})$: inizialmente sono i piani definiti dai triangoli incidenti sul vertice. Dopo una contrazione, al nuovo vertice si associa l'unione degli insiemi dei vertici che sono stati contratti

- A ciascun vertice $\mathbf{v}$ è associato un errore $\Delta(\mathbf{v})$, pari alla somma dei quadrati delle distanza di $\mathbf{v}$ dai piani di $\Pi(\mathbf{v})$. 5nizialmente l'errore è 0 per costruzione.

$$\Delta(\mathbf{v}) = \sum_{\mathbf{p}\in\Pi(\mathbf{v})} (\mathbf{p}^T\mathbf{v})^2 = \sum_{\mathbf{p}\in\Pi(\mathbf{v})} (\mathbf{v}^T\mathbf{p})(\mathbf{p}^T\mathbf{v}) = \sum_{\mathbf{p}\in\Pi(\mathbf{v})} \mathbf{v}^T(\mathbf{p}\mathbf{p}^T)\mathbf{v} = \mathbf{v}^T \underbrace{\left(\sum_{\mathbf{p}\in\Pi(\mathbf{v})} \mathbf{p}\mathbf{p}^T\right)}_{Q} \mathbf{v}$$

- Quindi per calcolare l'errore di un vertice basta mantenere la matrice Q (l'insieme dei piani è solo concettuale).
- Il costo di uno spigolo $(\mathbf{v}_1, \mathbf{v}_2)$ è l'errore $\Delta(\bar{\mathbf{v}}) = \bar{\mathbf{v}}^T(Q_1 + Q_2)\bar{\mathbf{v}}$, dove $\bar{\mathbf{v}}$ è la posizione del vertice risultante dalla contrazione di $(\mathbf{v}_1, \mathbf{v}_2)$.

- Gli spigoli vengono mantenuti in uno heap con chiave pari al costo.
- Rimane da specificare la regola che assegna la posizione finale del vertice dopo la contrazione.
- Si potrebbe prendere il punto medio del segmento $\overline{\mathbf{v}_1\mathbf{v}_2}$, ma la cosa migliore è scegliere la posizione $\bar{\mathbf{v}}$ che rende minima $\Delta(\mathbf{v})$ (si risolve con un sistema lineare).
- Dopo la contrazione $\mathbf{v}_1$ e $\mathbf{v}_2$ diventano un vertice solo. Diciamo che $\mathbf{v}_1$ rimane, spostato nella posizione $\bar{\mathbf{v}}$ e $\mathbf{v}_2$ sparisce.
- Schema dell'algoritmo:
    1. Calcola le matrici Q per tutti i vertici della maglia.
    2. Per ciacuno spigolo $(\mathbf{v}_1, \mathbf{v}_2)$ calcola la posizione ottima per il vertice dopo la contrazione $\bar{\mathbf{v}}$. L'errore $\Delta(\bar{\mathbf{v}}) = \bar{\mathbf{v}}^T(Q_1 + Q_2)\bar{\mathbf{v}}$ è il costo associato allo spigolo.
    3. Costruisci uno heap con gli spigoli e chiave pari al costo.
    4. Rimuovi lo spigolo $(\mathbf{v}_1, \mathbf{v}_2)$ di minor costo dalla cima dello heap, effettua la contrazione $(\mathbf{v}_1, \mathbf{v}_2) \leftarrow \bar{\mathbf{v}}$.
    5. $Q_1 \rightarrow Q_1 + Q_2$ ed aggiorna i costi deggli spigoli incidenti in $\mathbf{v}_1$
    6. Ripeti da 4.
- Bisogna prestare attenzione a non creare incoerenze nella maglia, come p.es. il *fold-over*.

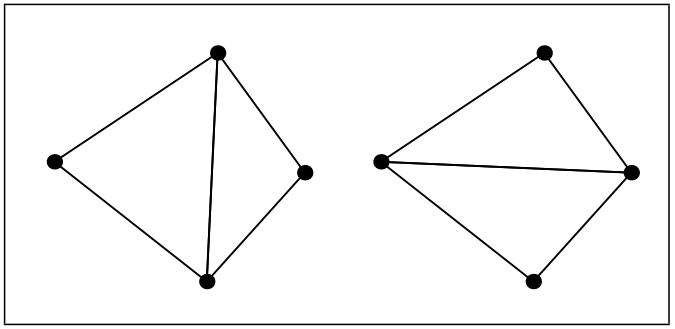
- Un algoritmo di semplificazione come questo genera una piramide di modelli a complessità crescente, ovvero una rappresentazione multirisoluzione dell'oggetto.
- Si parte da un modello M_0 (maglia triangolare)
- Si semplifica il modello, in modo opportuno, ottenendone uno nuovo M_1 che sia più piccolo (come numero di triangoli) del precedente
- Si itera il procedimento n volte, ottenendo una **piramide** di modelli $M_0 \dots M_n$
- Tale piramide può essere usata per svariati compiti
- Da notare che in realtà, non vengono archiviati tutti i modelli, ma solo M_n e tutte le mosse di semplificazione inverse.

Vediamo tre applicazioni

1. **Livello dinamico di dettaglio**: utilizzo la piramide per variare la complessità dell'oggetto a seconda del punto di vista (inutile proiettare un modello di 5000 triangoli su 4 pixel).
2. **Compressione di una maglia**: con una opportuna fase di codifica, lo spazio occupato da M_n e dalla lista di movimenti di ricostruzione è più piccolo dello spazio occupato da M_0. Questo può dare vantaggi notevoli, per esempio, nella trasmissione remota di modelli.
3. **Rendering progressivo**: è possibile ottenere un rendering progressivo, ovvero ottenere una prima immagine di M_n, e poi progressivamente "migliorarla" percorrendo la piramide dei modelli. Analogo a quanto già siamo abituati a vedere nella trasmissione e visualizzazione di immagini JPEG su internet.

Ottimizzazione

- L'ottimizzazione è concettualmente più semplice della semplificazione.
- Si tratta di cambiare la connettività della maglia senza rimuovere o spostare i vertici, in modo da ottenere una superficie di miglior qualità.
- Il passo elementare è l'**edge flipping** (o **edge swapping**), come nella triangolazione di Delaunay.



- Il criterio di ottimizzazione riflette la geometria della superficie. Per esempio si potrebbe voler massimizzare la lisciezza della superficie penalizzando triangoli adiacenti le cui normali sono troppo diverse.
- Nel caso di maglie triangolari planari (vertici in $\mathbb{R}^2$)) l'algoritmo di Delaunay si può vedere come un algoritmo di ottimizzazione della maglia, secondo il criterio del massimo angolo più piccolo.

Riassumendo

- Le maglie poligonali sono semplici dal punto di vista della creazione
- Sono semplici da manipolare
- Esiste però un compromesso tra precisione della rappresentazione e spazio occupato (in memoria)
- Un altro compromesso è tra la semplicità di effettuare ricerche algoritmiche sulla maglia e lo spazio occupato
- Inoltre non sempre il rendering risulta perfetto; a volte diventa percepibile la tessellazione sottostante
- In ogni caso è tuttora la tecnica più usata nelle applicazioni grafiche interattive

Curve e superfici

- Abbiamo visto diversi aspetti della descrizione di oggetti 3D tramite rappresentazioni poligonali
- A seconda delle necessità i modelli poligonali possono risultare non utilizzabili; se per esempio si vuole un alto grado di approssimazione di una superficie curva, il numero di poligoni può diventare troppo grande per essere manipolato in tempi ragionevoli
- Studiamo adesso alcune curve fondamentali utili per la modellazione di oggetti complessi
- Il vantaggio principale nell'uso di superfici per modellare un oggetto è l'assenza del problema della tessellazione visibile (vedremo comunque che non è del tutto vero)
- Fino a pochi anni fa l'uso di curve, o superfici liscie, era impensabile per applicazioni in tempo reale; per lo più venivano utilizzate in fase di modellazione o per rendering non interattivo
- Negli ultimi tempi le cose sono cambiate ed oggi cominciano ad apparire applicazioni di grafica avanzata che usano superfici curve anche in tempo reale
- Iniziamo interessandoci di curve unidimensionali; il passaggio a superfici sarà quindi banale.

Forma implicita e parametrica

- Una curva o una superficie può essere rappresentata sia in forma **implicita**, sia in forma **parametrica**
- La forma implicita è più compatta; ad esempio un cerchio di raggio unitario può essere scritto come

$$x^2 + y^2 = 1$$

- La forma parametrica richiede di trovare due funzioni $f(t)$ e $g(t)$ di un parametro t tali che la curva possa essere descritta dalle equazioni

$$x = f(t)$$
$$y = g(t)$$

  al variare di t in un certo intervallo (di solito $[0, 1]$)

- **Esercizio 3**: dare una formulazione parametrica del cerchio unitario
- Le curve parametriche sono le più usate in grafica al calcolatore; la forma implicita è usata solo in casi particolari (ad esempio nei ray-tracer)

Curve parametriche

- Forma parametrica di una curva nello spazio

$$P(t) = (p_x(t), p_y(t), p_z(t), 1)$$

- In genere si usano **polinomi** di grado fissato per le funzioni di t che descrivono la curva
- Un esempio banale di curva parametrica lo abbiamo già visto: le rette

$$P(t) = Q + t\mathbf{u} = (q_x + tu_x, q_y + tu_y, q_z + tu_z, 1)$$

- Vettore tangente alla curva

$$P'(t) = \frac{\mathrm{d}P(t)}{\mathrm{d}t} = (p'_x(t), p'_y(t), p'_z(t), 0)$$

- Quindi la derivata di un punto in forma parametrica è un vettore (lo si vede anche dal fatto che l'ultima coordinata omogenea derivata si annulla)

Curve polinomiali a tratti

- Come per rappresentare una curva al primo ordine si usa una spezzata (lineare a tratti), così anche nel caso di rappresentazioni polnomiali non si cerca un polinomio di grado elevato che rappresenti tutta la curva, ma si usa una rappresentazione polinomiale a tratti di grado basso (tipicamente 3).
- Si deve fare in modo che le giunzioni tra i diversi polinomi siano "continue".
- Continuità; una curva si dice **k-continua** nel punto $p(t)$ e si indica con C^k se è derivabile k volte in quel punto
- Due curve possono essere unite per gli estremi; in tal modo si garantisce una curva risultante C^0, ovvero continua
- Per ottenere una curva C^1 si devono imporre ulteriori condizioni; le derivate delle due curve negli estremi che si congiungono devono coincidere
- Una richiesta meno forte è che le due derivate (che sono vettori) giacciano sulla stessa retta ed abbiano la stessa direzione; in tal caso la curva è G^1 (continuità geometrica)

Curve razionali

- Non tutte le curve nel piano ammettono una rappresentazione parametrica polinomiale; si introducono quindi le **curve razionali**
- Si consideri una curva parametrica in cui anche la quarta componente è descritta da un polinomio in t

$$P(t) = (p_x(t), p_y(t), p_z(t), p_w(t))$$

Sappiamo che non è un punto nello spazio (ultima coordinata diversa da 1

- Se normalizziamo otteniamo

$$P(t) = (\frac{p_x(t)}{p_w(t)}, \frac{p_y(t)}{p_w(t)}, \frac{p_z(t)}{p_w(t)}, 1)$$

- Questa è una curva **razionale**
- Per esempio, il cerchio in due dimensioni non è descrivibile con una curva parametrica polinomiale, ma lo è con una curva razionale.

Interpolanti vs Approssimanti

- In computer graphics si descrivono in genere le curve con l'ausilio di punti
- Bisogna distinguere tra curve **interpolanti** e curve **approssimanti**
- Le prime interpolano i punti, ovvero passano per questi; le seconde invece adattano la propria forma ai punti, ma non è detto che vi passino
- Le curve interpolanti si usano in genere per specificare le traiettorie nelle animazioni (in questo caso io voglio che la traiettoria passi per certi punti);
- Le curve approssimanti sono invece utilizzate più per la modellazione e si possono descrivere come combinazioni lineari di n punti $P_1 \cdots P_n$ i cui coefficienti sono opportune funzioni polinomiali o razionali di un parametro t

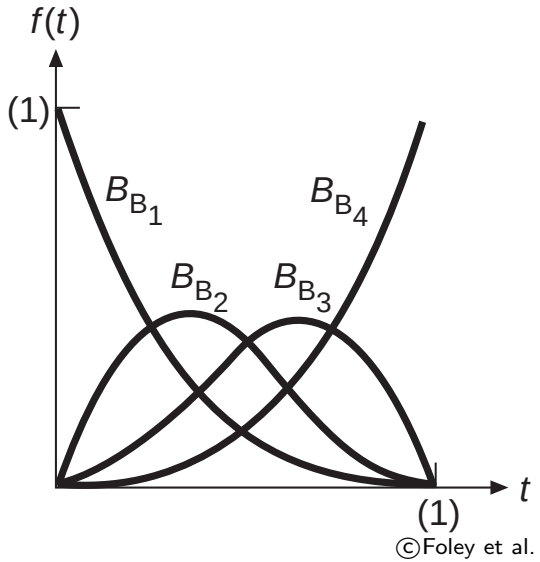
$$P(t) = \sum_i B_i(t) P_i$$

- Le funzioni di **blending** $B_i(t)$ determinano il tipo di curva.
- P.es i polinomi di Bernstein danno origine alle curve di Bézier (v. figura), la scelta di altri polinomi porta alle B-splines o alle NURBS.

Curve di Bézier

- Inventate (o scoperte) contemporaneamente da **Bézier** e da **De Casteljau**
- Siano dati $n+1$ punti $P_0 \dots P_n$ detti **punti di controllo**
- La poligonale data dai punti di controllo si chiama **poligonale di controllo**
- La curva di Bézier è data dalla seguente forma parametrica:

$$P(t) = \sum_{i=0}^{n} B_{n,i}(t) P_i$$



dove le funzioni

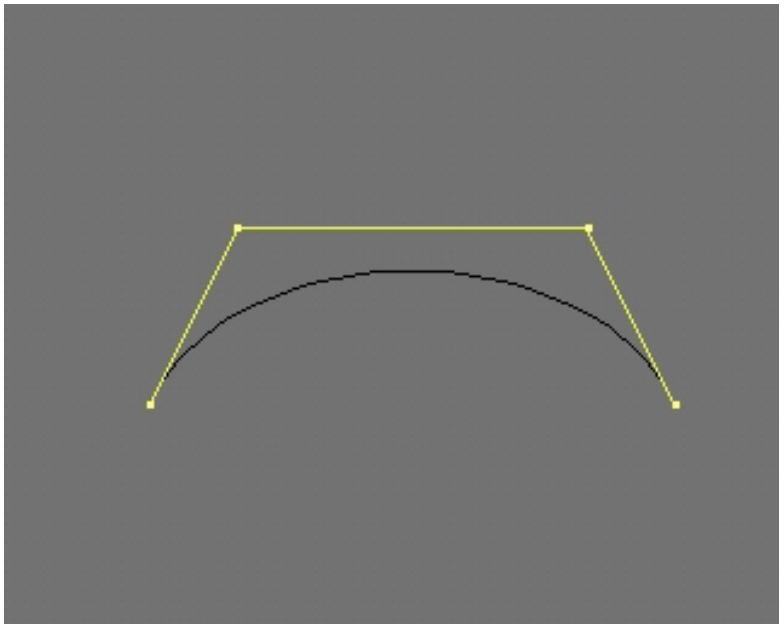
$$B_{n,i} = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i}$$

sono le **funzioni di base** ed il parametro t appartiene all'intervallo $[0, 1]$

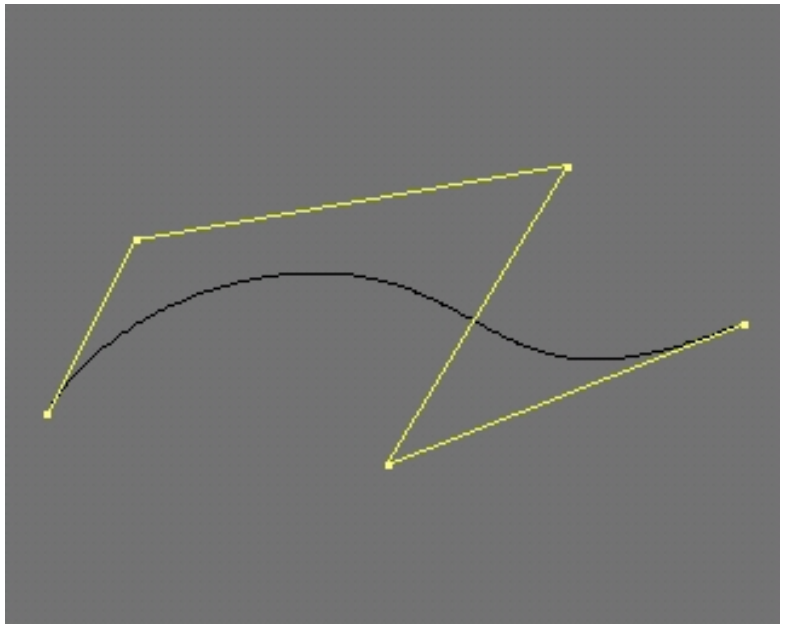
- È facile dimostrare che

$$\sum_{i=0}^{n} B_{n,i}(t) = 1 \quad \forall t \in [0,1]$$

- Dunque il punto $P(t)$ è una opportuna combinazione affine dei punti di controllo; al variare di t tale punto percorre la curva



(13) 4 punti di controllo



(14) 5 punti di controllo

Proprietà

1. Ciascun coefficiente della combinazione affine è un polinomio di grado n in t (se i punti di controllo sono $n+1$)
2. La curva $P(t)$ passa da P_0 e da P_n (per i valori di t pari a 0 ed 1 rispettivamente)
3. $B_{n,i}(t) \geq 0 \quad \forall i$
4. Le funzioni di base definiscono una **partizione dell'unità**, ovvero si sommano a 1
5. La curva è interamente contenuta nell'inviluppo convesso dei punti di controllo
6. La curva è invariante per trasformazioni affini, ovvero basta trasformare i punti di controllo per avere la curva trasformata

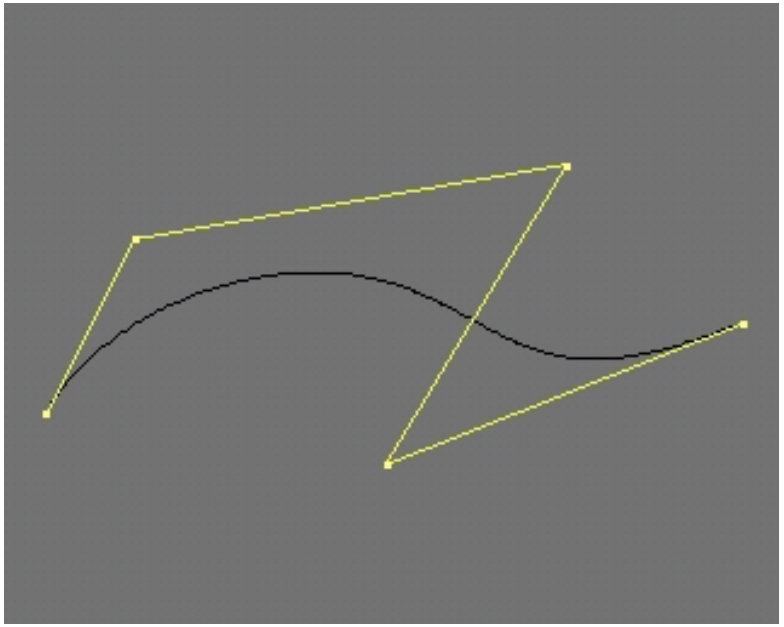
- Come cambia la curva se si sposta un punto di controllo?

$$P_k \longrightarrow P_k + \mathbf{v}$$

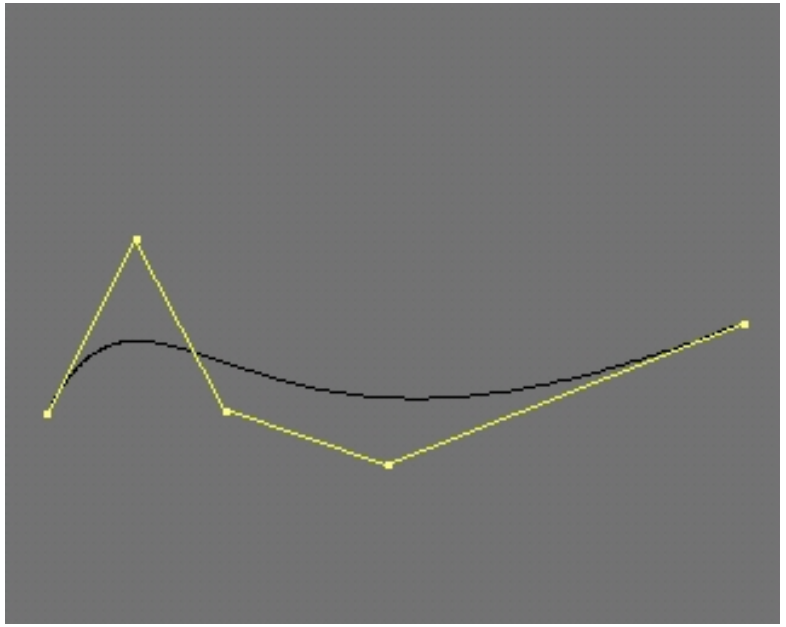
- È facile dimostrare che

$$P(t) \longrightarrow C(t) = P(t) + B_{n,k}(t)\mathbf{v}$$

- Da questo si vede che spostando un punto di controllo si spostano **tutti** i punti della curva
- I punti di controllo sono quindi **non locali**, una loro perturbazione si ripercuote su tutta la curva
- Questo deriva dal fatto che le funzioni di blending per le curve di Bézier hanno supporto su tutto l'intervallo $[0, 1]$

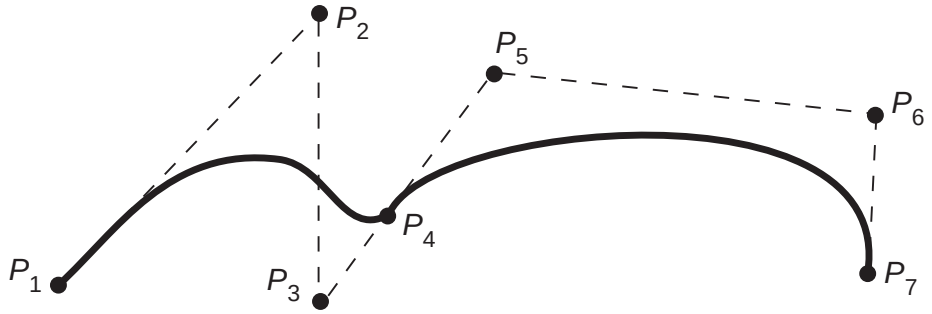


(15) Prima dello spostamento



(16) Dopo lo spostamento

- Tipicamente si considerano polinomi cubici (quindi 4 punti di controllo).
- I punti estremi determinano la "posizione" della curva (ci deve passare), mentre i due interni controllano la derivata agli estremi.
- L'incollatura di due curve di Bézier cubiche sarebbe solo C^0. Per ottenere G^1 bisogna costringere i tre punti di controllo a cavallo della giuntura ad essere collineari.



©Foley et al.

L'algoritmo di De Casteljau

- Algoritmo ricorsivo per ottenere un punto sulla curva di Bézier, fissato un valore del parametro t.
- Si definisce una **gerarchia** di punti

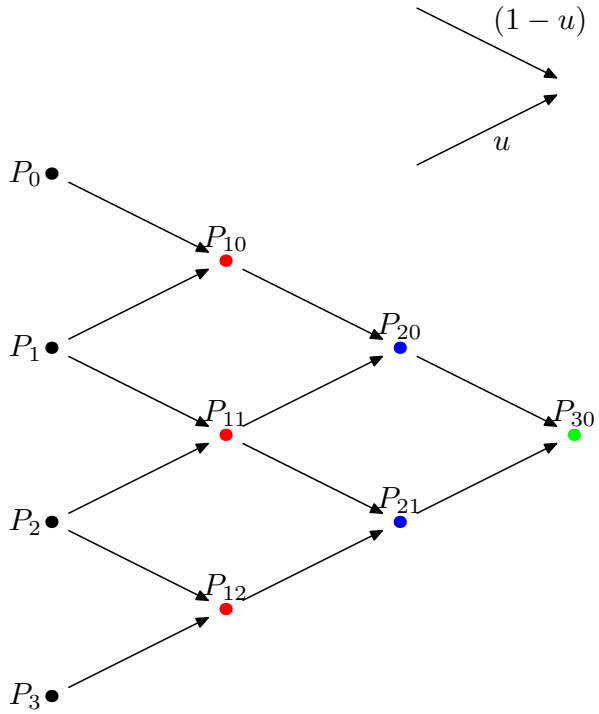
$$
\begin{gathered}
P_0 \dots P_n \\
P_{1,0}(t) \dots P_{1,n-1}(t) \\
P_{2,0}(t) \dots P_{2,n-2}(t) \\
\vdots \\
P_{n-1,0}(t) P_{n-1,1}(t) \\
P_{n,0}(t)
\end{gathered}
$$

data dalla seguente formula ricorsiva

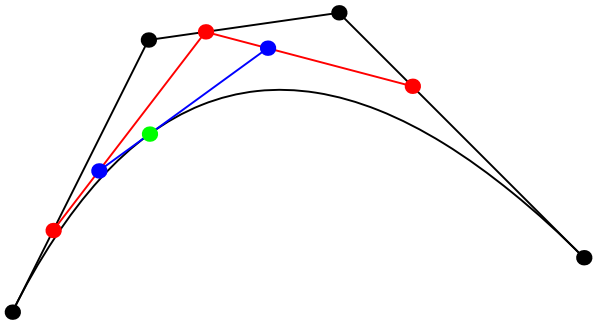
$$P_{i,j}(t) = (1-t)P_{(i-1),j} + tP_{(i-1),(j+1)}$$

dove $i = 1 \dots n$, $j = 0 \dots n-i$ e dove si intende che i punti di controllo, che sono al livello zero della gerarchia (e che non dipendono dal parametro t), possano essere indicati con $P_{0,0} \dots P_{0,n}$

- Si dimostra che l'ultimo vertice della gerarchia descrive proprio la curva di Bézier, al variare di t, ovvero $P_{n,0}(t) = P(t)$



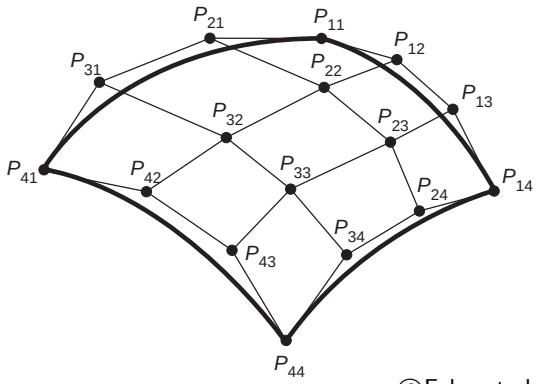
(17) Diagramma



(18) Interp. geometrica

Superfici

- Una volta capite le curve parametriche viste fino ad ora, il passaggio a superfici parametriche è immediato



©Foley et al.

Patch di Bézier bicubico con i suoi 16 punti di controllo.

- Una superficie parametrica dipenderà da due parametri (u, v), anziché da un parametro solo t come nel caso delle curve. Sarà quindi descritta da un punto $P(u, v)$ tale che

$$p_x = f_x(u, v)$$
$$p_y = f_y(u, v)$$
$$p_z = f_z(u, v)$$

- In ogni punto della superficie sono definiti due vettori tangenti, dati dalle derivate rispetto a u e rispetto a v rispettivamente

$$P_u(u,v) = \frac{\partial P(u,v)}{\partial u} \quad e \quad P_v(u,v) = \frac{\partial P(u,v)}{\partial v}$$

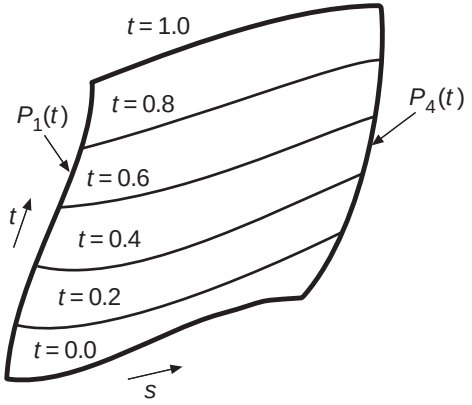
- La normale alla superficie può allora essere calcolata semplicemente come il prodotto vettore di questi due vettori tangenti

$$\mathbf{n}(u,v) = P_u(u,v) \times P_v(u,v)$$

  eventualmente normalizzata

- Fissato uno dei due parametri, ad esempio $u = u_0$, il punto $P(u_0, v)$ percorrerà una curva parametrica con parametro v. Stessa cosa per u.

- Tali curve sono chiamate **isoparametriche**



©Foley et al.

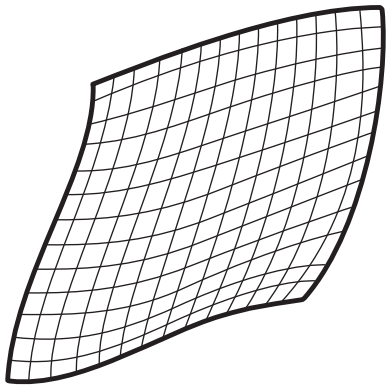
- Date le funzioni di blending $X_{n,i}(t)$, che possono essere quelle di **Bézier**, delle **B-Spline** o delle **NURBS**, e dati $n \times m$ punti di controllo P_{ij}, la superficie è definibile nel seguente modo

$$P(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} X_{n,i}(u) X_{m,j}(v) P_{ij}$$

- Per rappresentare la superficie di un oggetto 3D, invece che usare facce planari, si usao superfici parametriche (tip. bicubiche) che usualmente vengono denominate **patch**, opportunamente "saldate" tra loro.
- La saldatura consiste nel forzare un certo grado di continuità nei raccordi.

Conversione in maglia poligonale

- Vedremo che sarà utile (per disegnarle) convertire la superfici parametriche in maglie poligonali.
- I vertici della maglia si ottengono valutando le funzioni parametriche per valori discreti dei parametri u e v.

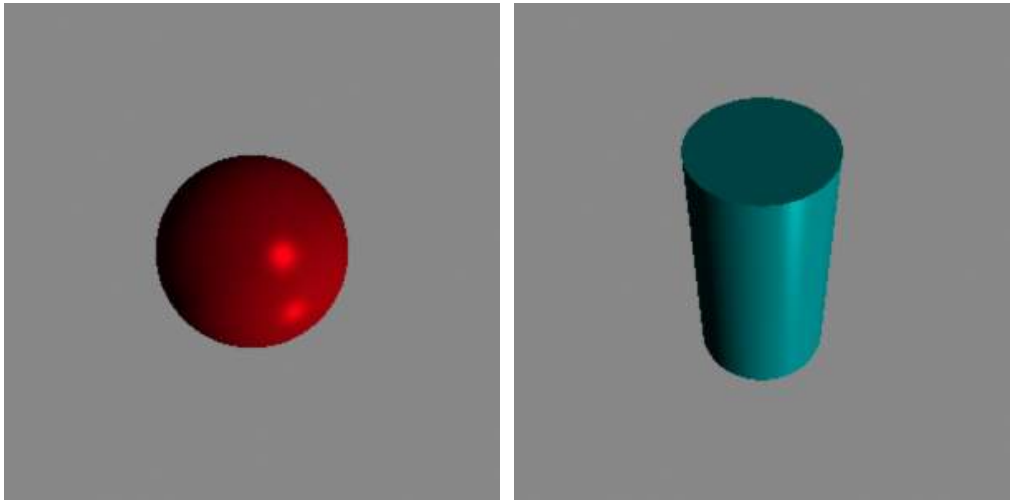


©Foley et al.

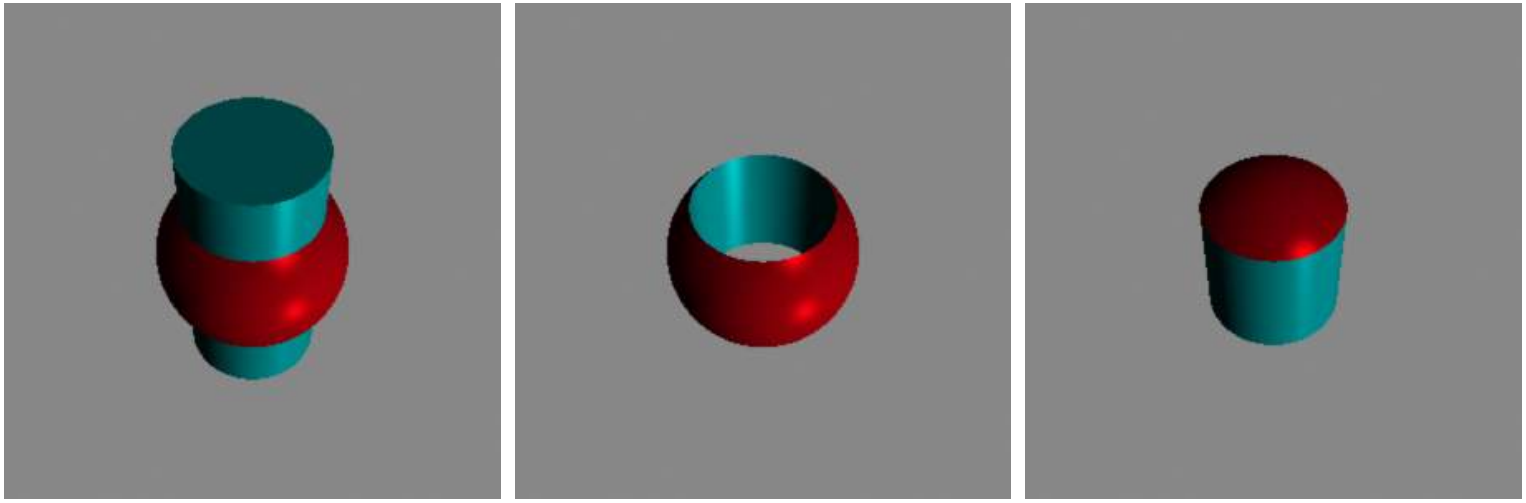
- Considerando i valori discreti assunti dai parametri si ottiene una famiglia di curve isoparametriche che solcano la superficie determinandone una suddivisione in patch quadrangolari. Ciascuno di questi viene suddiviso in due triangoli passanti per i vertici, ottenendo così una maglia triangolare.
- La valutazione efficiente della funzione parametrica avviene con il metodo di Horner o con le differenze in avanti.
- Altri metodi sono basati su schemi di suddivisione ricorsiva (algoritmo di De Casteljau).

Geometria Costruttiva Solida

- Passiamo ora a rappresentazioni **volumetriche** degli oggetti.
- Una rappresentazione volumetrica esatta di oggetti è data dalla cosidetta **CSG** o **C**ostructive **S**olid **G**eometry
- È una rappresentazione particolarmente adatta per il modeling (è diffusa nel settore CAD), ma meno efficiente per il rendering.
- Si tratta, essenzialmente, di costruire degli oggetti geometrici complessi a partire da oggetti elementari (cubi, sfere, cilindri etc) e da operazioni booleane tra questi
- Quali sono le operazioni booleane? Consideriamo due oggetti semplici A e B



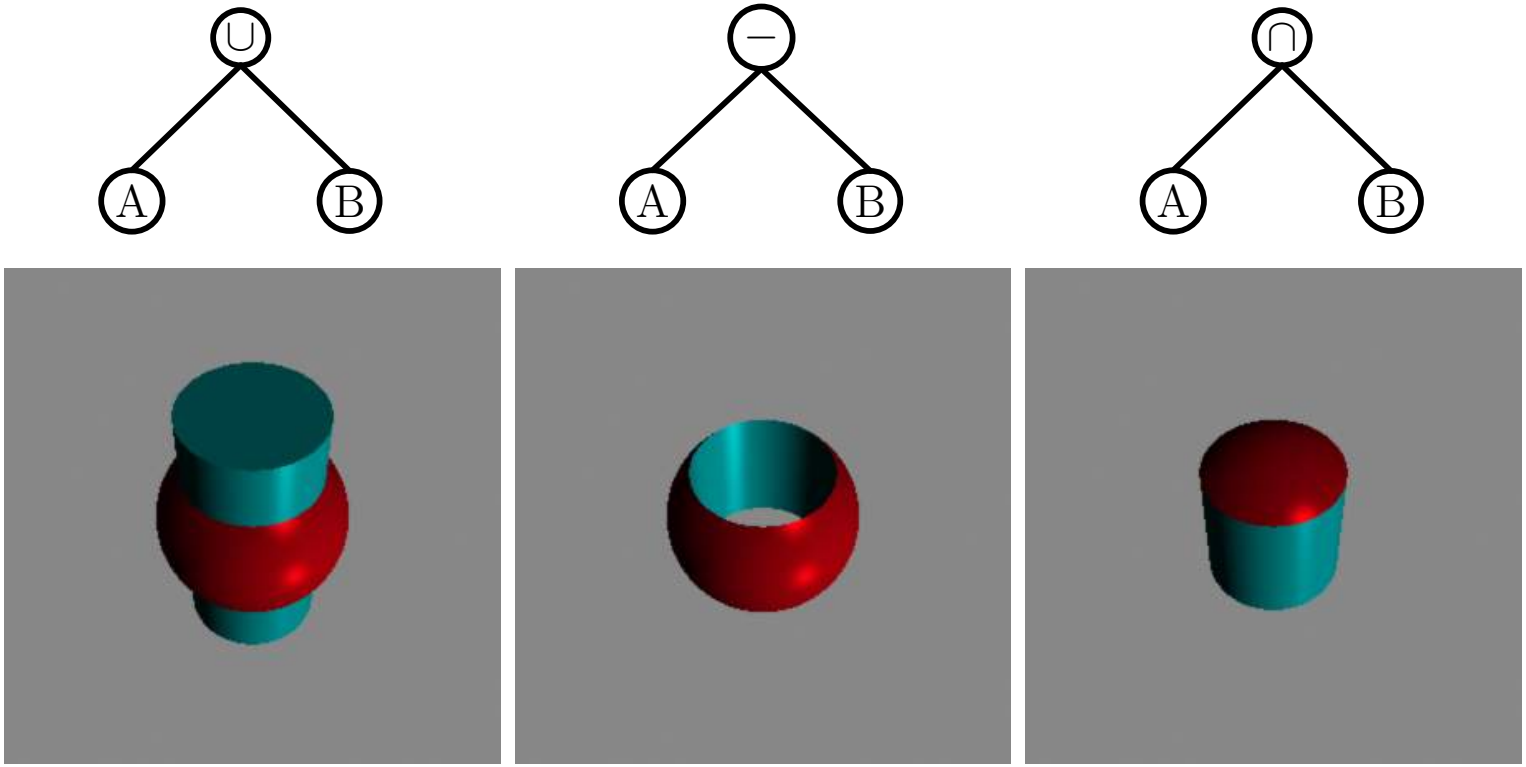
Operazioni booleane su solidi elementari



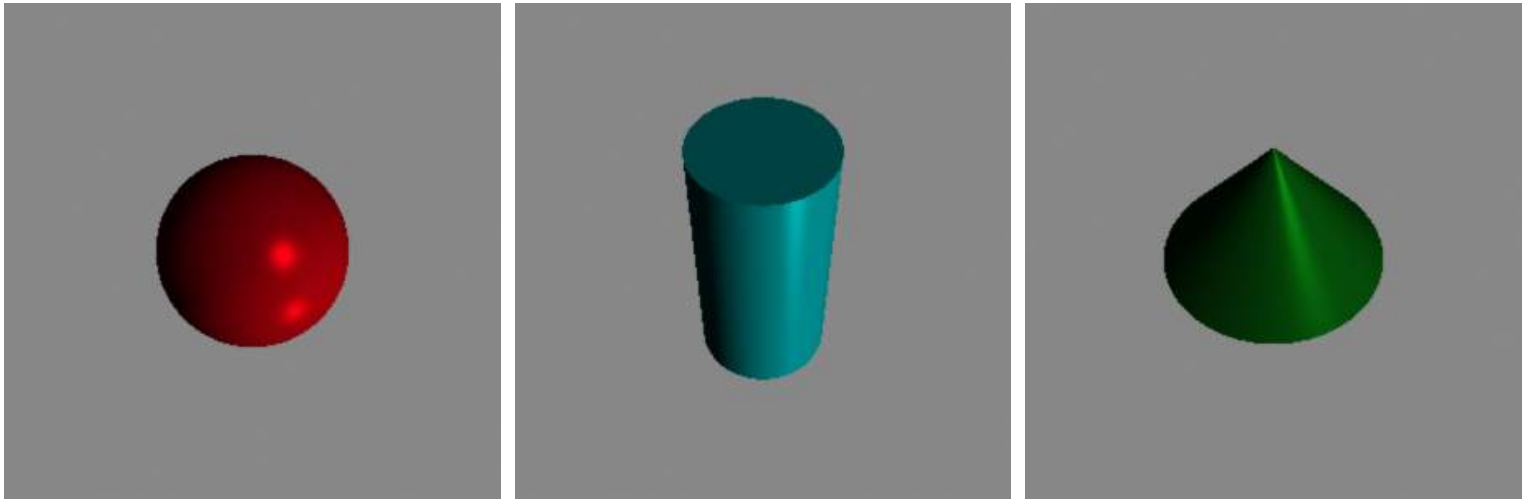
1. **Unione:** l'unione $A \cup B$ è l'insieme dei punti che appartengono ad almeno uno dei due solidi (or non esclusivo)
2. **Differenza:** la differenza $A - B$ è l'insieme dei punti che appartengono ad A, ma non a B
3. **Intersezione:** l'intersezione $A \cap B$ è l'insieme dei punti che appartengono ad A ed a B (and)

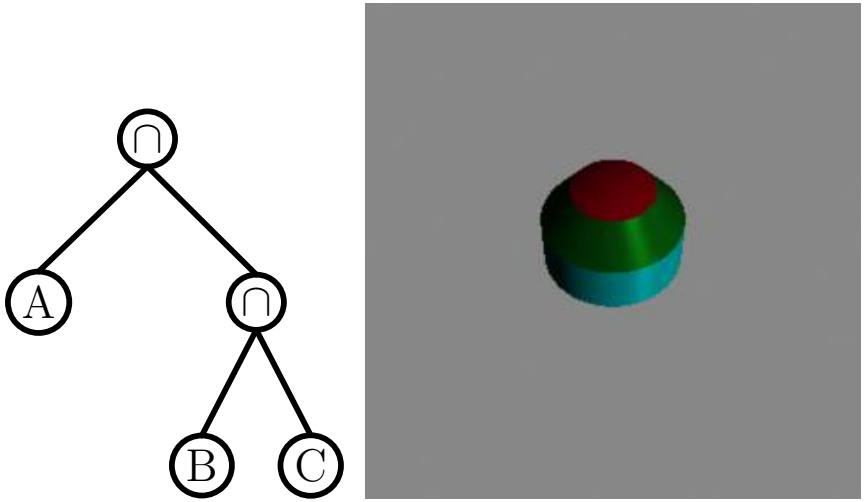
Alberi CSG

- Le operazioni CSG possono essere descritte tramite un albero (gerarchia)
- Ciascun nodo di un albero che non sia una foglia contiene una delle tre operazioni elementari $\cup$, $\cap$ o $-$
- Ciascuna foglia contiene una primitiva



- Ovviamente si possono fare cose più complesse



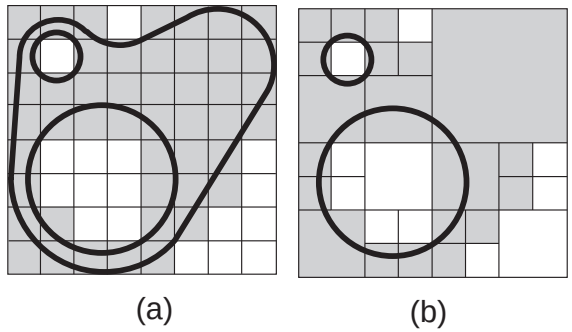


Partizionamento spaziale

- Rappresentazione di regioni in base allo spazio occupato.
- Lo spazio viene suddiviso in celle adiacenti dette pixel (o voxel): una cella è "piena" se ha intersezione non vuota con la regione, è detta vuota in caso contrario.
- Una rappresentazione di una scena complessa ad alta risoluzione richiederebbe l'impiego di un numero enorme numero di voxel, per cui questa rappresentazione è limitata a singoli oggetti.
- Nel caso di una rappresentazione volumetrica voxelizzata vedremo come ottenere un rappresentazione poligonale della superficie mediante l'algoritmo dei marching cubes.
- Quadtrees e octrees vengono utilmente impiegati nella rappresentazione di **regioni** (del piano o dello spazio): si parla in tal caso di **region quadtrees** e **region octrees**.

Region quadtree e octree

- Nel caso del region quadtree si parte con un quadrato contenente la regione, e la suddivisione ricorsiva si ferma quando un quadrante contiene tutte celle piene o tutte celle vuote.
- La rappresentazione della regione che si ottiene è più economica rispetto alla enumerazione delle singole celle, poiché grandi aree uniformi (piene o vuote) vengono rappresentate con una sola foglia (anche se nel caso peggiore il numero delle foglie è pari a quello delle celle).
- L'octree supporta in modo efficiente ricerche di intersezione voxel-raggio (per il ray-casting).



- In questa figure si vede una oggetto rappresentato (a) con enumerazione delle celle e (b) con un quadtree. Sotto è raffigurato il corrispondente albero (F=full,E=empty,P=partially full). Tratto dal Foley et al.