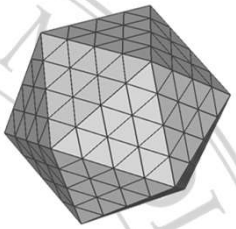
Marco Tarini - Computer Graphics 2024/2025
Università degli Studi di Milano

Mesh poligonali: regolarità

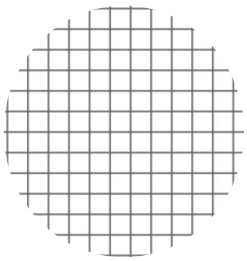
argomento svolto nella lezione del 18-03-2025



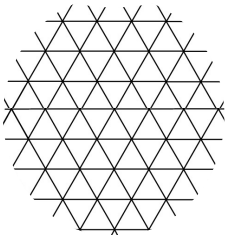
92

Regolarità di una mesh: concetto

- ✓ Osservazione: un piano può essere tassellato in modo **regolare** da quadrati ==>
- ✓ Tanto più la *connettività* di una **quad-mesh** si avvicina a questo caso, tanto più la consideriamo "**regolare**"



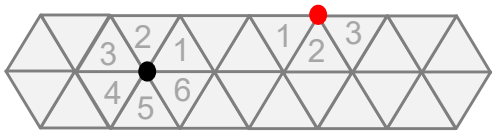
- ✓ Osservazione: un piano può essere tassellato in modo **regolare** da triangoli equilateri ==>
- ✓ Tanto più la *connettività* di una **tri-mesh** si avvicina a questo caso, tanto più la consideriamo "**regolare**"

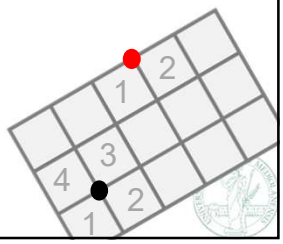


93

Regolarità di una mesh definizioni

- ✓ Vertice interno = vertice che non compare su un edge aperto
  - ⇒ Altrimenti: il vertice è di bordo
- ✓ «Valenza» di un vertice
  - ⇒ numero di facce adiacenti ad quel vertice
  - ⇒ a volte: numero di edge adiacenti ad quel vertice
  - ⇒ per tutti i vertici interni: le due definizioni sono equivalenti
- ✓ «Vertice regolare» interno = un vertice di valenza…
  - ⇒ …4, in una quad-mesh
  - ⇒ …6, in una tri-mesh
- ✓ «Vertice regolare» di bordo = un vertice di valenza…
  - ⇒ …2, in una quad-mesh (2 facce, e 3 edge)
  - ⇒ …3, in una tri mesh (3 facce, e 4 edge)





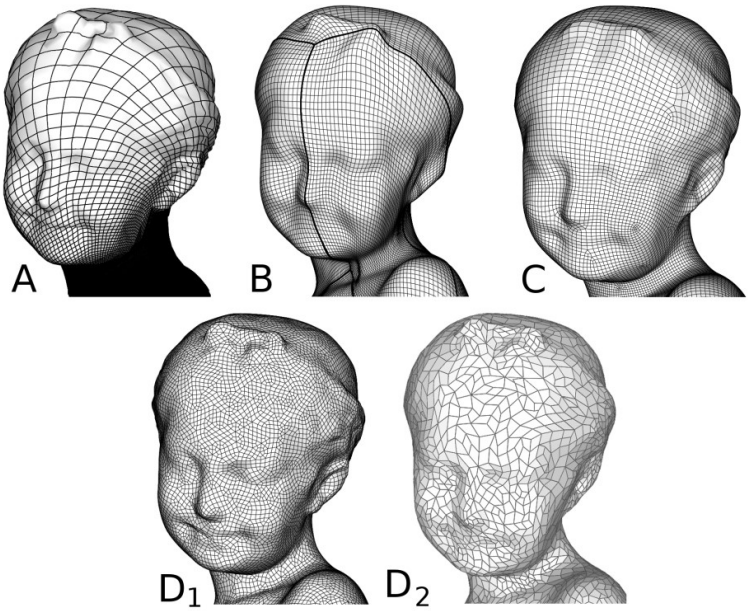
96

Regolarità di una mesh definizioni

- ✓ Quanti vertici in una mesh (tri o quad) sono regolari?
  - ⇒ Tutti => la mesh è (*perfettamente)* regolare
    (è detta anche «structured» mesh)
  - ⇒ Quasi tutti - per es il 99% => la mesh è «semi-regolare»
  - ⇒ Pochi (per es, solo 2/3 o la metà) = è una mesh irregolare
  - ⇒ Nota: la regolarità di una mesh è una questione di grado
- ✓ Quad-mesh fortemente regolare = in pratica, è un grigliato

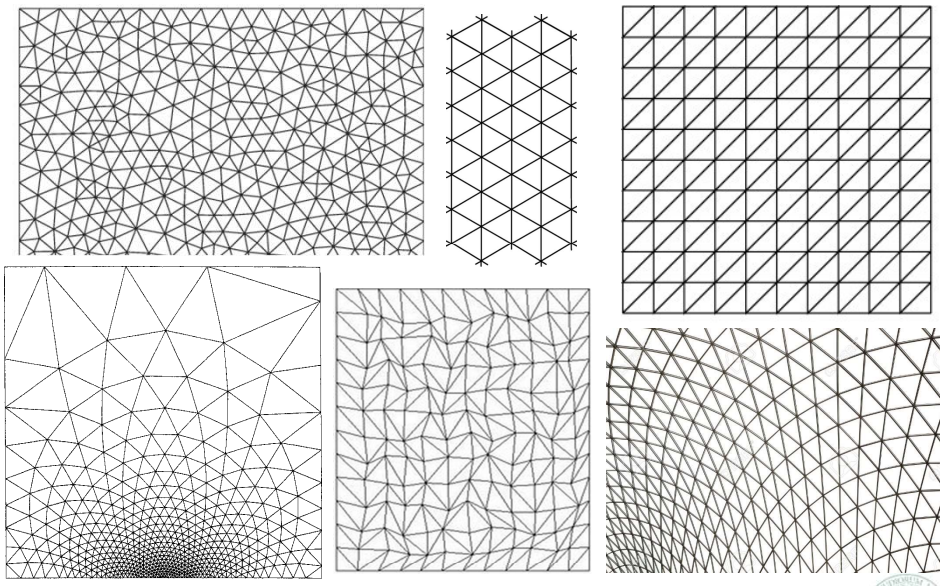
97

Mesh: la regolarità è questione di grado



98

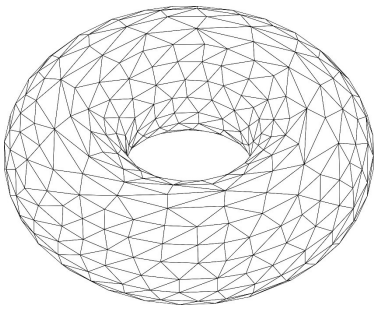
Regular VS irregular Triangular mesh



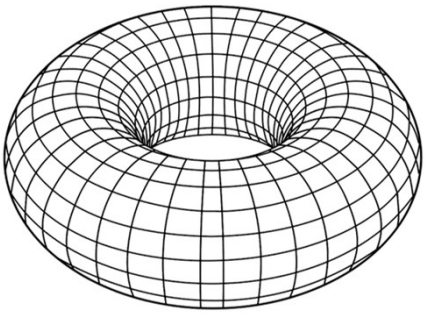
Esercizio: trova i vertici irregolari. Quali mesh sono (semi-) regolari?
Ripeti con altre mesh che vedi in questi lucidi

100

Regolare VS irregolare



una tri-mesh
irregolare

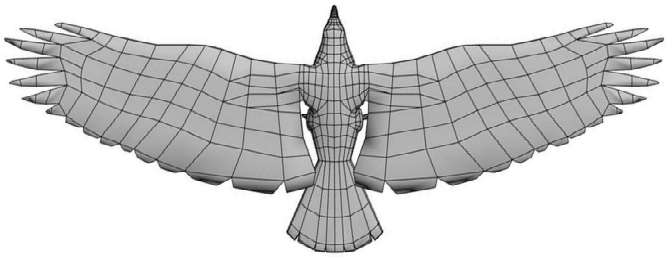


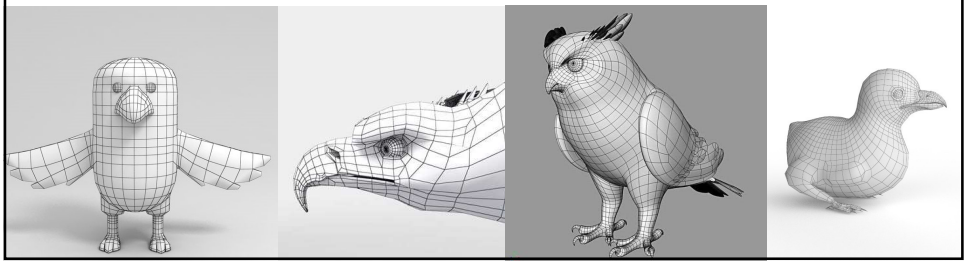
una quad-mesh
regolare

102

Regolarità delle mesh

✓ Le mesh modellate *manualmente* da modellatori (nel CAD, nel direct low-poly editing [vedi dopo]) tendono ad essere **quad-mesh regolari**

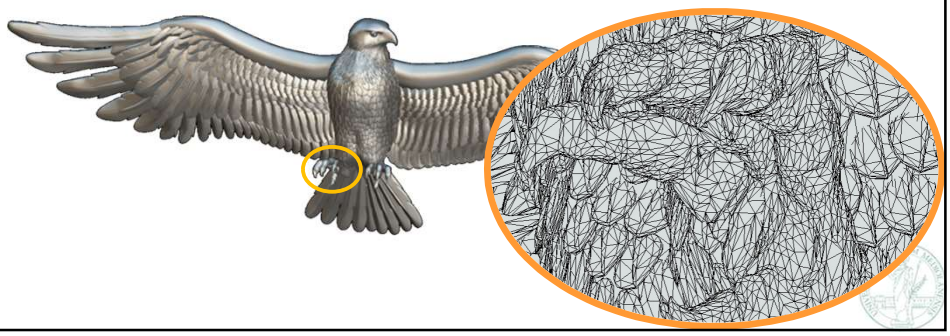




103

Regolarità delle mesh sul campo

- ✓ Sono di solito **tri-mesh irregolari** le mesh
  - ⇒ **acquisite** 3D da modelli reali (per es, scanning o fotogrammetria)
  - ⇒ prodotte o ri-processate attraverso tecniche di **geometry processing** per esempio tutti che vedremo: semplificazione automatica, front advancing methods, triangolazioni di Delaunay, vertex clustering… (eccetto ovviamente per le tecniche che si prefiggono questo obiettivo, come il semi-regular remeshing)
  - ⇒ Prodotte da artisti digitali con tecniche di **digital sculpting** (vedi dopo)



104

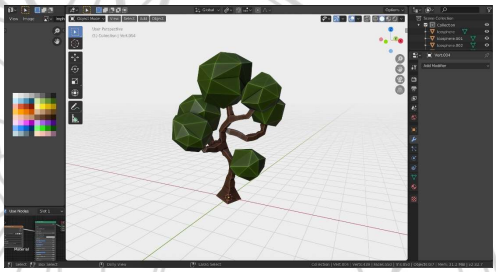
Molti vantaggi (e alcuni svantaggi) della regolarità

- ✓ Una mesh più regolare tende ad essere
  - ⇒ Più agevole da modificare / o ri-editare manualmente (ad esempio, consente la selezione di «stream di edge»)
  - ⇒ Più adatta ad essere animata
  - ⇒ Più efficiente da comprimere per risparmiare memoria (o da mandare in streaming in rete)
  - ⇒ Maggiormente adatta ad applicazioni di Machine Learning
  - ⇒ Spesso maggiormente accurata geometricamente, a parità di risoluzione
  - ⇒ Meno soggetta a presentare artefatti di rendering
  - ⇒ Tuttavia, la risoluzione di una mesh regolare tende ad avere una risoluzione meno adattiva: l'adattività «si paga» in termini di regolarità
- ✓ In generale, la regolarità mesh è una caratteristica desiderabile
  - ⇒ ma spesso difficile da ottenere in modo automatico
  - ⇒ Nota: per l'efficienza di rendering, non fa differenza
  - ⇒ Domanda: cosa si ottiene effettuando un diagonal-split a tutte le facce di una quad-mesh **regolare**?

105

Marco Tarini - Computer Graphics 2024/2025
Università degli Studi di Milano

Modelli poligonali: modellazione manuale



111

Come vengono generate le mesh

- ✓ Cattura dalla realtà
  - ⇒ Cioè 3D Acquisition
  - ⇒ Per es, laser scanning, fotogrammetria…
- ✓ Generazione procedurale
  - ⇒ Per es, modelli di piante o vegetali generati attraverso una simulazione della loro cresita
- ✓ Simulazione
  - ⇒ Per es, una simulazione fisica di un liquido per generare la sua superficie in movimento
- ✓ Modellazione manuale
  - ⇒ Da parte di artisti digitali

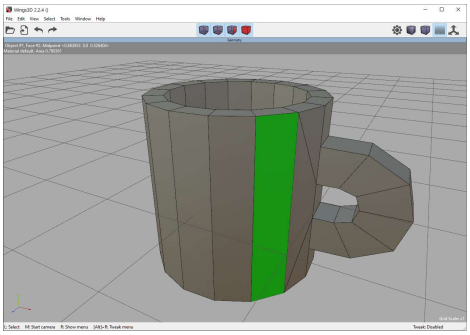
← questo argomento

112

Generazione manuale di mesh (cenni)

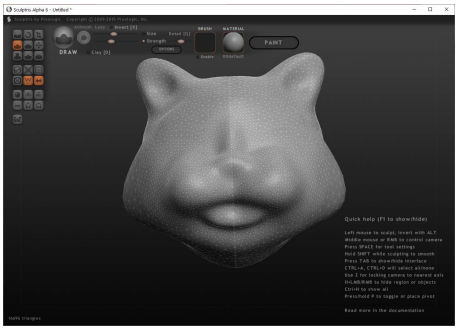
✓Due paradigmi di modellazione manuale:

Direct-poly modelling



esempio con Wings3D

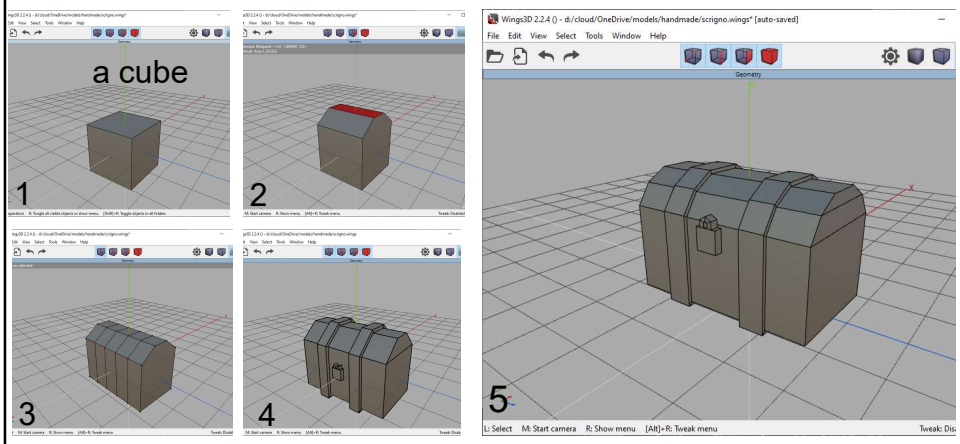
Digital Sculpting



esempio con Sculptris Alpha

113

Direct Low-poly Modelling



with wings3D

114

Direct (Low-poly) Modelling: note

- ✓ Il modellatore umano manipola direttamente, attraverso un'apposita interfaccia, gli elementi della mesh
  - ⇒ come facce, edge, vertici
- ✓ Il modellatore edita esplicitamente
  - ⇒ La connettività della mesh (per es, quali edge connettono quali vertici)
  - ⇒ La posizione dei vertici (singolarmente, o a piccoli gruppi)
- ✓ Le operazioni mantengono automaticamente two-manifoldness (e a volte la chiusura) delle mesh
- ✓ Tipico risultati: mesh poligonali, spesso quad-dominant, e spesso semi-regolari, e spesso a bassa risoluzione
- ✓ All'occorrenza vengono usati operazione di «suddivisione» (che vedremo più avanti)

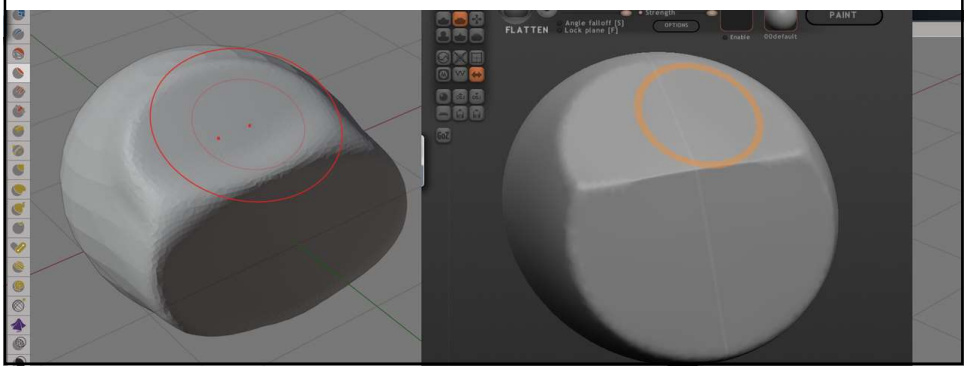
115

Digital Sculpting

mouse
(o stylus)

=

scalpello



116

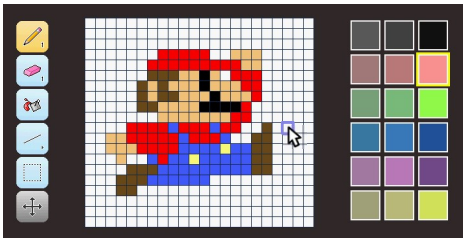
Digital sculpting: note

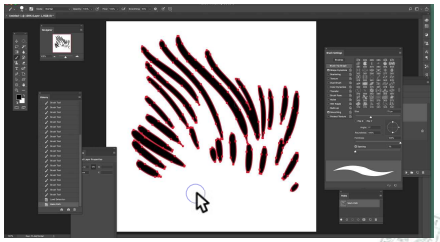
- ✓ Il modellatore scolpisce la forma attraverso pennellate («brush strokes») che imitano la manipolazione di un materiale plasmabile come creta / pongo / etc
  - ⇒ Ma ovviamente in modo libero da ogni vincolo fisico: per es, è possibile rimuovere o creare materiale a piacere
- ✓ La rappresentazione della mesh (facce, vertici, edge) non è esposta al modellatore, ma viene gestita internamente
- ✓ «Pennelli» (anzi, scalpelli) digitali con effetti diversi, per es, local smoothing (rendere più liscio), appiattire, estrudere, «pizzicare», scavare, etc
- ✓ Il sistema gestisce il meshing automaticamente in background,
  - ⇒ per es effettuando edge-collapse o edge split (l'operazione inversa) per mantenere una risoluzione adattiva adatta alla forma che viene generata
- ✓ Maggiormente adatta per modelli biologici / naturali
- ✓ Tipico risultato: una tri-mesh irregolare ad alta risoluzione

117

Un'analogia con immagini rasterizzate

- ✓ Anche per l'editing di immagini rasterizzate (cioè costituite da matrici di pixel) esistono …
  - ⇒ approcci dove (come nel low-poly modelling) viene esposta la struttura interna dell'immagine: l'operatore umano manipola direttamente i pixel (vedi «pixel-art»)
  - ⇒ e approcci dove (come del digital sculpting) si utilizza un'analogia (in questo caso, per es, pennelli e vernici) per consentire ad un operatore di generare l'immagine *prescindendo* dai pixel di cui è composta





118

Authoring delle mesh (cioè 3D modelling)

✓ Alcuni dei software di modellazione più diffusi

- ✓ **3D Studio Max** (autodesk) , **Maya** (autodesk) , **Cinema4D** (maxon) **Lightweight 3D** (NewTek), **Modo** (The Foundry) , **Blender** (open source!) ,
  - ⇒ all-purpose,completi
- ✓ **AutoCAD** (autodesk), **SolidWorks** (SolidThinking)
  - ⇒ Specializzati per il CAD
- ✓ **ZBrush** (pixologic), **Mudbox** (autodesk)
  - ⇒ Specializzati per lo scuplting
- ✓ **Wings3D**
  - ⇒ Solo low-poly modelling (& superfici di suddivisione) open-source, piccolo, specializzato
- ✓ **Sculptris Alpha**
  - ⇒ Solo sculpting open-source, piccolo, specializzato
- ✓ **[Rhinoceros]**
  - ⇒ parametric surfaces (NURBS)
- ✓ **FragMotion**
  - ⇒ small, specialized on animated meshes
- ✓ + molti altri con scopi più specifici
  - ⇒ editing of human models, of architectural interiors, environments, or specific editors for game-engines, etc...

119

Alcuni dei formati di interscambio mesh più diffusi

.OBJ (wavefront)
- ☺ Molto diffuso
- ☺ Indexed mesh
- ☺ ASCII: facile da leggere
- ☹ attributi per vertice: solo normali, UV-map (vedi poi)

.PLY (cyberware)
- ☺ customizzabile: qualsiasi attributo per vertice o per faccia
- ☹ abbastanza usato ma forse solo in ambito accademico

.FBX (AutoDesk)
- ☺ Abbastanza diffuso
- ☹ Propretario (non OpenSource)
- ☹ Supporto per mesh animate

. glTF (Chronos)
"Graphic Library Tranmission Format"
- ☺ molto completo e customizzabile
- ☺ open standard
- ☹ include animazioni, materiali, e molto altro

. DAE – COLLADA (Chronos)
In pratica versione precedente di glTF
- ☺ bastato su XML

121

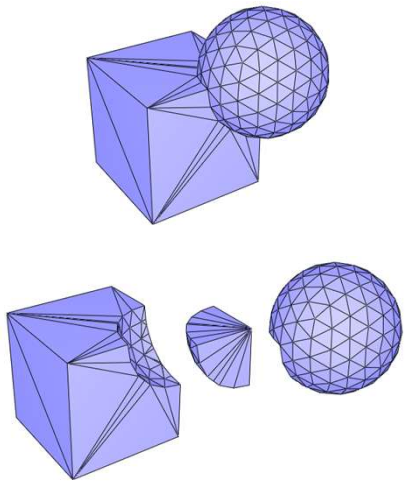
Marco Tarini - Computer Graphics 2024/2025
Università degli Studi di Milano

Mesh poligonali:
Mesh Processing 1/2

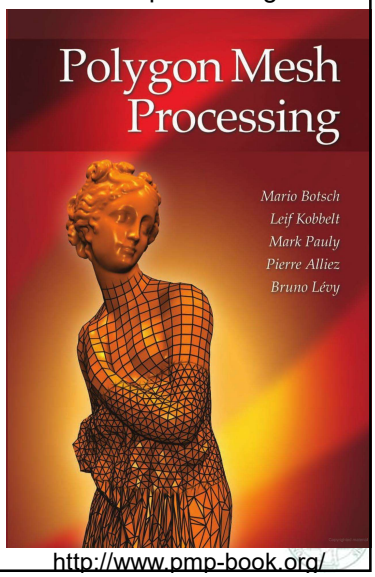
123

Mesh processing (in termini generali)

Il **Geometry Processing**
eseguito su **mesh poligonali**



Un buon manuale
per le basi al
mesh processing:



http://www.pmp-book.org/

125

Mesh processing (in termini generali)

- ✓ Il **mesh processing** è **geometry processing** eseguito su delle mesh poligonali
  - ⇒ Esempi che abbiamo già visto:
  - ⇒ la costruzione di una mesh da una nuvola di punti (meshing di una point cloud), o “surface reconstruction”.
  - ⇒ La stima di normali per faccia o per vertice
- ✓ Gli obiettivi sono i più vari
  Oggi ne vediamo due o tre esempi classici:
  - ⇒ Mesh Simplification (o mesh coarsening)
  - ⇒ Remeshing (detto anche “retopology”), e in particolare, Semiregular Quad Re-meshing

126

Semplificazione automatica di una mesh

- ✓ Da: mesh hi-res
  a: mesh low-res (detta anche low-poly mesh)
- ✓ Procedimento chiamato anche
  - ⇒ «Mesh coarsening»
  - ⇒ «Poly-reduction» (soprattutto in ambiente industriale)
  - ⇒ «Mesh decimation»
- ✓ Perché è utile: la risoluzione deve essere adatta all’applicazione che usa la mesh
  - ⇒ Molti dei procedimenti su una mesh (rendering compreso!) hanno una *complessità lineare* col numero di elementi: maggiore la risoluzione della mesh, più lento l’algoritmo
- ✓ Osservazione:
  - ⇒ in una nuvola di punti, bastava scegliere un sottoinsieme
  - ⇒ per mesh, è più complicato: non posso rimuovere elementi senza danneggiare le proprietà della mesh (es. chiusura)

127

Semplificazione automatica di una mesh: obiettivo

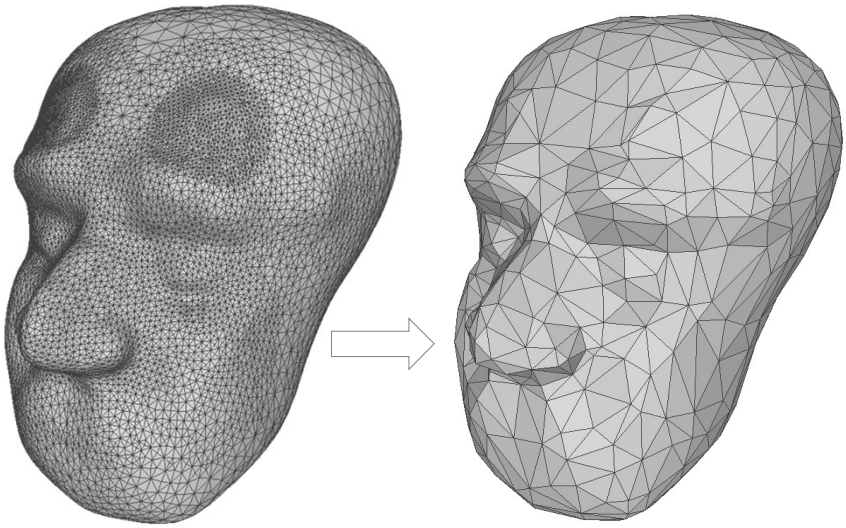
- ✓ Obiettivo: ottenere un buon bilancio fra:
  - ⇒ costo (risoluzione della mesh risultante, cioè numero di elementi residui)
  - ⇒ qualità (errore geometrico introdotto rispetto alla mesh originale)
- ✓ Q: come definisco l'errore introdotto?
  - ⇒ A: misuro la *distanza geometrica* fra le due superfici descritte da mesh originale e mesh semplificata
  - ⇒ cioè assumiamo: mesh originale = «ground truth»
- ✓ Q: come definisco la distanza fra due superfici?
  - ⇒ def matematica: la risposta esula da questo corso
  - ⇒ per approfondire: cercare «hausdorff distance»
- ✓ Q: come la posso calcolare?
  - ⇒ un task di geometry processing – ma la risposta esula da questo corso
  - ⇒ per approfondire: google search for «Metro: measuring error on simplified surfaces»

128

Geometry Processing: mesh simplification

Input: 19K facce

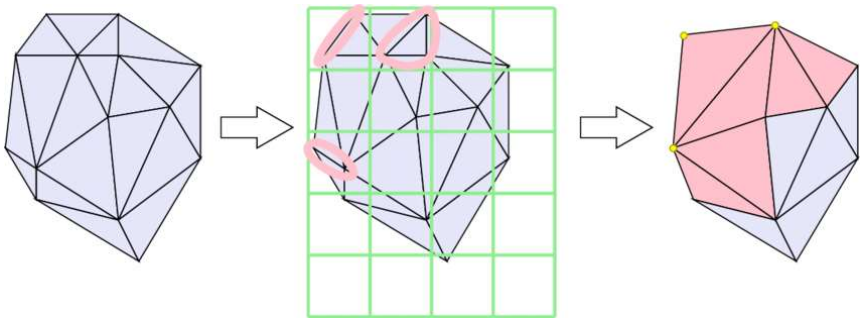
Output: 1200 facce



(esempio eseguito con MeshLab)

130

Mesh simplification: Strategia “Vertex clustering”:



131

Mesh simplification: Strategia “Vertex clustering”:

1. Suddivido virtualmente lo spazio 3D che contiene la mesh in una griglia regolare di “cubetti”, tutti di lato *k*
2. Per ogni vertice della mesh originale, trovo il cubetto a cui appartiene
   - ⇒ ogni vertice della mesh in input si trova all’interno di un cubetto.
   - ⇒ come trovo questo cubetto, data la posizione del vertice?
3. Per ogni cubetto popolato da almeno un vertice, creo UN SOLO vertice in output, che rappresenta tutti i vertici dentro quel cubetto
   - ⇒ assegno al nuovo vertice la posizione media di tutti i vertici finiti nel cubetto
   - ⇒ nota: ciascun vertice originale è rappresentato da uno dei nuovi vertici
4. Processo ogni triangolo della mesh originale: se connette tre vertici originali rappresentati ora da tre vertici diversi, allora genero un triangolo che connette quei tre vertici.
   - ⇒ Altrimenti, cioè quando il triangolo connette tre vertici che stanno tutti o in uno solo o in due soli cubetti distinti, scarto il triangolo

132

Mesh simplification: Strategia “Vertex clustering”: note

- ✓ Il parametro *k* controlla la risoluzione della mesh risultante: tanto maggiore è *k*, tanto minore la risoluzione
  - ⇒ Se *k* è molto piccolo, la mesh prodotta in output è la stessa della mesh in input
- ✓ L’algoritmo è molto semplice da implementare ed è LINEARE con la risoluzione della mesh
  - ⇒Infatti bastano due cicli, uno sui vertici e uno sulle facce

133

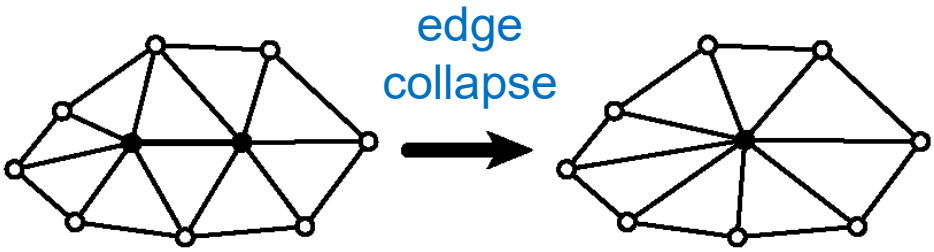
Mesh simplification: Strategia a operazioni locali

- ✓Operazione locale: una operazione che coinvolge (e modifica) solo una piccola parte della mesh
  - ⇒Modifica la geometria e la connettività della mesh solo in un intorno
- ✓Iterazione di *operazioni locali*
  - ⇒Repeat
    - scegli un’operazione locale (secondo un criterio)
    - esegui operazione locale (modifica locale della mesh)
  - ⇒until obiettivo raggiunto
    - es: risoluzione target raggiunga
    - es: errore massimo superato

134

Mesh simplification: Strategia a operazioni locali

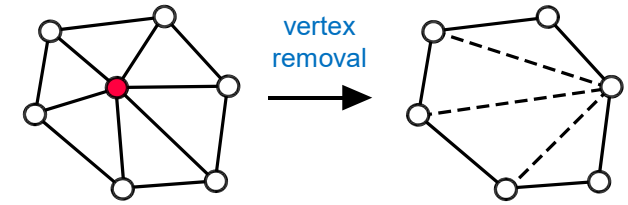
✓Esempio di operazione locale di riduzione della complessità (cioè “coarsening”)



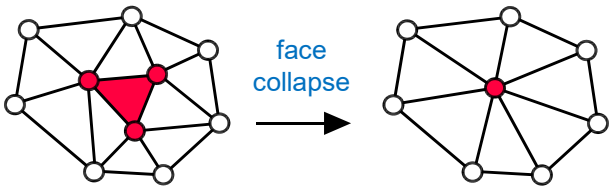
135

Semplificazione automatica di una mesh

✓Altri esempi di operazioni locali di coarsening:



nuovi triangoli:
che ri-poligonalizzano il buco lasciato dalla rimozione del vertice



nota:
si può considerare una successione di due edge collapse

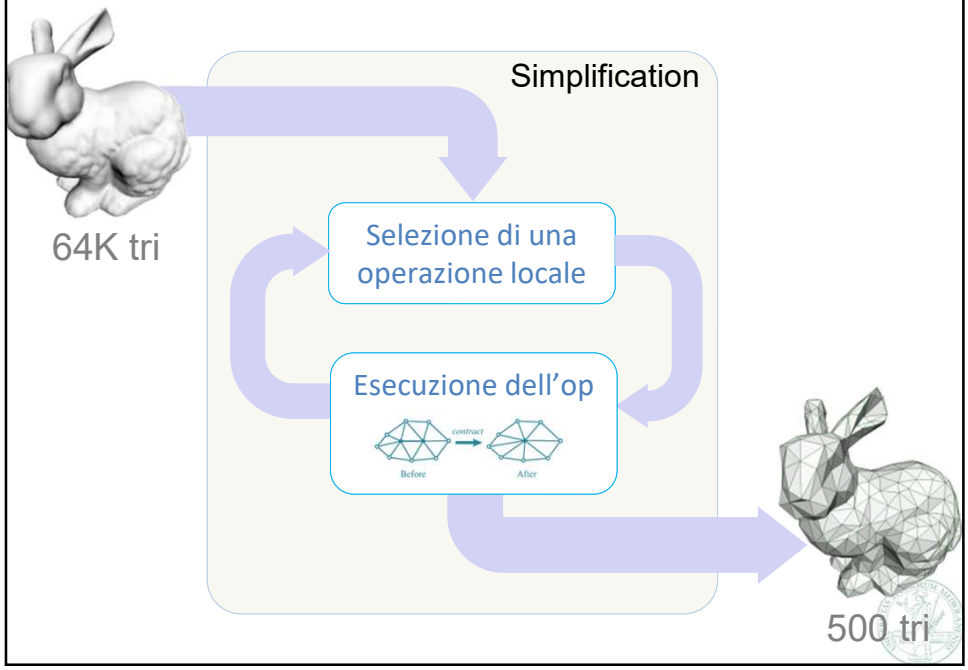
136

Esercizio

- ✓ Valuta le tre operazioni locali esposte sopra.
- ✓ Per ciascuna di esse, valuta di quanto decrementa la risoluzione della mesh in termini di
  - ⇒ numero di facce
  - ⇒ numero di vertici
  - ⇒ numero di edge
  - ⇒ (nota: non dipende dalla mesh, se l'operazione è fatta su un edge, un vertice, o una faccia INTERNA)
- ✓ Osserva che la proporzione fra facce e vertici e edge di una triangle mesh è mentenuta
  - ⇒ Ricorda che, come regola generale, una tri-mesh di n vertici ha circa 2n facce e 3n edges

137

Semplificazione automatica di una mesh



138

Mesh simplification: Strategia a operazioni locali

- ✓ Quali operazioni locali sono possibili?
  - ⇒ (ad un dato step)
  - ⇒ Per l'edge collapse, ogni edge della mesh rappresenta una possiible scelta
- ✓ Quale operazione locale scegliere?
  - ⇒ Possibili molti criteri: ad esempio…
  - ⇒ Quella che introduce un errore di approssimazione più contenuto
  - ⇒ Escludo dalla scelta le operazioni che inficiano le caratteristiche della mesh che mi interessa mantenere (come two-manifoldness)
- ✓ Quanto terminare il processo? (criteri possibili)
  - ⇒ Quando la risoluzione scende sotto un livello prefissato,
  - ⇒ Oppure: quando tutte le operazioni residue introducono un errore troppo elevato

139

Due strategie a confronto

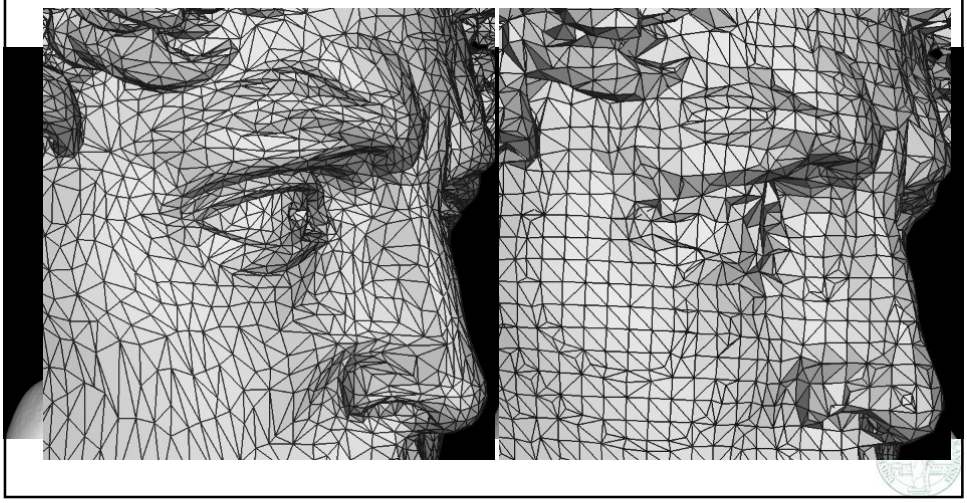
Simplificazione ad operaz locali	Semplificazione a clustering
✓ adattiva	✓ non adattiva
✓ continua	✓ discreta
✓ possibile mirare a triangoli di forma buona	✓ mira a facce di dimensione simile
✓ possibile specificare errore massimo	✓ possibile solo specificare dimensione griglia (solo indirettamente, errore o numero di facce)
✓ possibile specificare numero di facce target	✓ non mantiene caratteristiche mesh
✓ possibile mantenere caratteristiche mesh ⇒ two-manifoldness, chiusura, classe topologia	✓ veloce, semplice da implementare, robusta (funziona su qualsiasi mesh)
✓ spesso *richiede* mesh two-manifold in partenza	

140

Due strategie a confronto

Simplificazione ad edge collapse
(adattva)
215k faces

Semplificazione a clustering
(non adattiva)
235k faces



141

Semplificazione automatica di una mesh

- ✓ Un applicativo generico di mesh processing:

MeshLab

- ✓ Mini-esercizio pratico: testa un algoritmo di mesh decimation
  - ⇒ Ottieni e apri MeshLab:
  - ⇒ Scarica una mesh ad alta risoluzione di esempio (ce ne sono anche sul sito)
  - ⇒ Applica uno di questi due filtri:
    filtro «quadric edge collapse decimation»
    filtro «clustering decimation»

due algoritmi che seguono approcci diversi, come abbiamo visto

143

Task di Geometry Processing: meshing

- ✓ "**Meshing**": (in generale)
  dato una modello 3D, inizialmente non rappresentato come una mesh poligonale, produrre una sua rappresentazione di tipo mesh poligonale
  - ⇒ è un esempio di Surface Reconstruction:
    ricostruzione di una superficie
    (da dati che non sono una superficie)
- ✓ Detta anche poligonizzazione,
  o anche "segmentazione" in analogia con il 2D,
  - ⇒ Per es,
    «meshing di una nuvola di punti», come abbiamo visto
  - ⇒ Vedremo altri esempi di «meshing» a partire
    da altre strutture dati più avanti nel corso

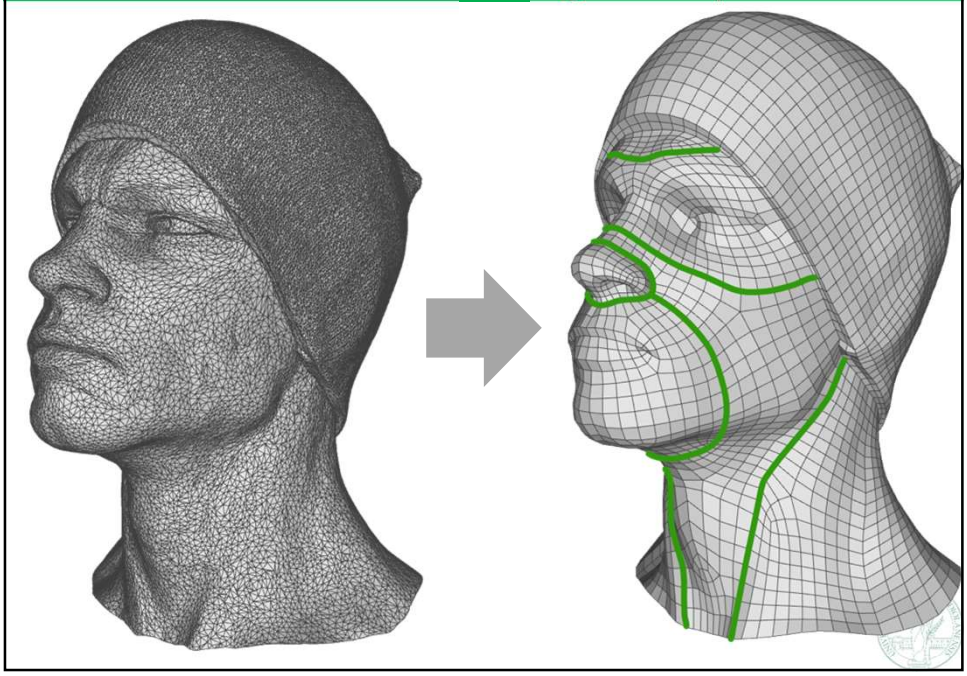
145

Geometry Processing: remeshing

- ✓ **Remeshing**:
  data una superficie già rappresentata come una mesh, costruire una rappresentazione mesh… diversa
- ✓ In ambiente industriale, è detto "retopology"
  - ⇒ A volte viene fatto manualmente da un'artista digitale
    (un modellatore)
  - ⇒ Ma esistono anche algoritmi automatici
- ✓ La mesh di partenza differisce dalla mesh di arrivo
  in termini di, per es…
  - ⇒ da tri a quad-dominant o pure quad («quad-remeshing»)
  - ⇒ da irregolare a (semi)regolare («semiregular-remeshing»)
  - ⇒ da risoluzione non adattiva a risoluzione adattiva
  - ⇒ da forma sfavorevole* delle facce a buona forma
    (* per esempio, triangoli lunghi e stretti o degeneri)

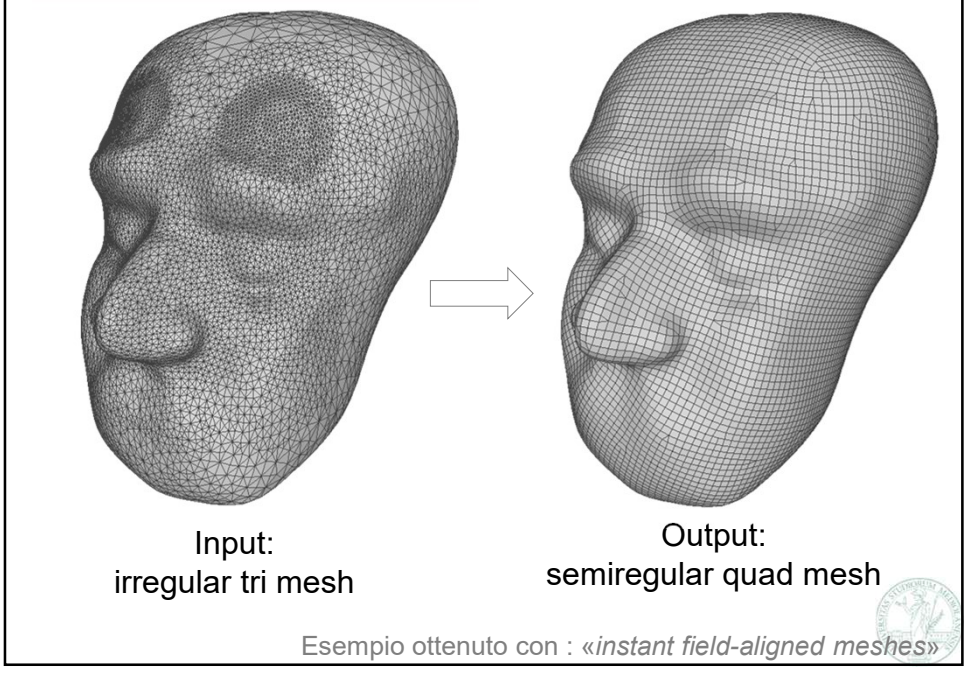
146

Semiregular Quad Remeshing: esempio



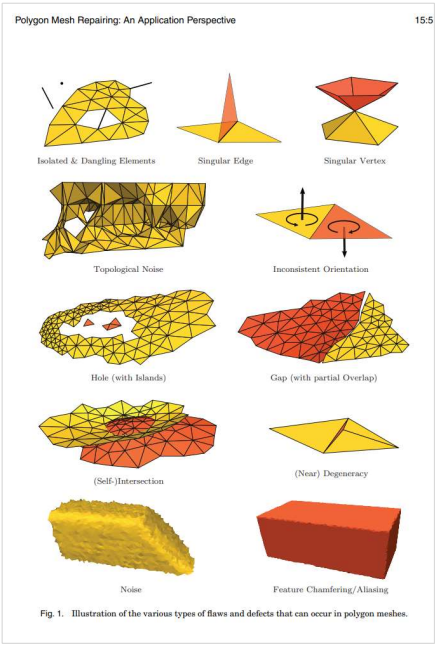
148

Semiregular Quad Remeshing: esempio



149

Geometry Processing: mesh cleaning / repearing



Pro-tip: collezione
di software a
http://meshrepair.org/

Una pagina dal survey:
"Polygon Mesh Repairing: An Application Perspective"
Attene, Campen, Kobbelt, 2012

152

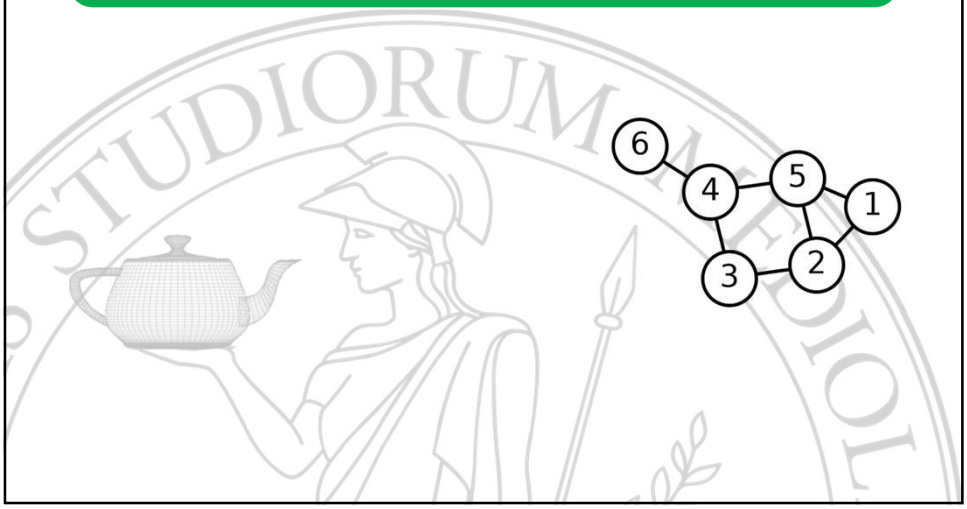
Geometry Processing: mesh cleaning / repearing

- ✓ Mesh cleaning (o repearing):
  l'insieme di task per la rimozione dei difetti della di una mesh, cioè
  - ⇒ le situazioni non two-manifold,
  - ⇒ buchi (per es un poligono mancante)
  - ⇒ Orientamento non consistente delle facce
    (se non è possibile farlo, ma mesh non è «ben orientabile»)
  - ⇒ Auto-intersezioni
  - ⇒ Replicazioni di vertici
  - ⇒ Facce degeneri
- ✓ Molte categorie di mesh, come le mesh scansionate (acquisizione 3D), presentano spesso molti di questi difetti
- ✓ Spesso, è un preprocessing necessario per consentire l'applicazione di vari altri procedimenti di geometry processing

153

Marco Tarini - Computer Graphics 2024/2025
Università degli Studi di Milano

Mesh Processing: algoritmi e strutture dati (cenni)



154

Mesh Processing: le basi algoritmche

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

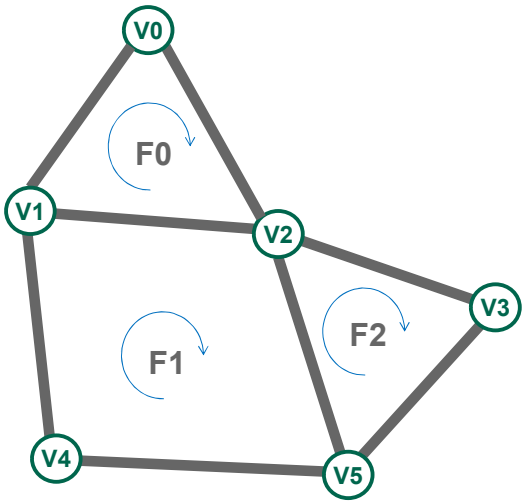
156

Strutture dati per Mesh Processing

- ✓ La struttura «lista facce» della mesh indexed, usata per memorizzare la connettività, non consente di effettuare queste operazioni in modo efficace
  - ⇒ (se non attraverso una scansione dell'intero vettore delle facce, che richiede ovviamente un tempo lineare col numero di facce)
  - ⇒ (eccetto una: «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
  - ⇒ che consentono di «navigare» sulla mesh molto più agevolmente
- ✓ Vediamo una di queste strutture

157

Struttura dati per la connettività: lista facce

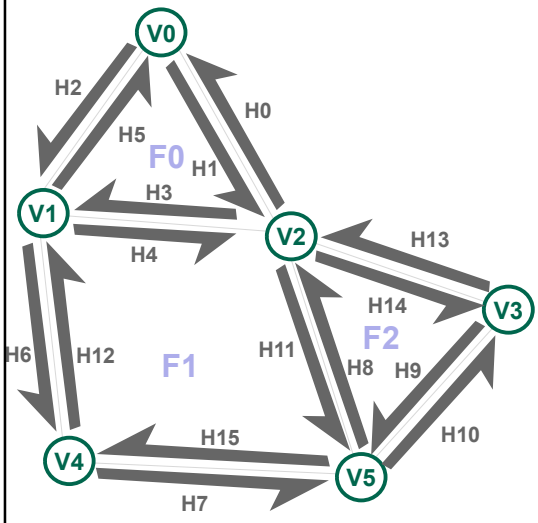


$F_0 \rightarrow$ 0 , 2 , 1
$F_1 \rightarrow$ 1 , 2 , 5 , 4
$F_2 \rightarrow$ 2 , 3 , 5

CONNETTIVITA'
(come lista facce)

158

Lista di half-edge



	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

159

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 faccie
    (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 3xN o 4xN (comodo)

164

Lista di half-edge

- ✓ Idea: posso rappresentare la connettività di una mesh come un Vettore di Half-Edge
- ✓ Un half-edge è un edge orientato
  - ⇒ E' detto half-edge perché rappresenta "la metà di un edge": ogni edge della mesh è composto da due half-edge
- ✓ Un half-edge è una struttura dati che include campi:
  - ⇒ Indice di Vertice: da quale vertice parte quell'half-edge
  - ⇒ 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-hege 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)
  - ⇒ Se ho una lista di facce, per ogni faccia memorizzo l'indice di un half-edge appartenente a quella faccia (uno qualsiasi)

165

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…

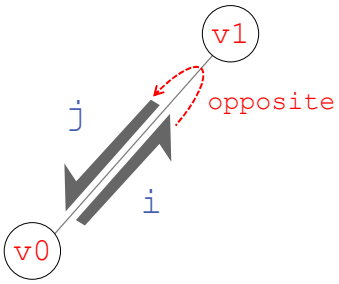
```
he[7].fi;
```

166

Esempio di uso base

- ✓ dato un halfedge di indice `i`,
  quali sono i due vertici `v0` e `v1` che delimitano l'edge corrispondente?

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



167

Strutture dati per connettività a confronto

- ✓ Lista di half hedge:
  - ⇒ Prolissa
    (calcola: 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

169

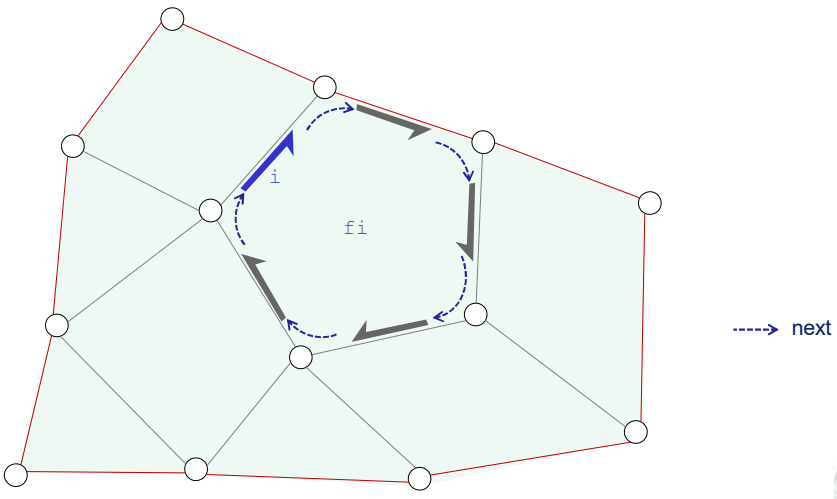
Strutture per connettività a confronto: sommario

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, o pure quad-meshes	✓ poligoni misti a piacere
✓ Compatta: 3 indici x tri	✓ Prolissa: 12 indici per tri
✓ Maggiormente adatta per rendering	✓ Meggiormente adatta per geometry processing

170

Esempio: conta quanti lati ha una faccia

✓ Dato un half-edge di indice `i` (adiacente ad una faccia `fi`), trova quanti `lati` ha questa faccia



171

Esempio: conta quanti lati ha una faccia

✓ Dato un half-edge di indice `i` (adiacente ad una faccia `fi`), trova quanti `lati` ha questa faccia

(era un half edge di bordo)

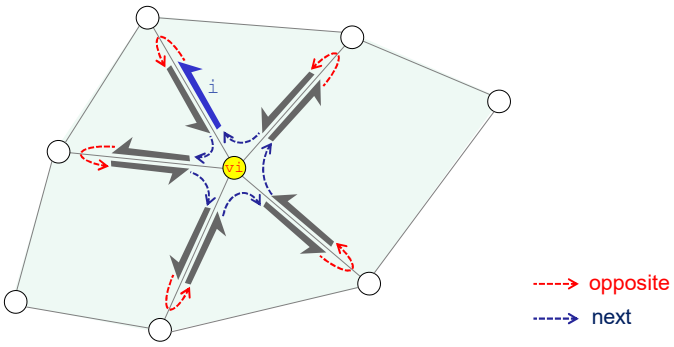
```
int fi = he[i].fi;
if (fi == -1) … /* non esiste la faccia */
int start = i; // half edge di partenza
int lati = 0;
do {
   lati ++;
   i = he[i].next;
} while (i!=start);
```

(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 `vi` , trova tutti i vertici `vj` nella stella di `vi`



(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 `vi` , trova tutti i vertici `vj` nella stella di `vi`

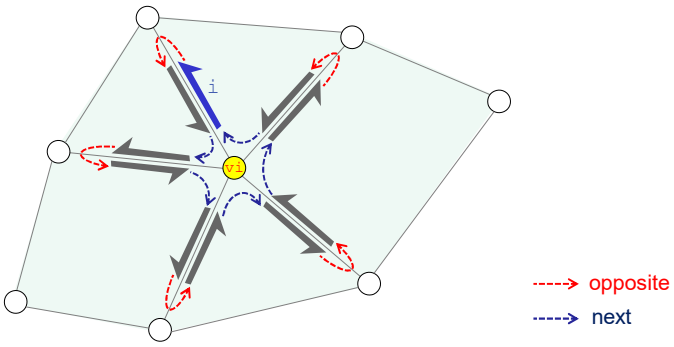
```
int vi = he[i].vi;

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 `vi` , trova tutte le faccie `fj` adiacenti a `vi`



(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 `vi` , trova tutte le faccie `fj` adiacenti a `vi`

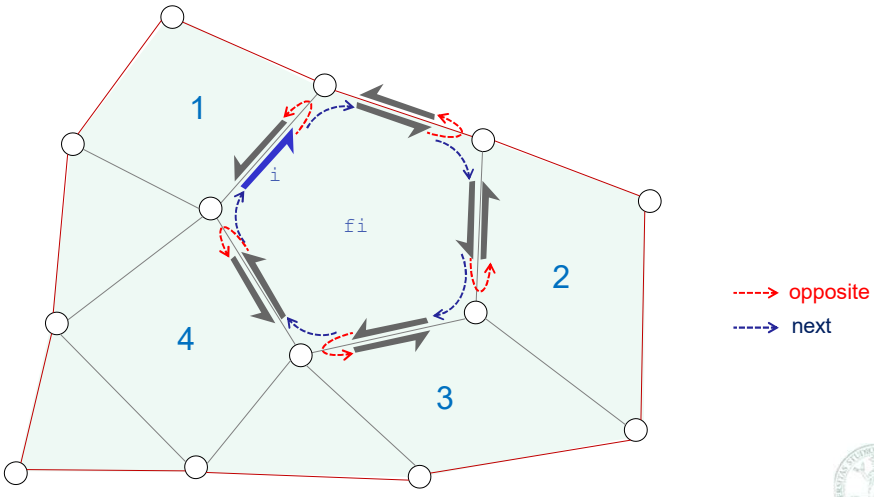
```
int vi = he[i].v;

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 `fi`, allora conta `quante_facce` adiacenti a `fi` ci sono



177

Esempio: contare la faccie adiacenti da una faccia

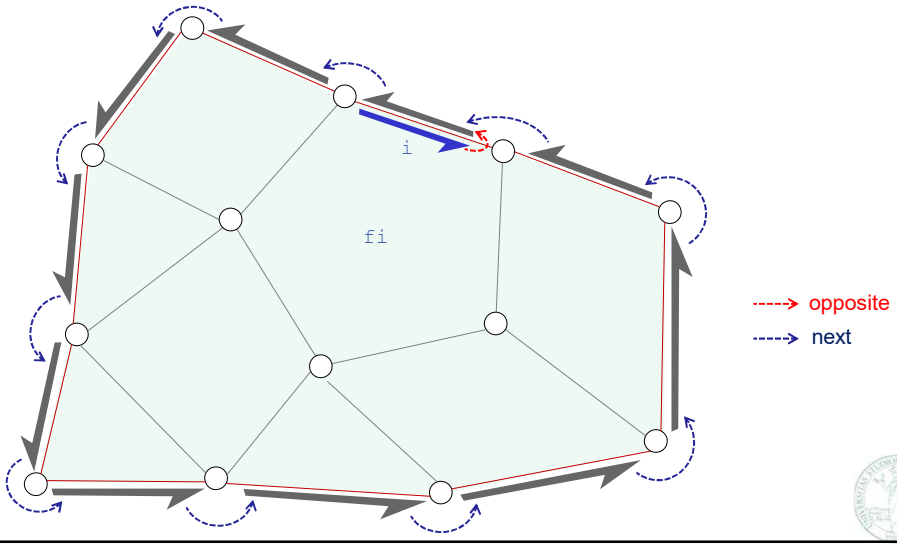
- ✓ dato un indice di halfedge `i`, se confina con una faccia `fi`, allora conta `quante_facce` adiacenti a `fi` ci sono

```
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 `fi`; se l'edge corrispondente è di bordo, allora scandici tutti gli edge che fanno parte dello stesso bordo:



179

Esempio: visita di un bordo (insieme di edge)

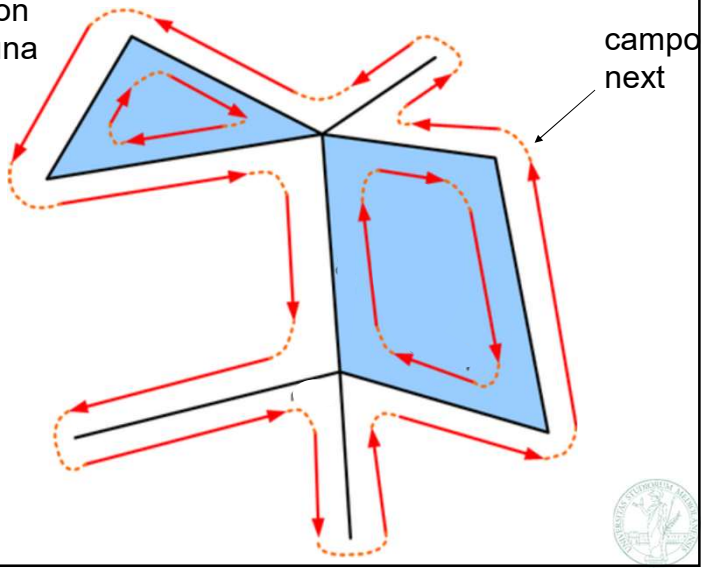
✓ Sia dato un indice di halfedge `i`, confinante con una faccia `fi`;
se l'edge corrispondente è di bordo, allora scandici tutti gli edge che fanno parte dello stesso bordo:

```
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)



181