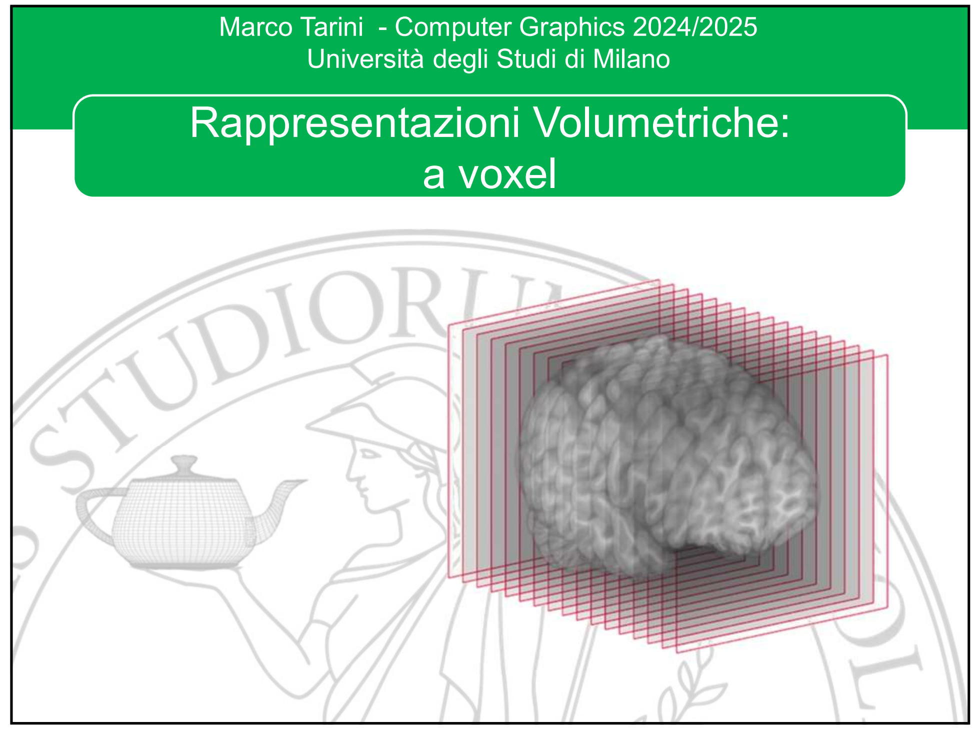


31

Una (imperfetta) categorizzazione dei tipi di modelli digitali 3D

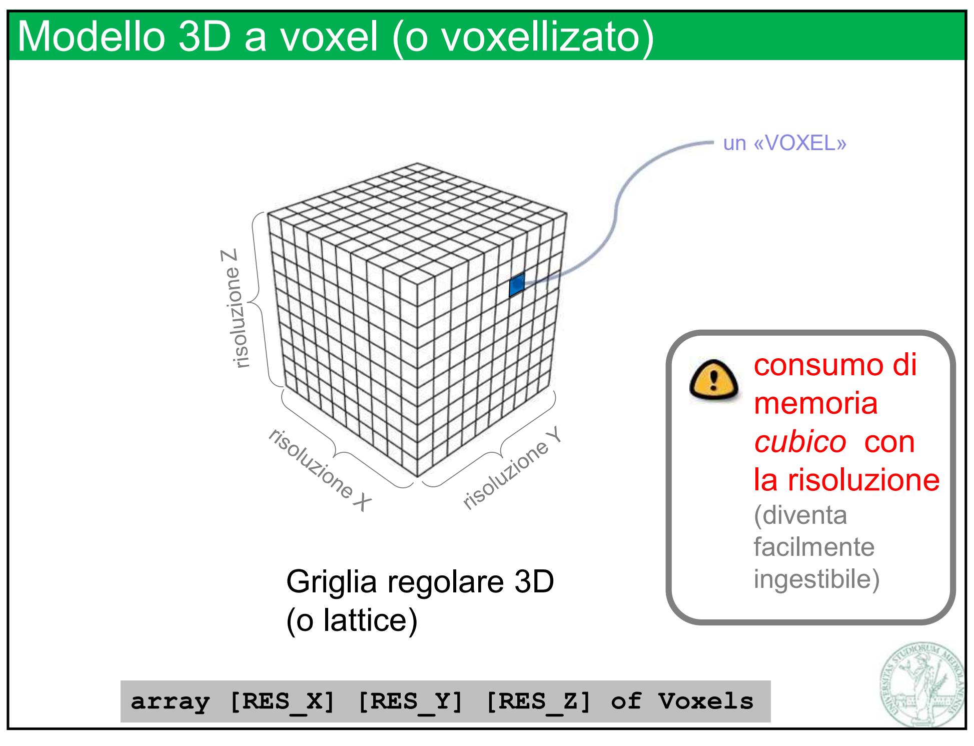
		ELEMENTI DISCRETI			CONTINUI
		regolari *«a griglia»*	semi-regolari o irregolari: elementi simpliciali	semi-regolari o irregolari: elementi non simpliciali	
SUPERFICIALI	2-manifold *«rappresenta una vera superficie»*	**Height Field** **Range Scan** **Geometry Images**	**Triangle Mesh**	**Polygonal Mesh** **Quad Mesh** **Quad dominant Mesh**	**Subdivision surfaces** **Parametric Surfaces (es. B-splines)**
SUPERFICIALI	non-manifold *«non rappresenta una sup»*	**Set di Range Scan**	**Point Cloud**		
VOLUMETRICI	(3-manifold)	**Voxelized Volume** **Volumetric Textures**	**Tetra Mesh**	**Hexa Mesh**	**Implicit models (es. CSG)**

32

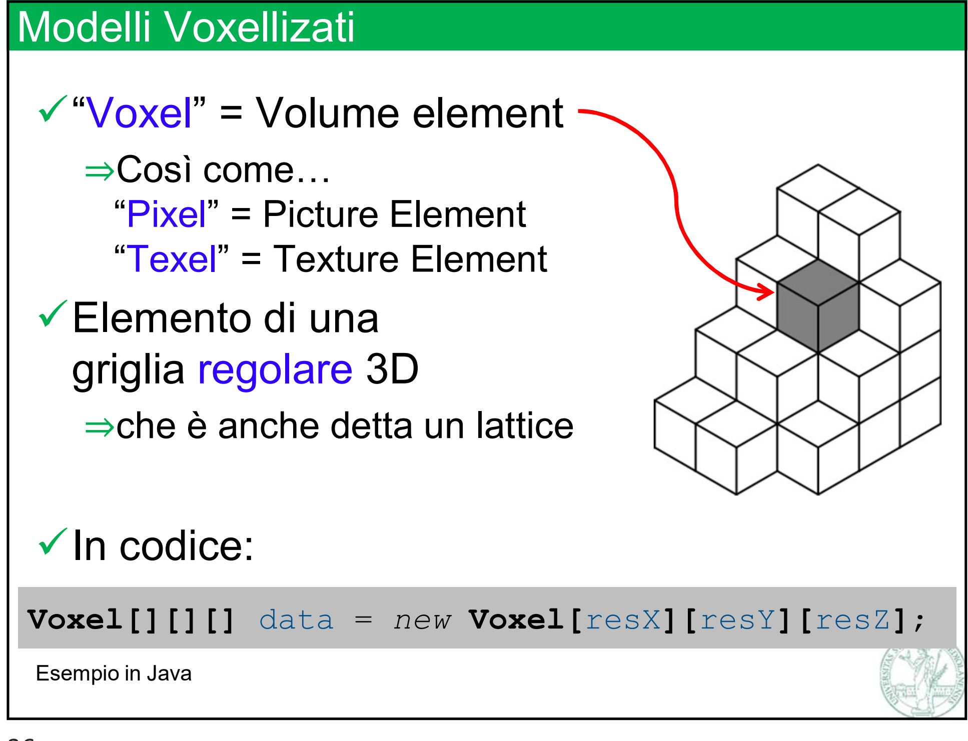
Modelli 3D Volumetrici

1. Discreti & irregolari: mesh poliedrali
   - ⇒analogo di una mehs poligonale, ma nel volume
   - ⇒insieme di poliedri adiacenti faccia a faccia
2. Discreti & regolari: dataset voxellizati
   - ⇒analogo di un immagine rasterizzata, ma in 3D
   - ⇒una griglia 3D regolare di voxel

34



35



36

In questo caso, 1 Voxel = 1 Boolean (1 bit)

ogni *voxel* è pieno (1) o vuoto (0)

37

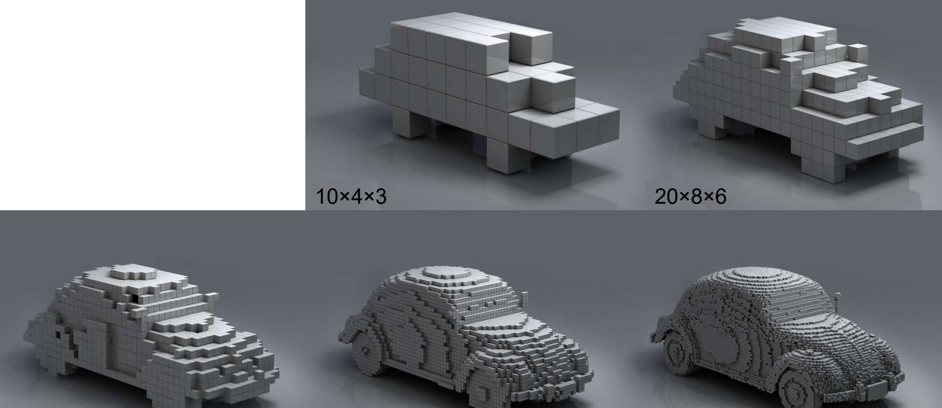
Occupazione spaziale dei dataset voxelizzati

- ✓ Lo spazio è cubico con la risoluzione (linare)
- ✓ E' di solito un prezzo troppo alto
  - ⇒Es: 1024^3 voxel = 1 gigavoxel
  - ⇒Molto oneroso, persino nel caso, come abbiamo visto fin'ora, di 1 solo bit per voxel (1 = pieno / 0 = vuoto)
  - ⇒Quando si memorizza 1 byte, 1 float, 1 double, 1 colore… etc, la situazione peggiora
- ✓ Detta la «curse of dimensionality»

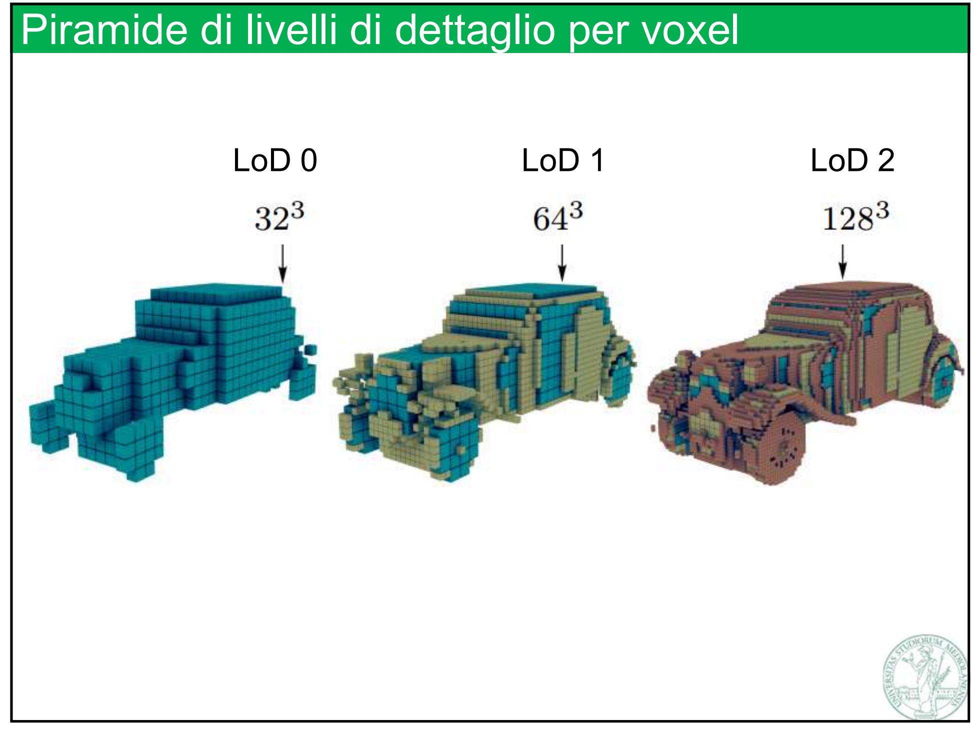
38

Risoluzione e costo in memoria

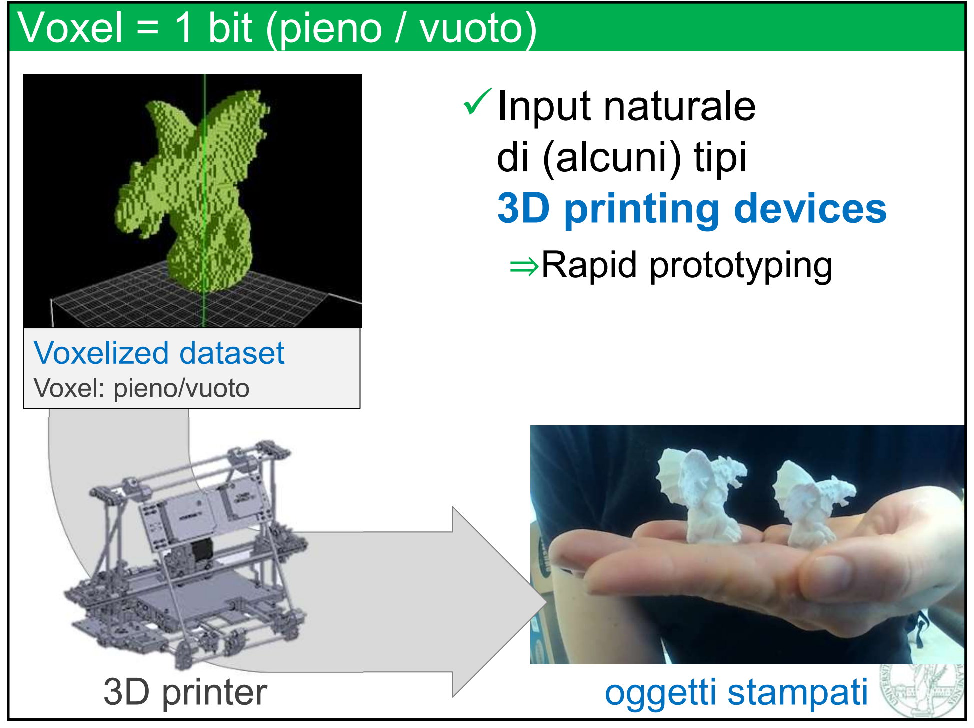
- ✓ risoluzione: un intero per lato res = (X,Y,Z)



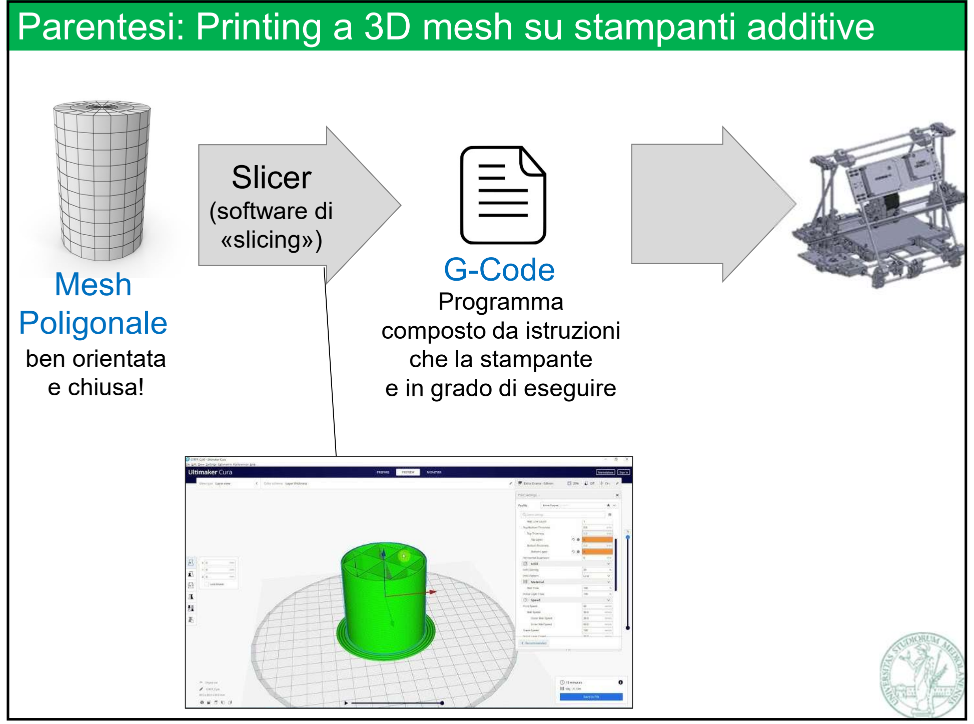
39



40



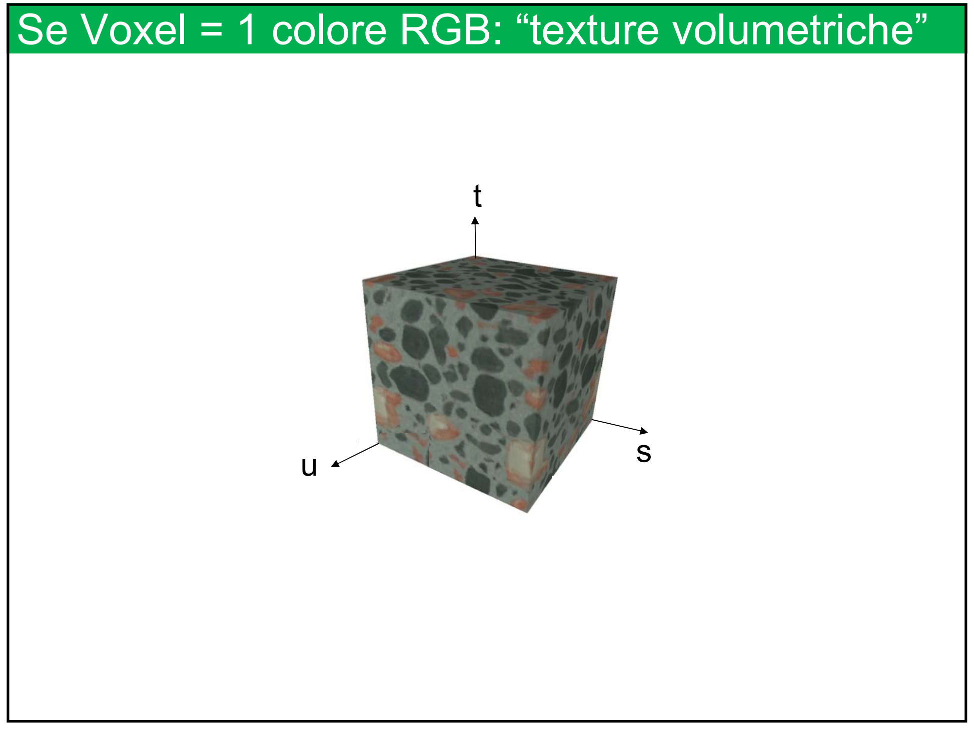
41



42

43

44



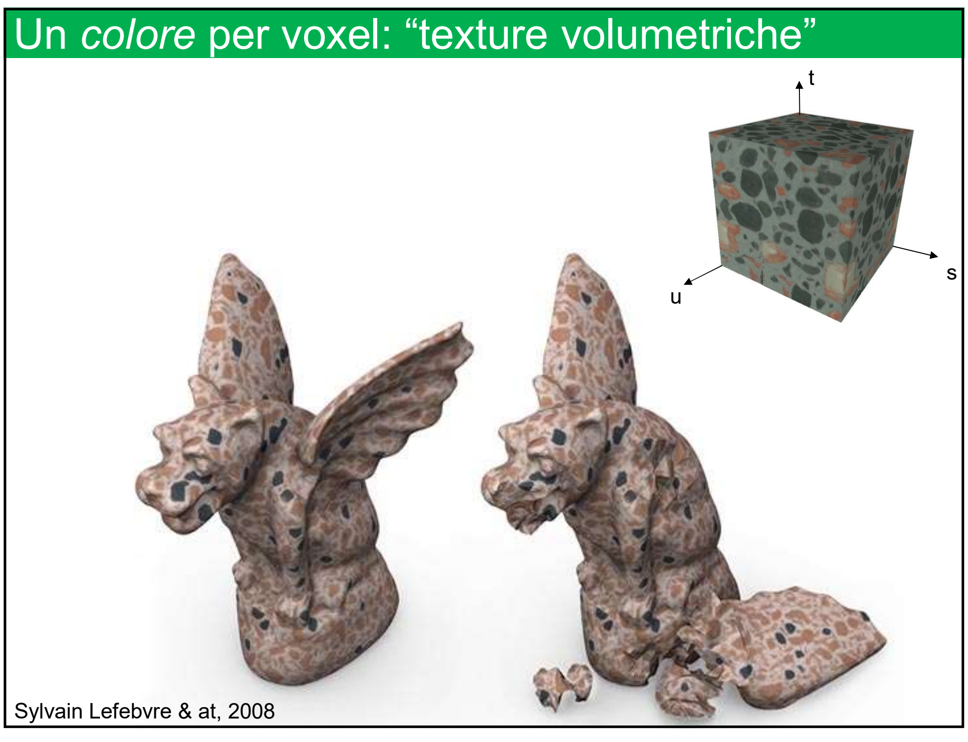
45

Volumetric Textures (o “solid Textures”)

- ✓ 1 texel = 1 voxel
  - ⇒ esempio, solid RGB textures: 1 texel = 1 RGB color
- ✓ E’ supportata dall’Hardware, come ogni altra tessitura:
  - ⇒ occupa la RAM della scheda video
  - ⇒ accesso HW accelerato durante il rendering
  - ⇒ interpolazione **tri**-lineare durante l’accesso …
- ✓ Modella il segnale (es. il colore) *dentro* al volume
  - ⇒ per es: come gli oggetti sono colorati all’interno
  - ⇒ utile per modelli che si possono rompere
  - ⇒ utile per pattern come legno, marmo…
- ✓ Non richiede alcuna parametrizzazione della superficie!
  - ⇒ La tessitura viene indicizzata dalle posizioni dei vertici
- ⚠ Solito problema, occupazione di memoria
  - ⇒ es: quanto per 1 tessitura 1024^3 8-bits-per-channel RGBA?
  - ⇒ es: quanto per 1 tessitura 265^3 8-bits-per-channel RGBA?

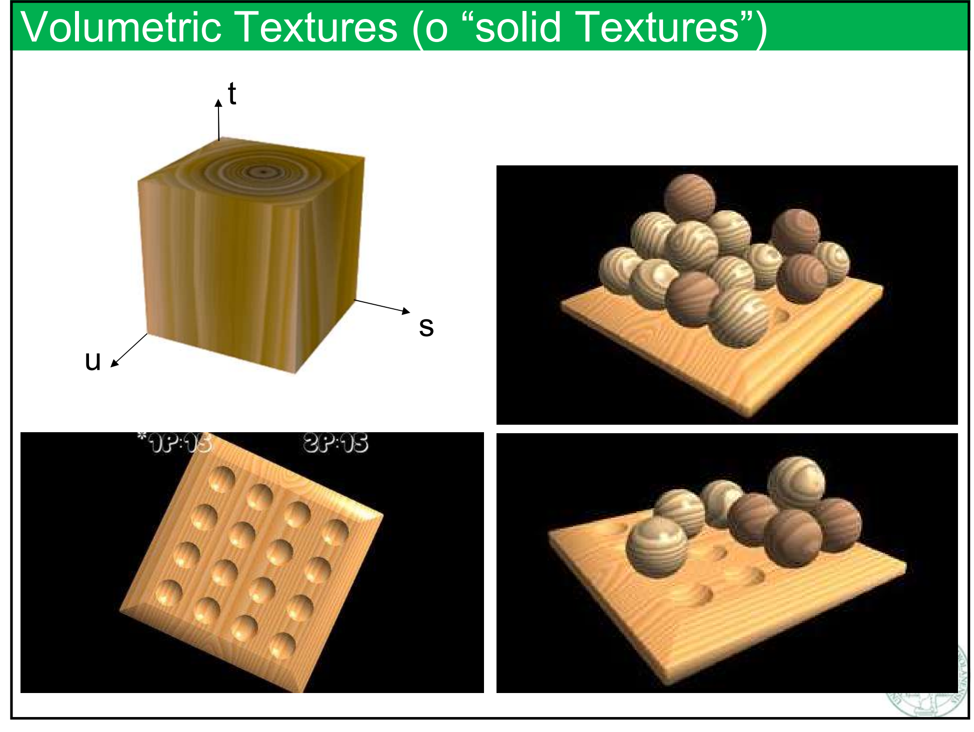
46

Un *colore* per voxel: “texture volumetriche”

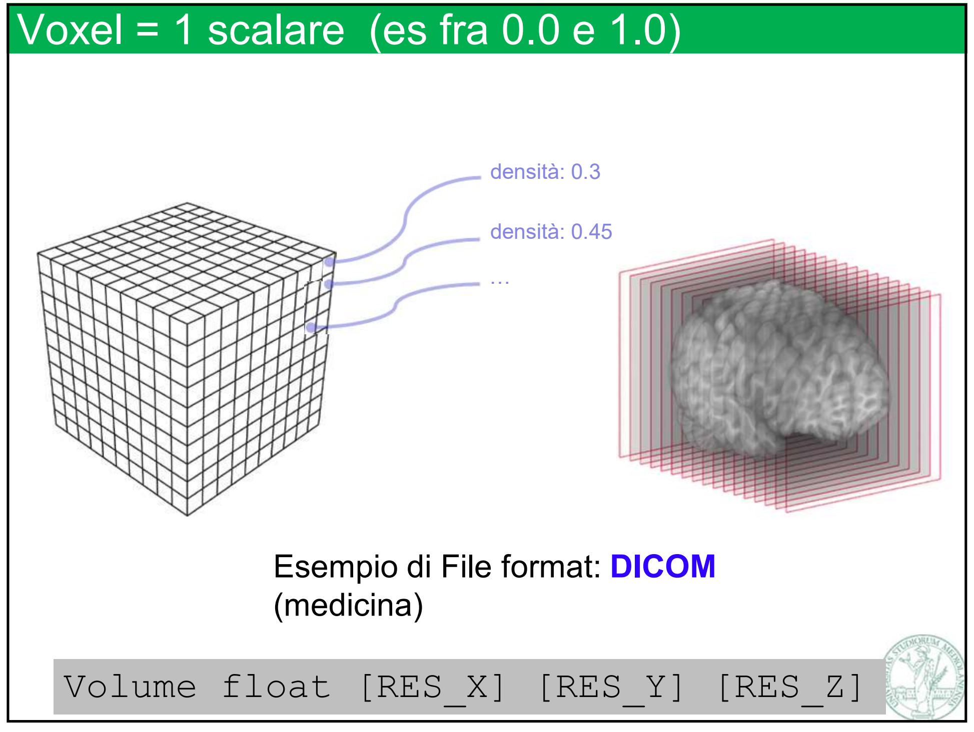


Sylvain Lefebvre & at, 2008

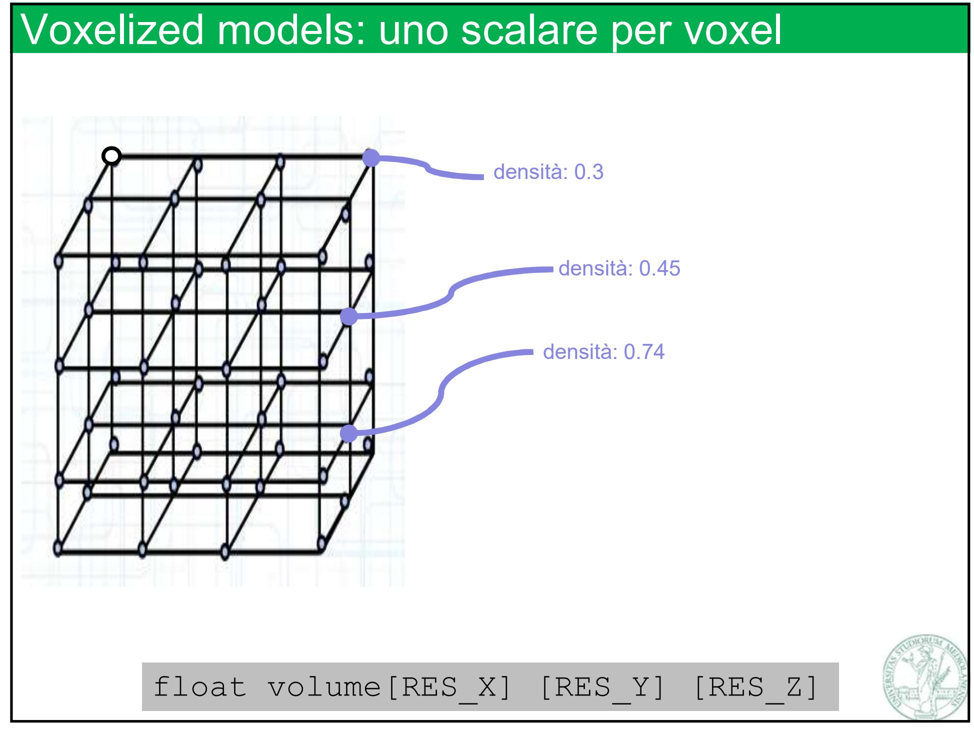
47



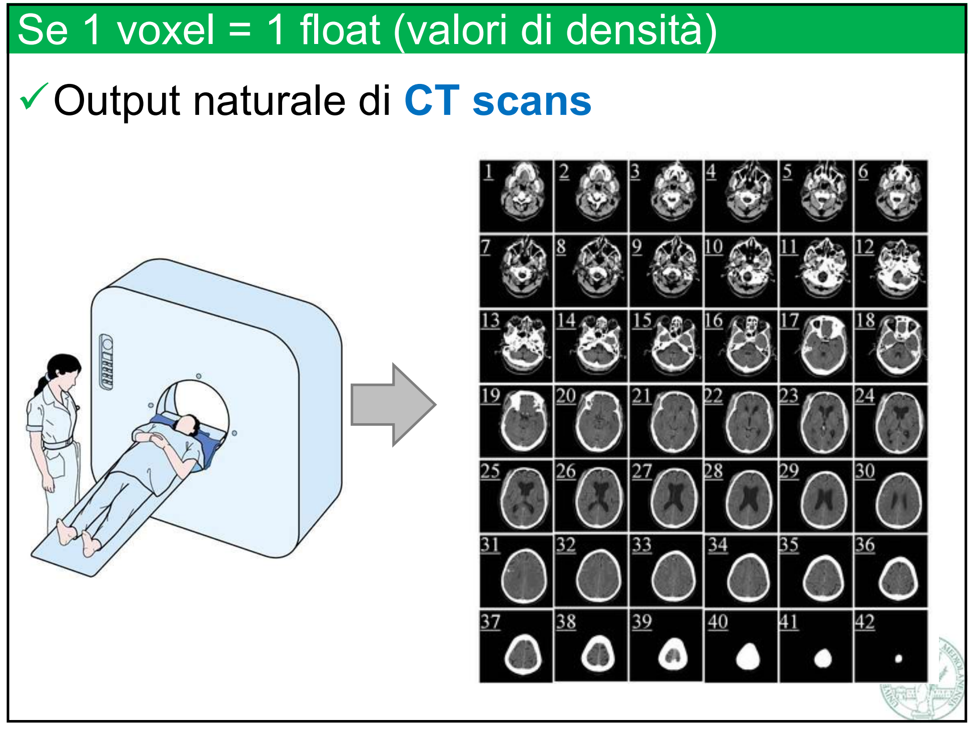
48



49

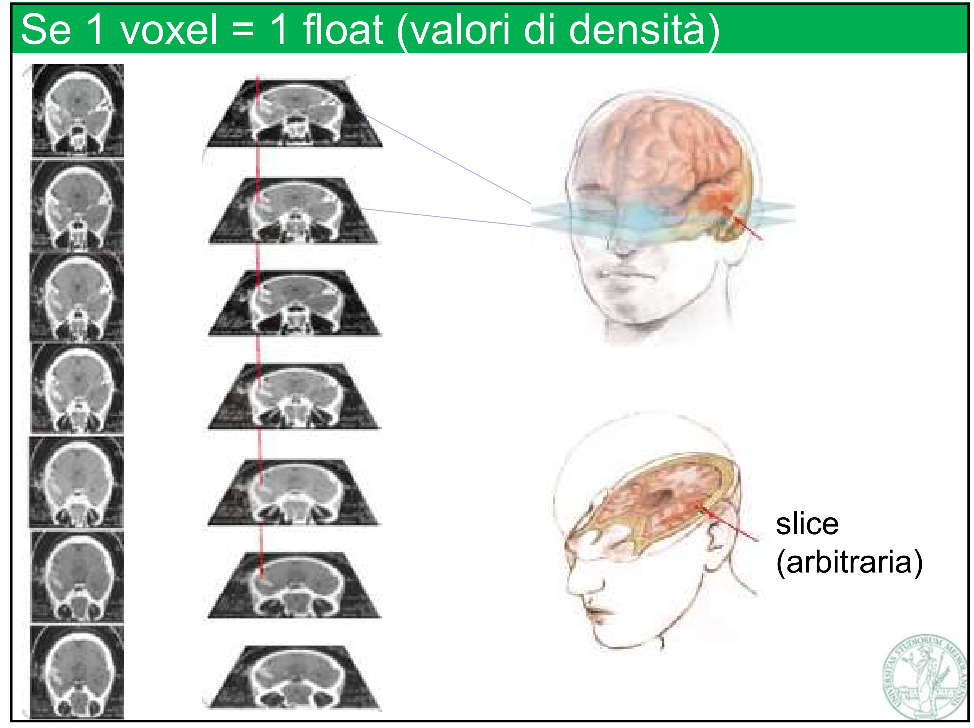


50



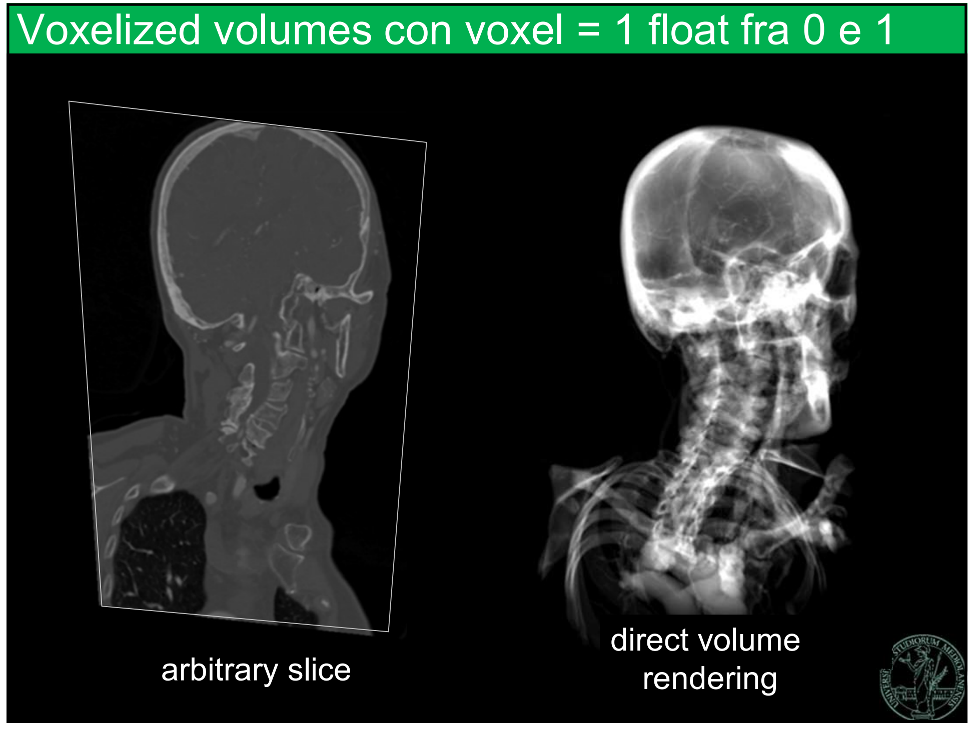
51

Se 1 voxel = 1 float (valori di densità)

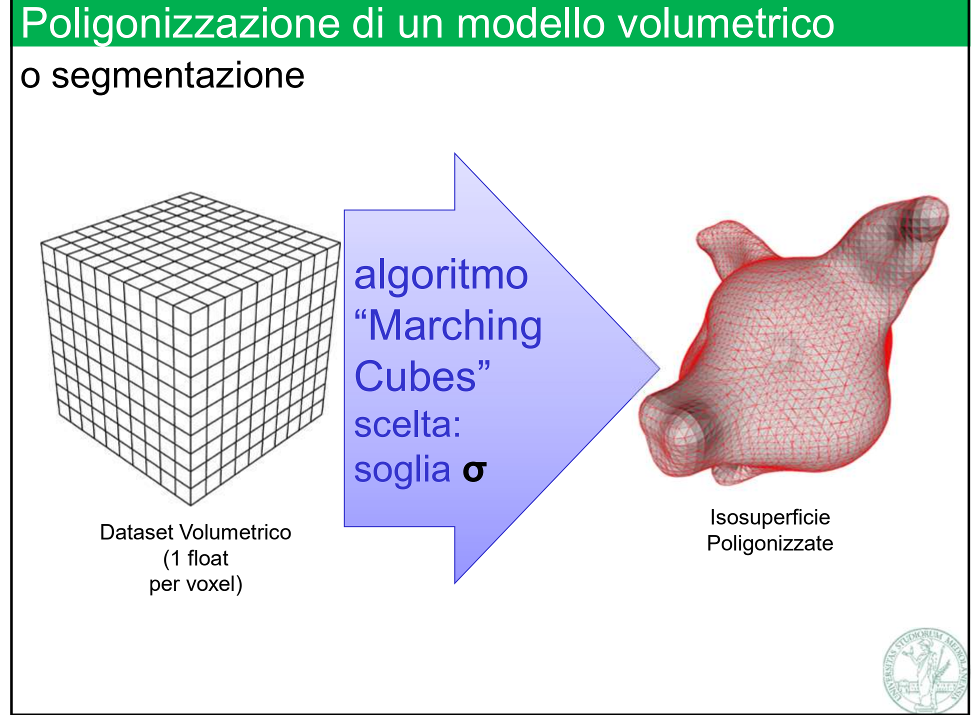


54

Voxelized volumes con voxel = 1 float fra 0 e 1



55



57

Isolinee e isosuperfici

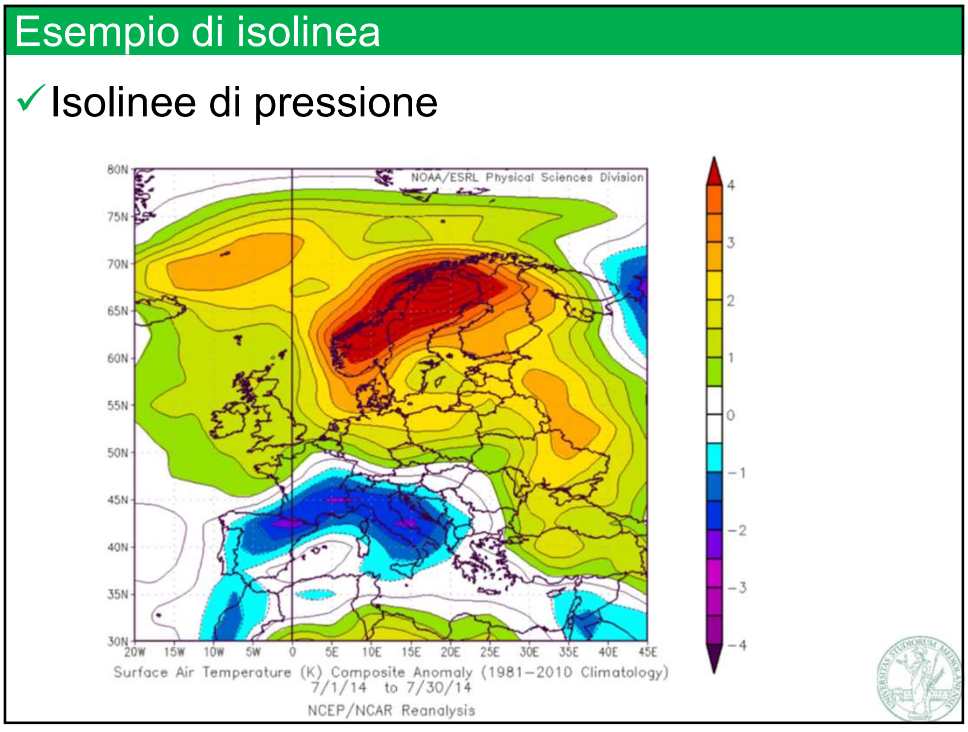
✓ Su un piano (2D):
- ⇒ogni punto del piano ha un valore scalare (esempio: pressione, o altezza – «height field»)
- ⇒prendo tutta la regione 2D con valore > **σ**
- ⇒il bordo di questa regione è una linea … i cui punti hanno valore tutti **σ** e che racchiude tutti i valori di valore > **σ**
- ⇒è la « **isolinea di valore σ** » (linea di isovalori)

✓ Su un volume (3D):
- ⇒ogni punto dello spazio ha un valore scalare (esempio: densità, pressione, temperatura…)
- ⇒prendo la regione 3D con valore > **σ**
- ⇒il bordo di questa regione è una superficie… i cui punti hanno tutti valore è **σ** che racchiude tutti i valori di valore > **σ**
- ⇒è la « iso-superficie di valore **σ** »

58

Esempio di isolinea

- ✓ Isolinee di pressione

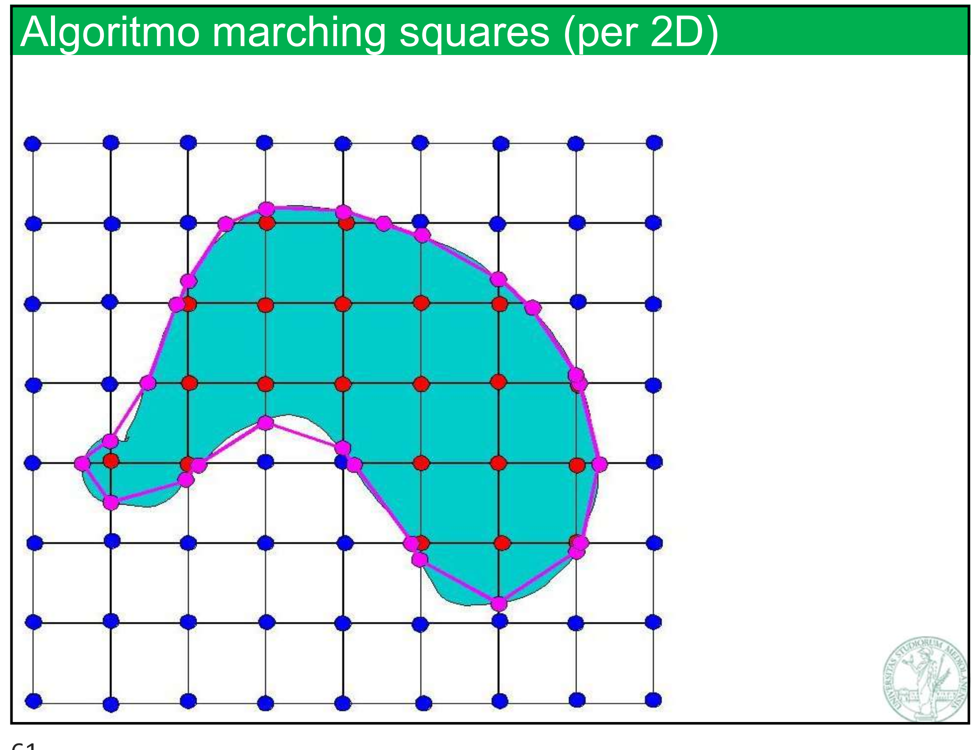


59

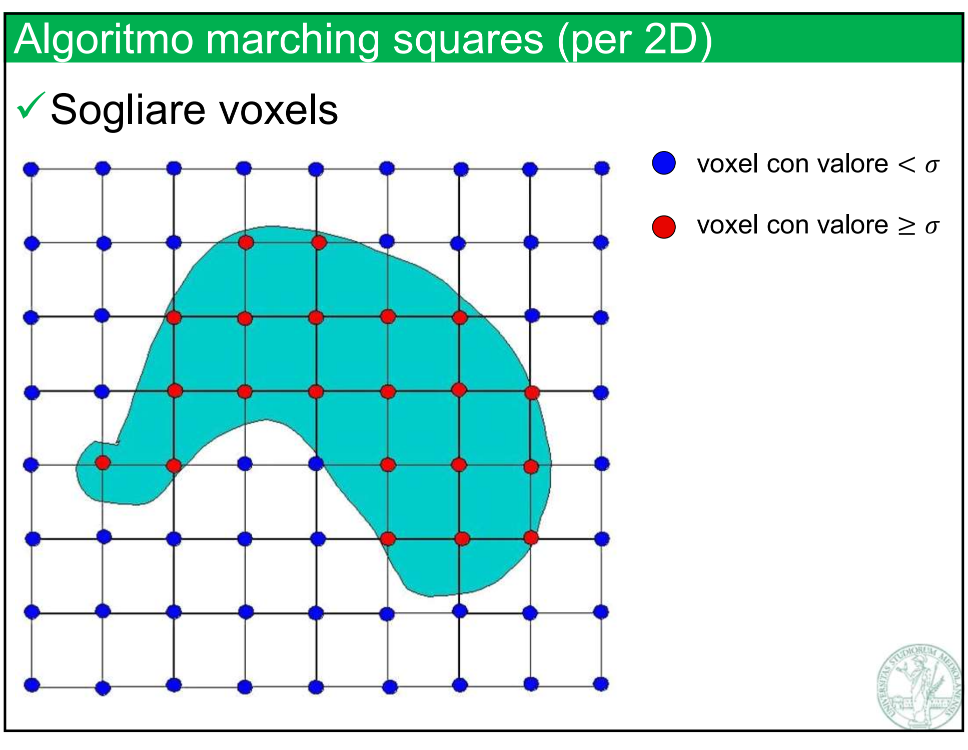
Poligonizzazione di un modello volumetrico

- ✓ Obiettivo:
  - ⇒ dato un modello volumetrico voxel (array 3D di voxel): un voxel = un valore scalare (per es, densità, temperatura…)
  - ⇒ produrre una tri-mesh (two-manifold, ben orientata, e chiusa) che racchiuda tutti i voxel di valore superiore ad una certo valore soglia σ
  - ⇒ Si tratta cioè di produrre una **iso-superficie** di valore σ: la superficie di tutti i valori continui nel volume che valgono esattamente σ
- ✓ Algoritmo: «**marching cubes**»
- ✓ Vediamo prima un analogo in 2D: algoritmo «**marching squares**»
  - ⇒ data un'immagine rasterizzata (array 2D di pixel) in cui un pixel = un valore scalare (per es, denistà, temperatura…)
  - ⇒ produce una **iso-linea** linea chiusa di valore σ (appossimata da segmenti) che racchiude tutti i pixel di valore superiore a σ

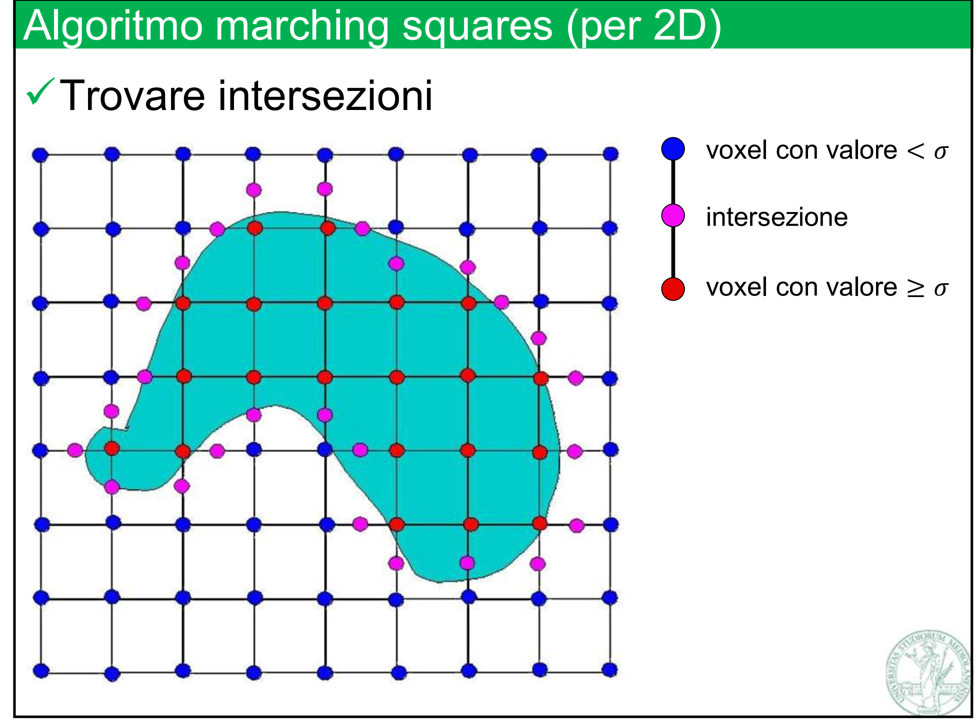
60



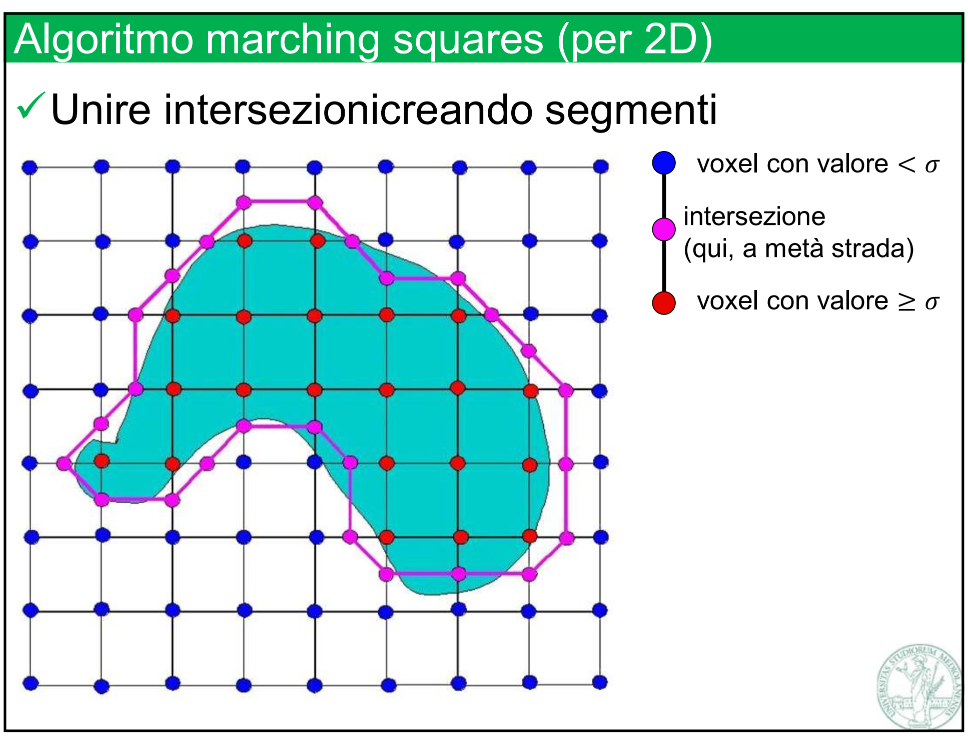
61



62



63

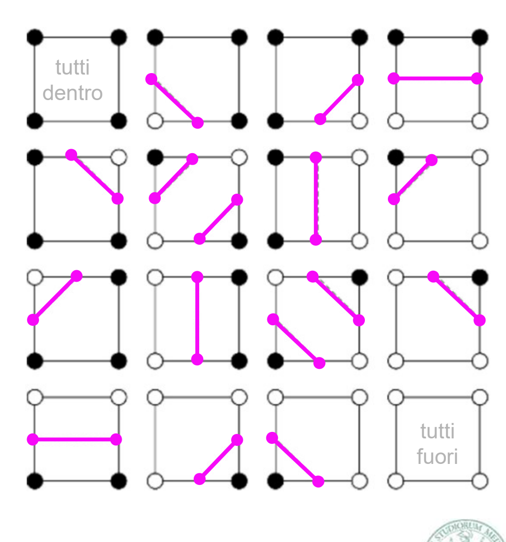


64

Algoritmo marching squares (per 2D)

Come trovare
i segmenti che connettono
le intersezioni?

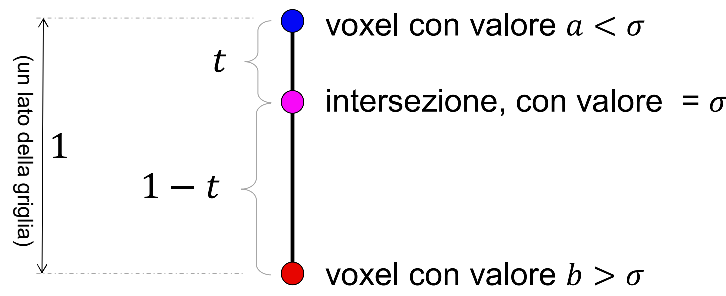
- ✓ Consideriamo un quadrato fra 4 voxel
- ✓ Ogni suo vertice è «dentro» $< \sigma$ o «fuori» $\geq \sigma$
- ✓ Per ogni combinazione, (e sono solo $2^4 = 16$) decido, una volta per tutte, i segmenti da costruire per unire le intersezioni
- ✓ ottengo questa tabella:



65

Algoritmo marching squares (per 2D)

- ✓ Come trovare le intersezioni



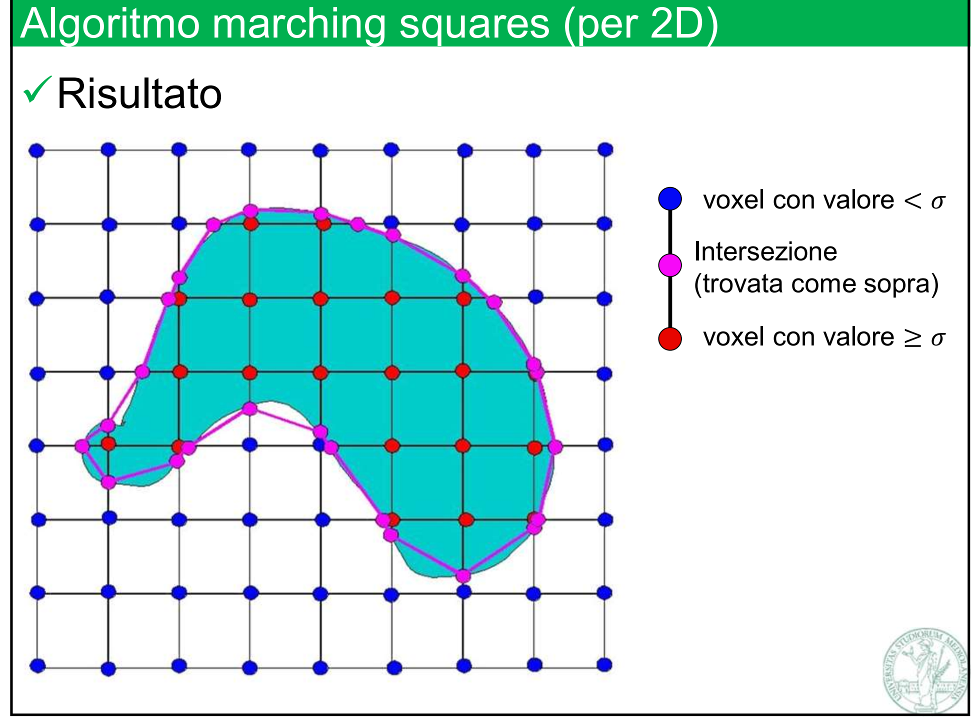
- ✓ Ipotesi: segnale interpolato linearmente:

$$a(1 - t) + b\,t = \sigma \quad \Leftrightarrow \quad t = \frac{\sigma - a}{b - a}$$

66

Algoritmo marching squares (per 2D)

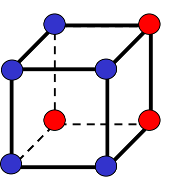
✓ Risultato



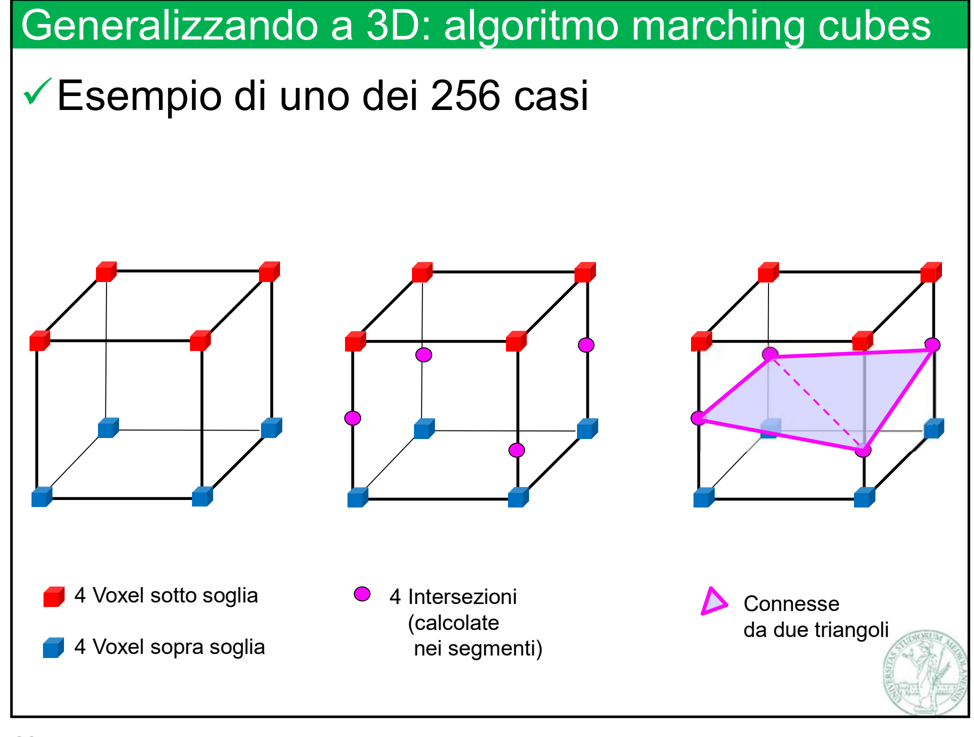
67

Generalizzando a 3D: algoritmo marching cubes

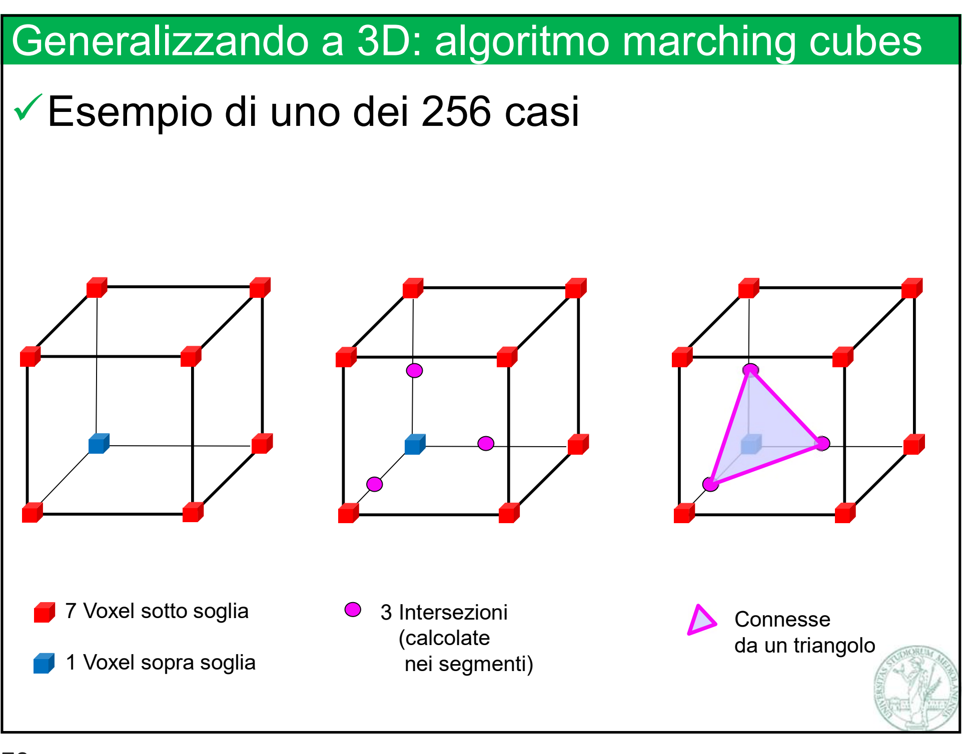
- ✓ Ogni voxel della griglia è dentro o fuori
  - ⇒ valore ≥ oppure < della soglia prescelta
- ✓ Ogni edge della griglia (orientato lungo la X, Y o Z), che connetta un vertice dentro ad uno fuori, ha una intersezione con l'isosuperficie cercata
  - ⇒ la si trova trovata come nel caso 2D
  - ⇒ per ciascuna intersezione, creo un vertice della mesh
  - ⇒ ho ottenuto la **geometria** della mesh!
- ✓ Come ottengo la **connettività**?
  - ⇒ Scompongo la griglia in cubetti 2x2x2
  - ⇒ Ogni cubo ha 8 vertici, ciascuno dei quali è dentro o fuori ➔ $2^8 = 256$ casi
  - ⇒ Uso una tabella per analizza, in ciascun caso, quali poligoni creare per connettere i vertici sui lati della griglia



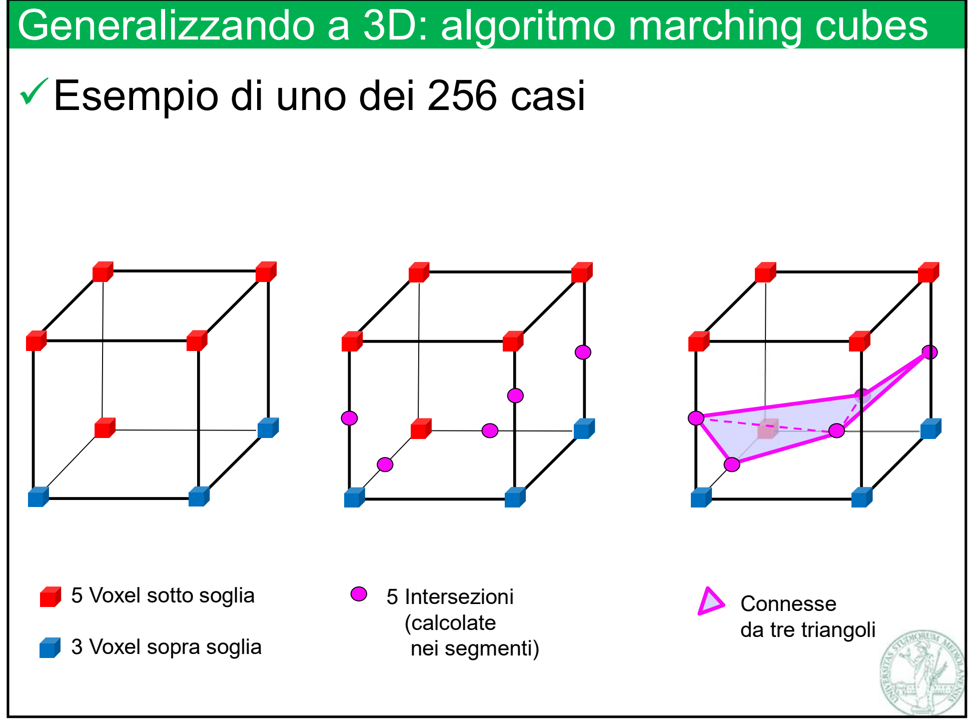
68



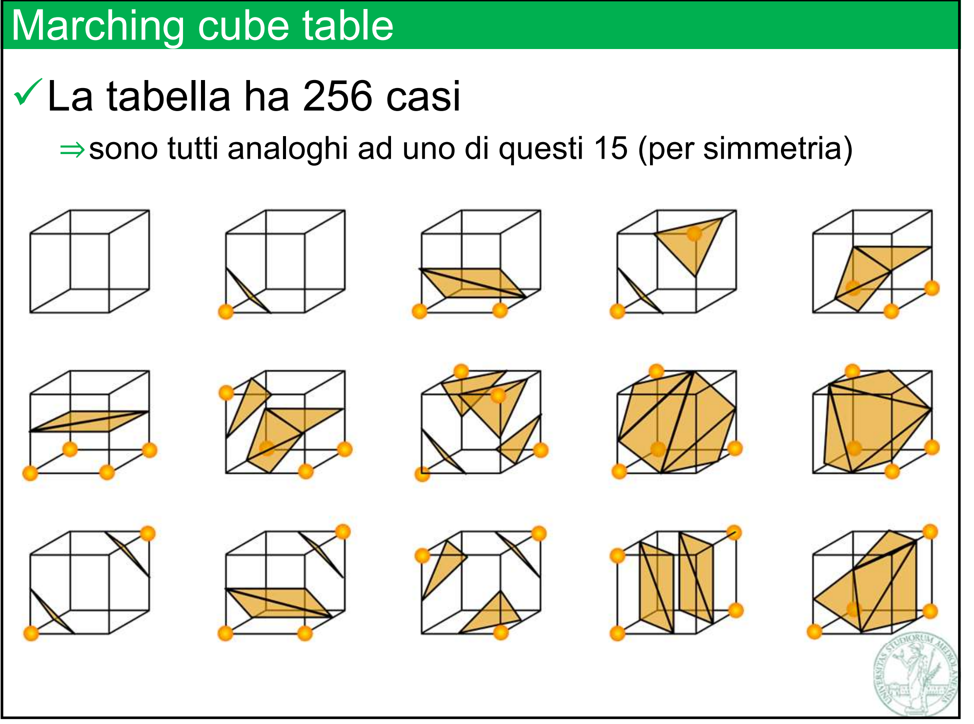
69



70



71



72

Algoritmo “Marching cubes”

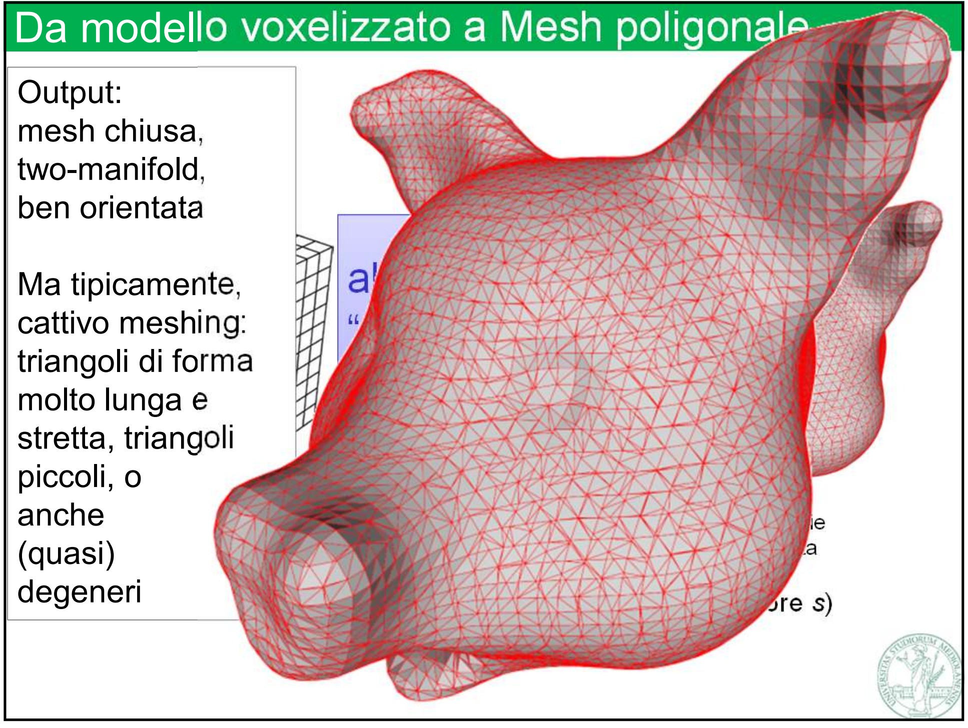
- ✓ Problema: efficienza.
  - ⇒ Curse of dimensionality: numero cubico di cubi da analizzare
- ✓ Soluzione possibile: evitare di processare il gran numero di cubi vuoti
  - ⇒ Molti dei cubi sono «tutti dentro» o «tutti fuori»
  - ⇒ Cioè non contengono né intersezioni sugli spigoli, né facce all’interno
- ✓ Idea: far «marciare» i cubi sull’isosuperficie:
  - ⇒ Trovo un primo cubo non vuoto → lo processo
  - ⇒ Passo a processare i cubi non vuoti vicini (che non siano già processati)
  - ⇒ Continuo fino ad aver completato la mesh connessa e chiusa
- ✓ L’algoritmo deve il suo nome a questo accorgimento, ma le potenze di calcolo dei moderni elaboratori consentono anche un approccio forza bruta:
  - ⇒ Processare una ad una *tutte* le celle cubiche

73

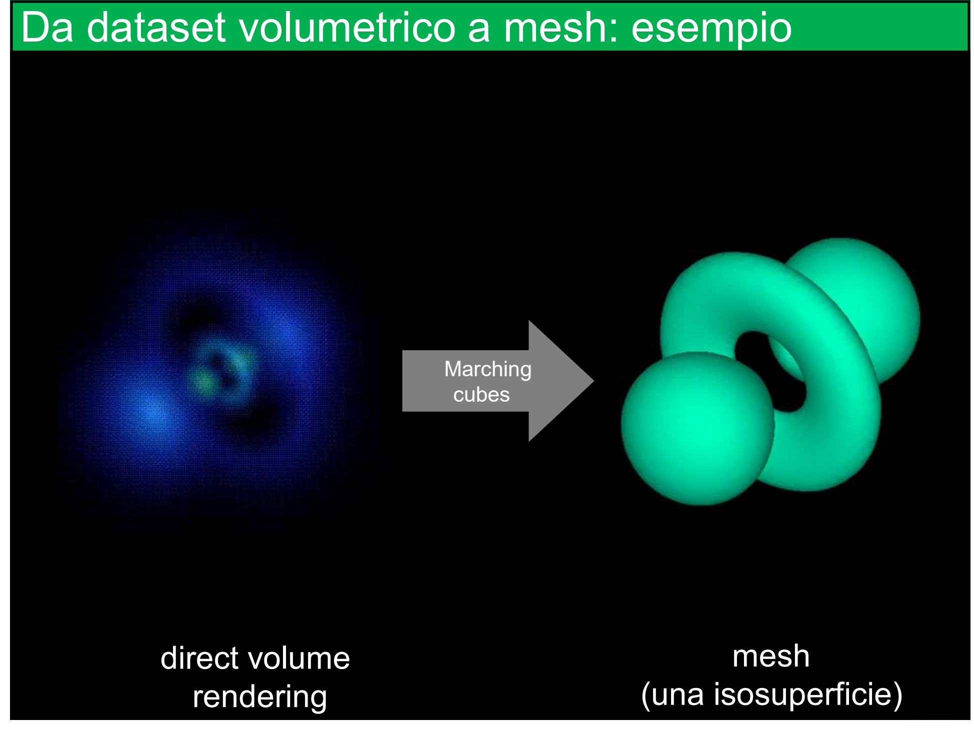
Da modello voxelizzato a Mesh poligonale

Output:
mesh chiusa,
two-manifold,
ben orientata

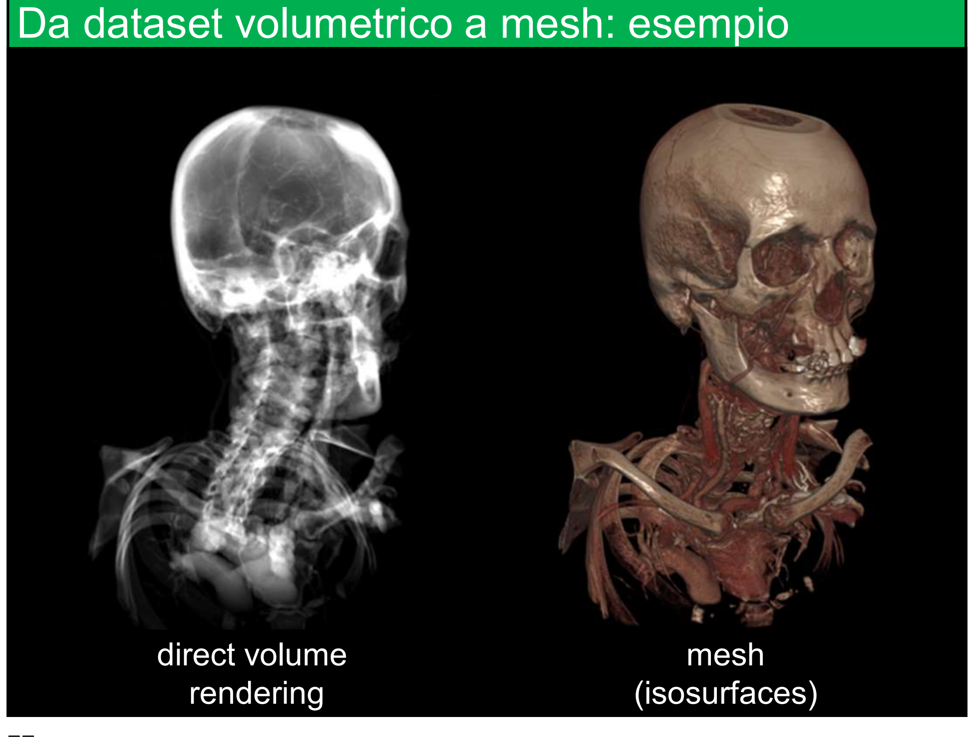
Ma tipicamente, cattivo meshing: triangoli di forma molto lunga e stretta, triangoli piccoli, o anche (quasi) degeneri



75



76



77