Marco Tarini - Computer Graphics 2025/2026
Università degli Studi di Milano

## 3D Models – Mesh Poligonali: normali come attributi

38

## Mesh indicizzata: come classe (qui: in C++)

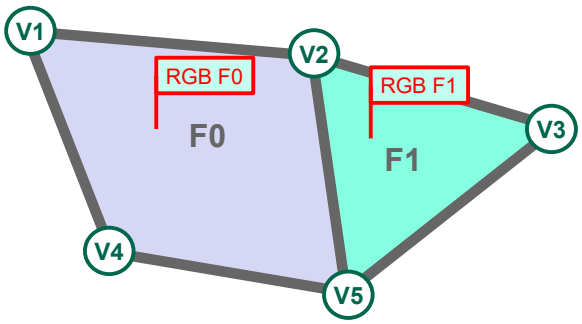
```
class Vertex {
  vec3 pos;
  rgb color;   /* attribute 1 */
};

class Face{
   int vertexIndex[3];
   vec3 normal; /* attribute 2 */
};

class Mesh{
  vector<Vertex> vert; /* geom + attr */
  vector<Face> face;   /* connettivita' */
};
```

Esempio di una variante:
le normali sono attributi memorizzati per faccia,
e la mesh è tri-mesh (3 indici di vertice per faccia)

40

## Mesh: attributi

- ✓ Quantità che variano sulla superficie
  - ⇒ es: colore RGB
  - ⇒ definiti per faccia:
    rappresentano valori costanti su ogni faccia

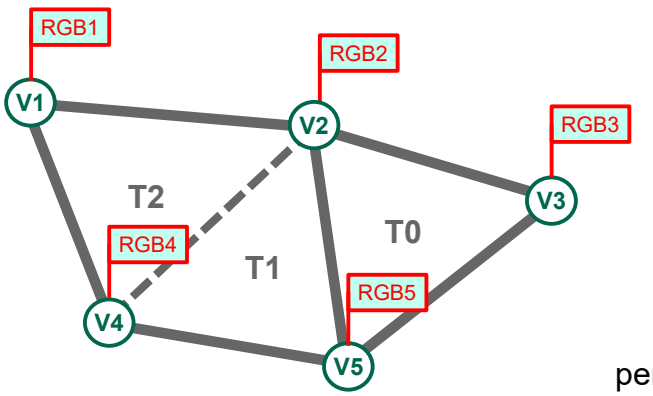



per faccia

41

## Mesh: attributi per vertice (mesh simpliciale)

- ✓ L'interpolazione degli attributi definiti sui vertici è definita dentro a facce *triangolari*
  - ⇒ Stiamo per vedere come, nella track matematica del corso
  - ⇒ (Quindi, per prima cosa, le facce che sono poligoni non triangolare devono essere suddivise in triangoli)




per vertice

42

## Interpolazione degli attributi per vertice dentro ai triangoli

- ✓ Se gli attributi sono definiti **per vertice** (il caso più comune) allora sono implicitamente interpolati dentro le facce
- ✓ Ad ogni punto **q** in un triangolo **T** viene implicitamente attribuita un'interpolazione di dei tre attributi definiti sui vertici di **T** , usando, come peso dell'interpolazione, le **coordinate baricentriche** di **q** in **T**
  - ⇒ Gli attributi, definiti esplicitamente solo sui vertici, sono così estesi implicitamente su tutta la superficie
  - ⇒ Nota: questo è definito solo su facce triangolari: facce poligonali devono essere scomposte in triangoli
- ✓ GPU support:
  - ⇒ La GPU è specializzata nei task quali il computo delle coordinate baricentriche di punti dentro ai triangoli, e la conseguente l'interpolazione degli attributi definiti sui vertici
  - ⇒ Durante il rendering, questo procedimento viene effettuato per ogni pixel coperto da un triangolo
  - ⇒ (nota: i pixel corrispondono a punti interni delle facce)

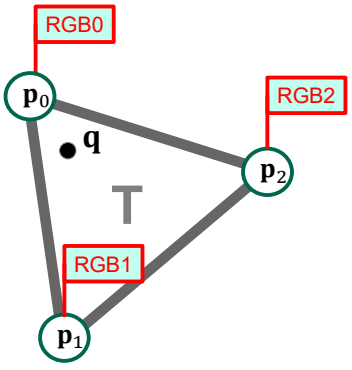
43

## Interpolazione degli attributi per vertice dentro ai triangoli

Esempio:

- ✓ Un triangolo **T** ha questi tre attributi colore assegnati ai suoi vertici $\mathbf{p}_0,\ \mathbf{p}_1,\ \mathbf{p}_2$
- ✓ Prendiamo un punto **q** su **T** di coordinate baricentriche $k_0, k_1, k_2$ in **T** , cioè con

$$\mathbf{q} = k_0\,\mathbf{p}_0 + k_1\,\mathbf{p}_1 + k_2\,\mathbf{p}_2$$



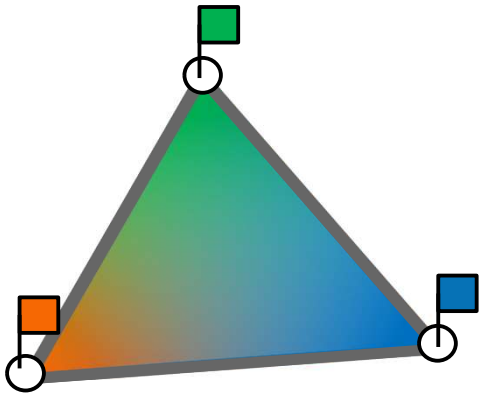

- ✓ Allora a **q** assegno il colore

$$k_0\ \boxed{\text{RGB0}} + k_1\ \boxed{\text{RGB1}} + k_2\ \boxed{\text{RGB2}}$$

per vertice

44

## Interpolazione degli attributi per vertice dentro ai triangoli



$$k_0\mathbf{p}_0 + k_1\,\mathbf{p}_1 + k_2\,\mathbf{p}_2$$

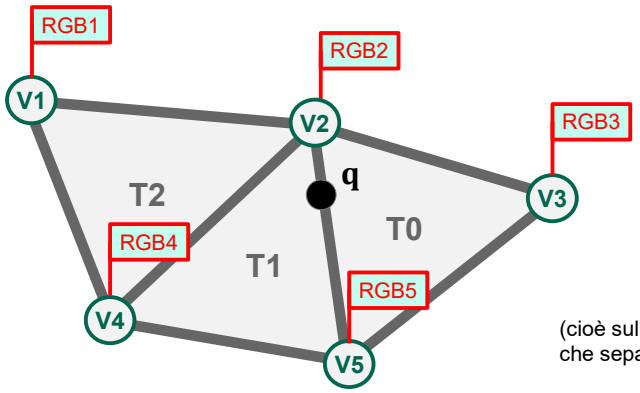
$k_0$ + $k_1$ + $k_2$

per vertice

45

## Gli attributi per vertice sono continui sulla mesh

- ✓ Gli attributi definiti per vertice (interpolati dentro le facce) sono per costruzione continui (C∞) dentro alle facce
- ✓ Anche fra coppie di facce adiacenti, abbiamo una continuità di attributo (C0)




(cioè sull'edge che separa T0 e T1)

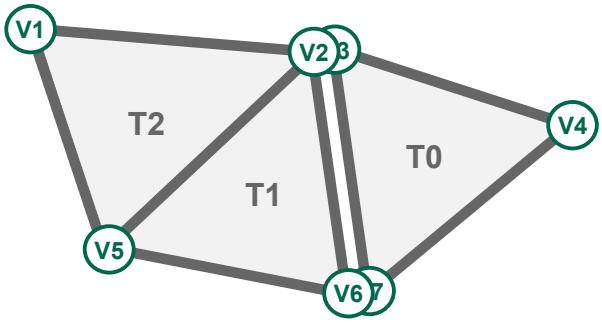
Infatti: si consideri un punto $\mathbf{q}$ a cavallo fra T0 e T1:

- ✓ quale attributo è associato a $\mathbf{q}$, se lo si considera parte di T0?
- ✓ quale attributo è associato a $\mathbf{q}$, se lo si considera parte di T1? (lo stesso!)

46

## Attributi per vertice con discontinuità

- ✓ Se è necessario, è possibile modellare delle discontinuità (C0) di attributo lungo un edge, duplicando i vertici ai suoi estremi
  - ⇒ La mesh avrà due vertici geometricamente coincidenti (stessa posizione) ma con attributi associati differenti
  - ⇒ Due edge tecnicamente aperti saranno quindi perfettamente sovrapposti, causando una superficie continua (senza «spiragli»)
  - ⇒ Facce adiacenti useranno una oppure l'altra di questa coppia di vertice




- ⇒ Questa configurazione si chiama un «seam» (letteralmente: cucitura) o «vertex seam»

47

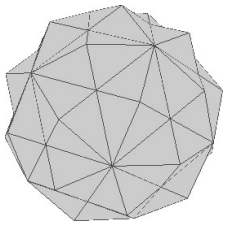
## Attributi: il caso delle normali

Nel caso in cui l'attributo è la normale
(cioè il *vettore unitario ortogonale alla superficie*)

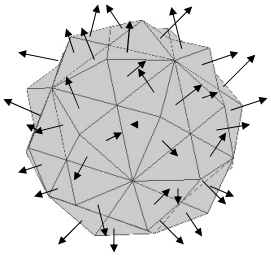
- ✓ se memorizzato per faccia:
  - ⇒ la normale costante su ciascuna faccia
  - ⇒ facce (dall'aspetto) piatto
  - ⇒ discontinuità C0 sugli edge:
    edge sono «spigoli taglienti»,
    detti edge di crease, o «hard» edges
- ✓ se memorizzato per vertice
  - ⇒ normale che varia sulla faccia
  - ⇒ facce (dall'aspetto) curvo, in generale
  - ⇒ continuità C0: → aspetto «smooth» (liscio),
  - ⇒ (ma non continuità C1)

48

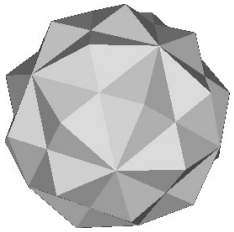
## Normale come attributo per faccia



mesh di partenza



per ogni faccia,
calcolo normale

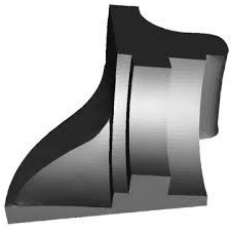


rendering
risultante
(detto "**flat shading**")

49

## Normali di una superficie: osservazioni

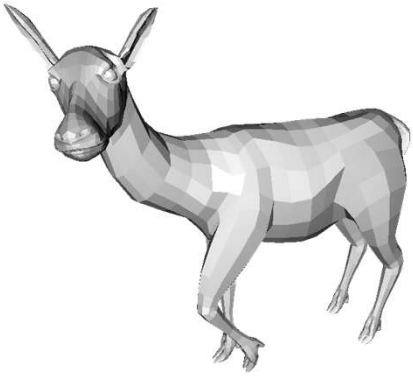
- ✓ Normale costante => superficie *piatta*
- ✓ Normale che varia con continuità => superficie *curva*
- ✓ Normale che varia con discontinuità => un ***crease***
  - ⇒ un *crease* è uno spigolo vivo, una linea dove la normale è discontinua
  - ⇒ anche detto ***hard-edge***, o ***feature line***



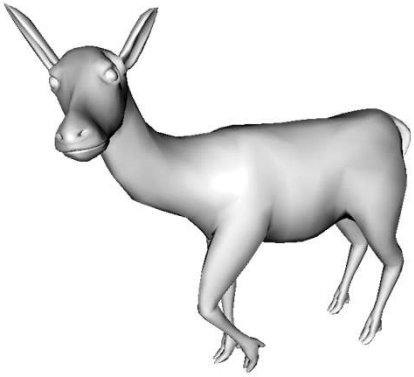
50

## Normali come attributi

Normali per faccia



Normali per vertice



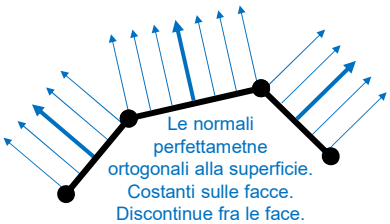
Nota: le normali sono rivelate all'occhio dall'*illuminazione*
(l'interazione fra superficie e luce, computata durante il rendering)

51

Superfice "effettiva"
della mesh, e normali utlizzate:

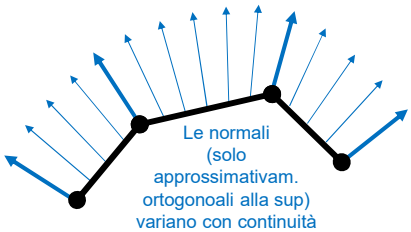
Appare come:

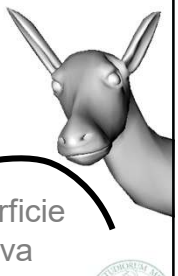
Flat shading
(normali per faccia)




Superficie
piatta a tratti

Smooth shading
(normali per vertice)



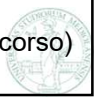



Superficie
curva

52

## Normali come attributi per vertice di una mesh

- ✓ Normali definite per faccia:
  - ⇒ Le normali sono costanti sulle facce:
  - ⇒ Le facce appaiono piatte
  - ⇒ coerentemente col modello digitale, ma a differenza (spesso) dell'oggetto 3D che si intendeva rappresentare!
  - ⇒ Appaiono piccoli crease su ogni edge della mesh
- ✓ Normali definite per vertice:
  - ⇒ Le normali variano sulle facce (vengono interpolate dentro le facce, come qualsiasi altro attributo)
  - ⇒ Le facce appaiono in generale curve
  - ⇒ Come ottengo una faccia che appaia piatta? Uso una stessa normale come attributo dei tre vertici di un triangolo
  - ⇒ Come ottengo un crease (una discontinuità di normale)? (vedi next)
- ✓ Nota: le normali risultano visibili all'osservatore attraverso l'illuminazione del modello digitale
  - ⇒ Il calcolo dell'illuninazione è un task del rendering (2nda metà del corso)



54

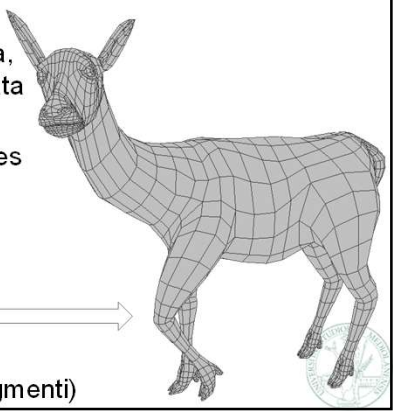
## Interpolare l'attributo per vertice "vettore normale"

- ✓ Come abbiamo visto, interpolando vettori unitari si ottengono vettori non unitari
  - ⇒ Che vanno quindi ri-normalizzati
- ✓ Questo significa che gli attributi di tipo *vettore unitario*, come le normali, devono essere rinormalizzati dopo ogni l'interpolazione
- ✓ Esercizio: determinare con una stima se questo comporta un onere di calcolo eccessivo, per una GPU, durante un rendering in real-time
- ✓ Traccia
  - ⇒ stimare quante operazioni in virgola mobile (Floating-point Operation) sono necessarie per normalizzare un vettore (soluzione: circa 10)
  - ⇒ stimare quante volte sarà necessario ri-normalizzare un attributo «normale» in un rendering (stimiamo, a braccio: 1 volta in ogni pixel, quindi circa 1000x1000 = 10^6)
  - ⇒ frame per secondo, in un rendering in tempo reale: stimiamo 60
  - ⇒ totale **FL**oating-point **OP**eration per **S**econdo (FLOPS) necessari a questo passaggio: 10 x 10^6 x 60 = 6x10^8 = 0.6 Giga-FLOPS
  - ⇒ Una GPU in commercio è capace di erogare… Tera-FLOPS = migliaia di Giga-FLOPS

55

## Note su rendering delle mesh

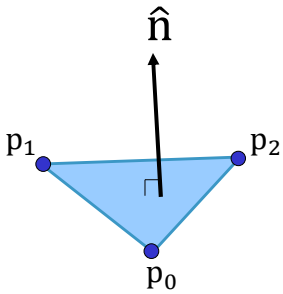
- ✓ L'uso di normali come attributo *per faccia* produce un tipo di rendering detto **flat shading**
  - ⇒ Perché le facce appaiono piatte (flat)
  - ⇒ Questo può essere utile per rivelare la forma poligonale delle mesh (ad esempio per poter giudicare la qualità della triangolazione)
- ✓ L'uso di normali come attributo *per vertice* produce un tipo di rendering detto **smooth shading**
  - ⇒ Perché la superficie appare liscia e curva, nascondendo la sua natura poligonalizzata
  - ⇒ Questo è di gran lunga in metodo più utilizzato, per esempio nei videogames
- ✓ Un tipo di rendering che rivela ancora più chiaramente la poligonalizzazione della mesh è detto **wireframe**
  - ⇒ Nel quale vengono disegnati esplicitamente anche gli edge (come segmenti)



56

## Computo della normale di un triangolo

(Cioè del suo orientamento nello spazio)
(Vedi lezione nella marte matematica sul cross product)




due cosiddetti "edge vectors"

$$\vec{n} = (p_1 - p_0) \times (p_2 - p_0)$$

"area vector" definito come un vettore ortogonale al triangolo, che è lungo quanto la (doppia) area del triangolo

normalizzazione

$$\hat{n} = \frac{\vec{n}}{\|\vec{n}\|}$$

"normale" del triangolo (un vett unitario)

E' anche la doppia area del triangolo

57

## Codice: computo della normale di una faccia

```
Mesh m; // m è la la mesh su cui lavoro

int fi = 12;  // indice di una faccia

int vi,vj,vk; // tre indici di vertice
vi = m.face[fi].vertexIndex[0];
vj = m.face[fi].vertexIndex[1];
vk = m.face[fi].vertexIndex[2];

vec3 p0 = m.vert[vi].pos;
vec3 p1 = m.vert[vj].pos;
vec3 p2 = m.vert[vk].pos;

m.face[fi].normal = cross( p1-p0 , p2-p0 );

normalize(m.face[fi].normal);
```

- ✓ Esempio di pseudo-codice in C++ (vedi lucido a pagina 40 per la classe)
- ✓ Questo codice calcola la normale della faccia di indice 12.
- ✓ Incapsulando questo brano in un ciclo for su `fi`, calcolo le normali su tutte le faccie (con complessità lineare)

58

## Calcolo delle normali per faccia di una mesh (note)

- ✓ La normale delle facce triangolari è trovata nel modo visto
  - ⇒ ripetuto su ciascuna faccia (complessità lineare con la risoluzione)
- ✓ E' la normale «geometrica» della faccia
  - ⇒ l'orientamento del piano su cui giace il triangolo (costante sulla faccia) (nota: questo non è applicabile alle facce quads o poligonali – a meno che non siano planari)
  - ⇒ è definita dalla geometria della mesh
  - ⇒ Non è definita se ho triangoli «degeneri» (quelli con area 0) (per es, con 3 vertici allineati o 2 vertici coincidenti, etc). ESERCIZIO: cosa fa l'algo visto applicato ad una faccia degenere?
- ✓ La normale per faccia è orientata consistentemente (nello stesso verso) solo se la mesh è ben orientata
  - ⇒ cioè se i vertici nelle facce sono sempre in senso orario oppure antiorario (vedi dopo)
  - ⇒ normale verso l'interno oppure l'esterno? Dipende della formula che uso per il suo computo (quali due edge-vector scelgo, e in che ordine li moltiplico con cross)

59

## Le normali **per vertice** di una mesh (note)

- ✓ Le normali per vertice di una mesh (necessarie per lo smooth shading) non sono definite in modo univoco per via geometrica
  - ⇒ dobbiamo in pratica «inventarcene» una
- ✓ La domanda da porsi è:
  - ⇒ «se la mia mesh (che è composta da facce *piatte*) modella invece una superficie reale che invece è *curva*, quale è la normale di questa superficie curva, nella posizione di questo vertice?»
  - ⇒ La risposta dipende da quale superficie si intende rappresentare!
- ✓ Cioè: *le normali (per vertice) fanno parte del modello* (sono parte della descrizione della superficie) e spesso vengono costruite insieme al modello.
  - ⇒ Solo nei casi di una mesh sprovvista di normali, sarà necessario computarle per via geometrica
- ✓ Strategia spesso utilizzata in pratica:
  - ⇒ usare come normale di un vertice è dato dalla media (o una qualche interpolazione) delle normali delle facce adiacenti (NB: normalizzata)

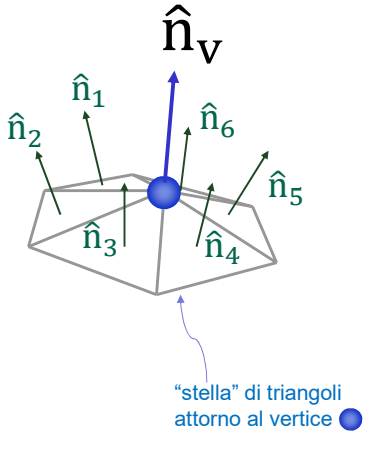
60

## Normali come attributo per vertice

Un modo per definire la normale di un vertice condiviso da $k$ triangoli è: la media delle $k$ normali definite sui triangoli

$k$ è detta la valenza del vertice, (nel disegno, $k$ = 6)

$$\hat{n}_v = \frac{\hat{n}_1 + \cdots + \hat{n}_6}{\|\hat{n}_1 + \cdots + \hat{n}_6\|}$$




Nota: dato che si normalizza comunque il risultato della sommatoria, non c'è necessità di dividere per il numero di elementi sommati, come si fa di solito per la media

61

## Algoritmo 1 per computo di normale per vertice

Passo 1: computo delle normali per faccia (triangolare)

⇒ come visto

Passo 2: computo delle normali per vertice

✓ ∀ vertice **v**:
trovo tutte le facce adiacenti a **v**,
(cioè tutte le facce che annoverano l'indice di **v** fra i propri indici)
computo la somma di tutte le loro normali, e normalizzo il risultato
(ottenendo così la normale di **v**)

Problema: come trovo «tutte le facce adiacenti ad un vertice dato»?

✓ Se scandisco l'intero vettore della «lista-facce» in cerca di facce che contengano l'indice di **v**, il mio algoritmo diviene *quadratico*

⇒ se ho $n$ vertici e $2n$ facce, l'algoritmo 2 richiede O( $2n$^2 ) operazioni!

⇒ nel mesh processing, gli algoritmi quadratici non sono accettabili (perché il numero $n$ può essere molto grande)

Esiste invece un algoritmo efficiente (lineare)
che usa solo la struttura «lista-facce».

*Sapresti identificare questo algoritmo? (è nel prossimo lucido)*

62

## Algoritmo 2 per computo di normale per vertice

✓ Passo 1 ∀ vertice: azzero il suo vettore normale

⇒ Questo vettore servirà per cumulare le somme parziali

✓ Passo 2 ∀ faccia: calcolo la sua normale,
e la sommo alla normale di ciascun vertice ai suoi angoli

⇒ Alla fine di questo ciclo, su ogni vertice avrò accumulato un contributo da ciascuna faccia adiacente a quel vertice

✓ Passo 3 ∀ vertice: normalizzo il risultato

✓ La strategia generale dell'algoritmo 1 viene detta «*gathering*»

⇒ raccolgo («*gather*») su ogni vertice le normali delle facce adiacenti

✓ La strategia generale dell'algoritmo 2 viene detta «*scattering*»:

⇒ distribuisco («*scatter*») da ogni faccia la normale sui vertici adiacenti

✓ Anche se il risultato è lo stesso, l'algoritmo 2 è lineare:

⇒ Se ho $n$ vertici e $2n$ facce, mi ci vogliono solo $\sim n + 2n + n \in O(n)$ operazioni

⇒ In geometry processing, gli algoritmi lineari sono *feasible*: cioè sono efficienti anche per mesh a risoluzione molto grande (per es, $n$ = 10^9)

63