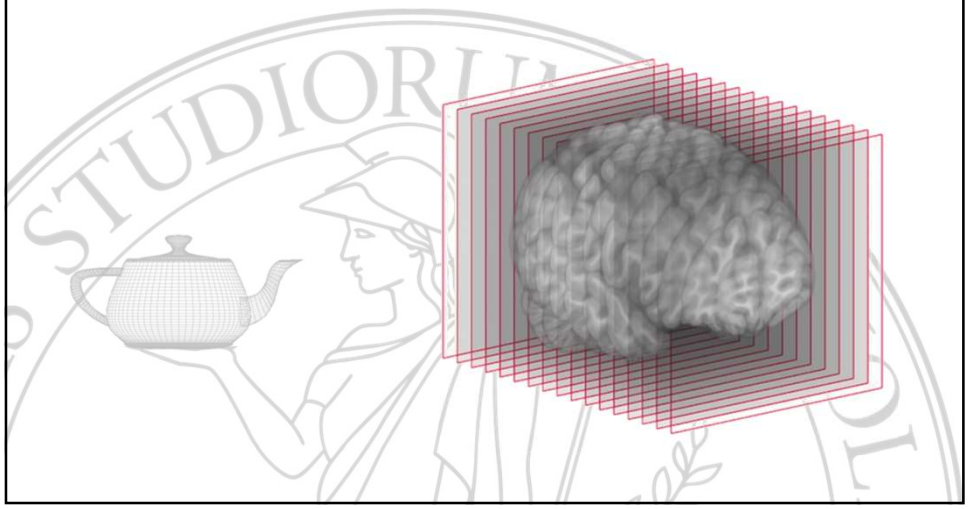
Marco Tarini - Computer Graphics 2025/2026
Università degli Studi di Milano

Rappresentazioni Volumetriche: a voxel



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)**
	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

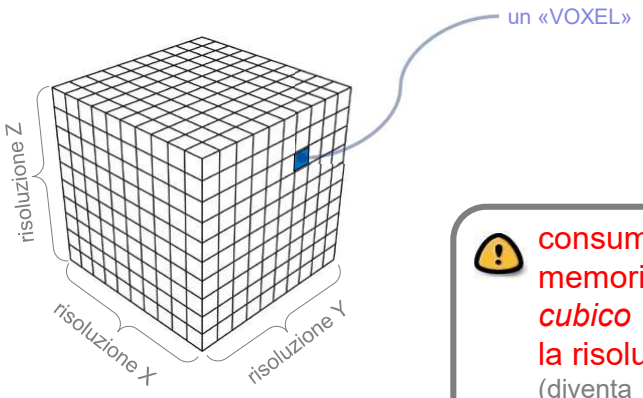
Modelli 3D Volumetrici

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

Modello 3D a voxel (o voxellizato)



Griglia regolare 3D
(o lattice)

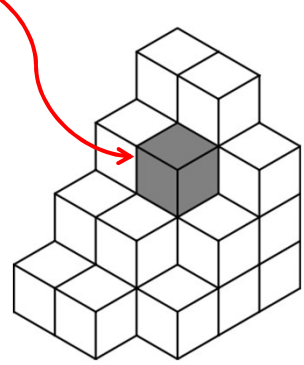
consumo di memoria *cubico* con la risoluzione (diventa facilmente ingestibile)

```
array [RES_X] [RES_Y] [RES_Z] of Voxels
```

35

Modelli Voxellizati

- ✓"Voxel" = Volume element
  - ⇒Così come…
    "Pixel" = Picture Element
    "Texel" = Texture Element
- ✓Elemento di una griglia regolare 3D
  - ⇒che è anche detta un lattice



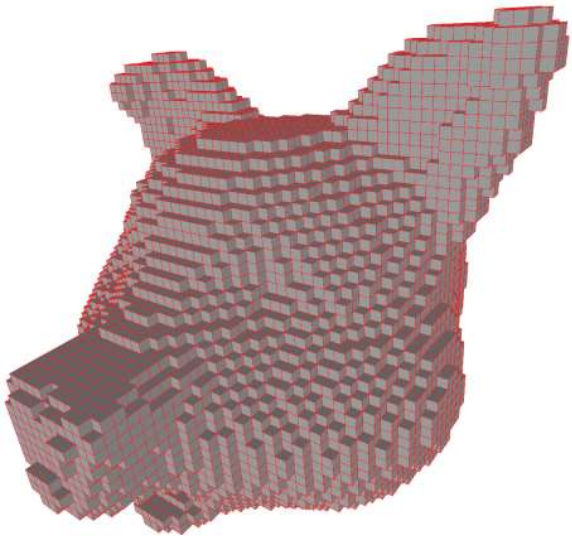
- ✓In codice:

```
Voxel[][][] data = new Voxel[resX][resY][resZ];
```

Esempio in Java

36

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



ogni *voxel* è pieno (1) o vuoto (0)

37

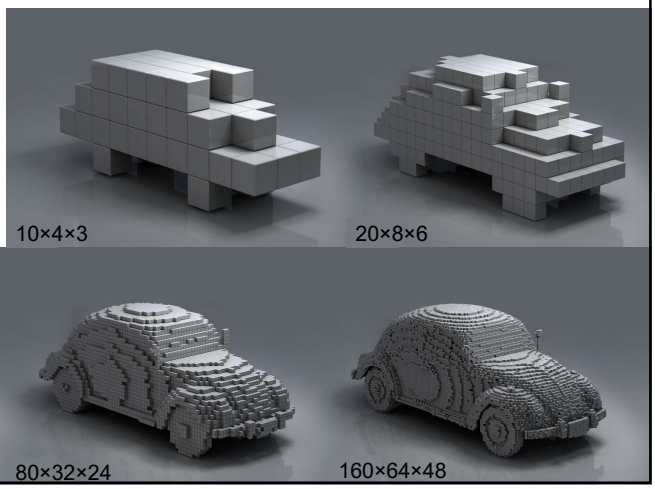
Occupazione spaziale dei dataset voxelizzati

- ✓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

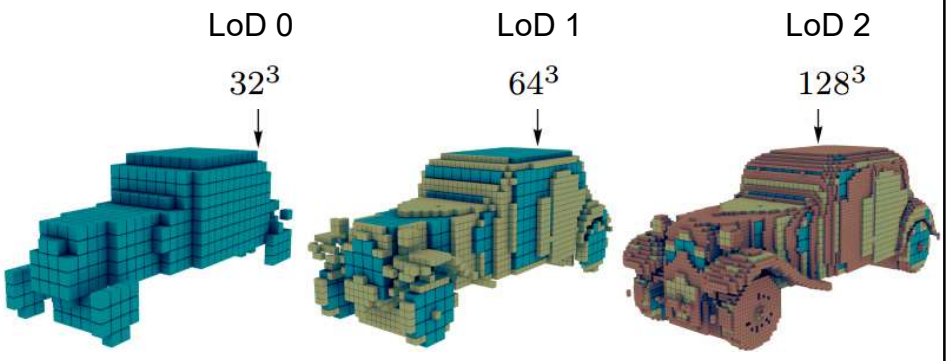
Risoluzione e costo in memoria

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



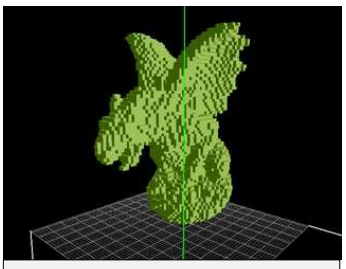
39

Piramide di livelli di dettaglio per voxel



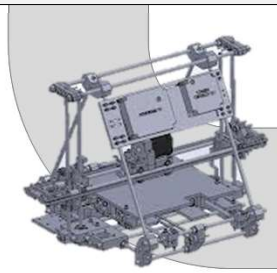
40

Voxel = 1 bit (pieno / vuoto)



Voxelized dataset
Voxel: pieno/vuoto

- ✓Input naturale di (alcuni) tipi **3D printing devices**
  - ⇒Rapid prototyping



3D printer

oggetti stampati

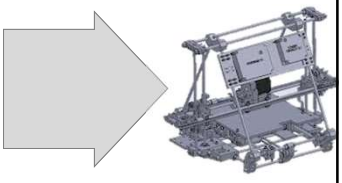
41

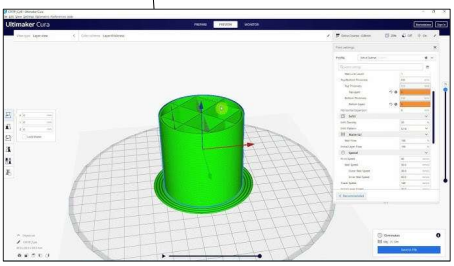
Parentesi: printing a 3D mesh su stampanti additive

Mesh Poligonale
ben orientata e chiusa!

Slicer
(software di «slicing»)

G-Code
Programma composto da istruzioni che la stampante e in grado di eseguire





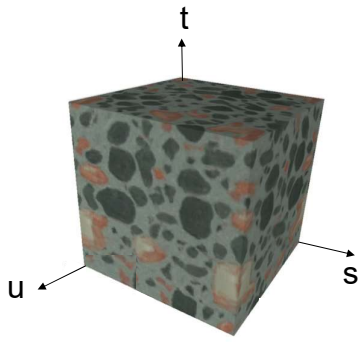
42

Quando 1 Voxel = un *indice* (al tipo di terreno)

Esatto, proprio i terreni di minecraft ™

44

Se Voxel = 1 colore RGB: “texture volumetriche”



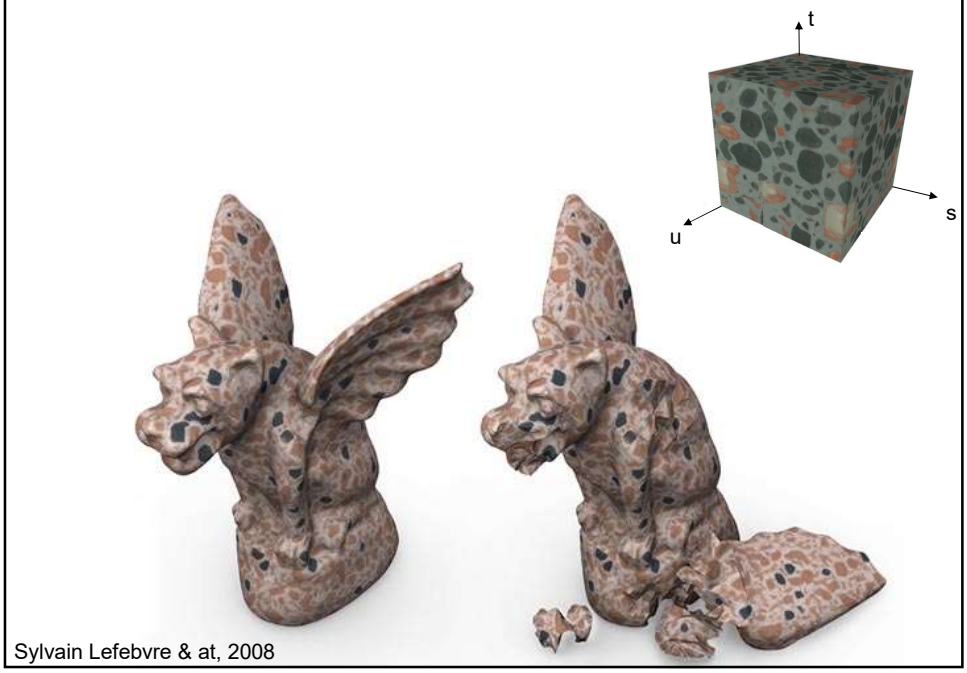
45

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?

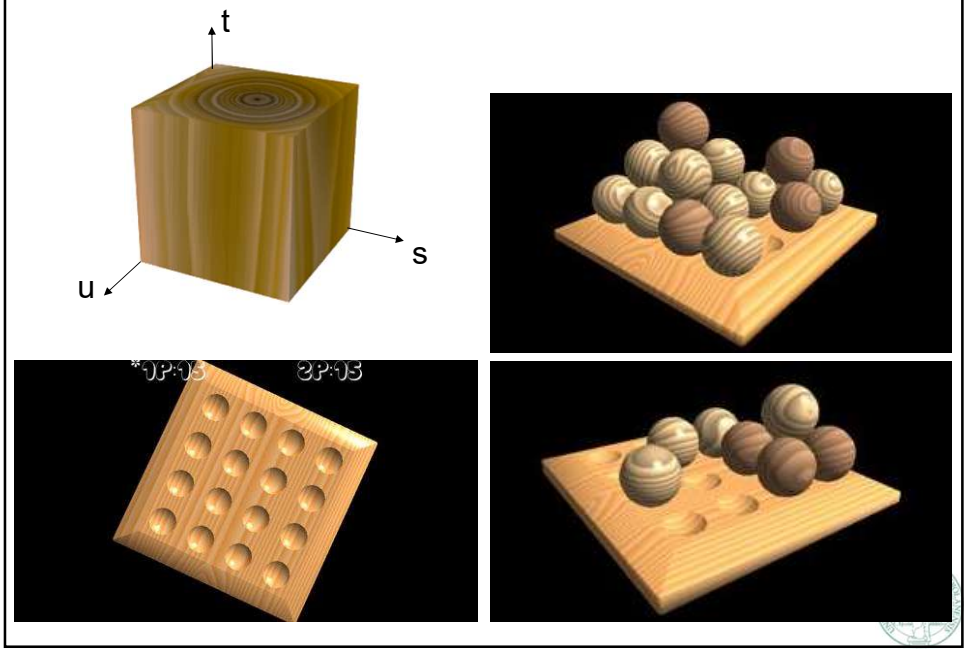
46

Un *colore* per voxel: “texture volumetriche”



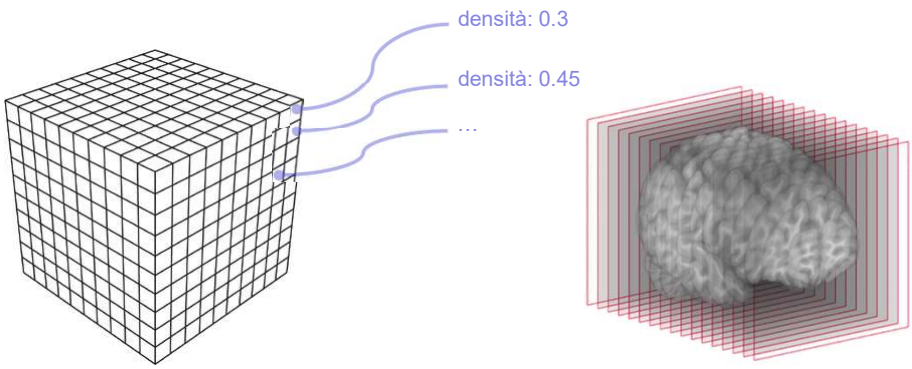
47

Volumetric Textures (o “solid Textures”)



48

Voxel = 1 scalare (es fra 0.0 e 1.0)

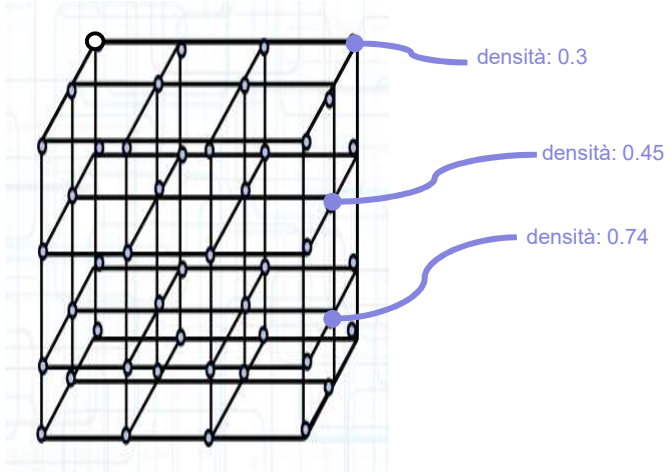


Esempio di File format: **DICOM**
(medicina)

```
Volume float [RES_X] [RES_Y] [RES_Z]
```

49

Voxelized models: uno scalare per voxel

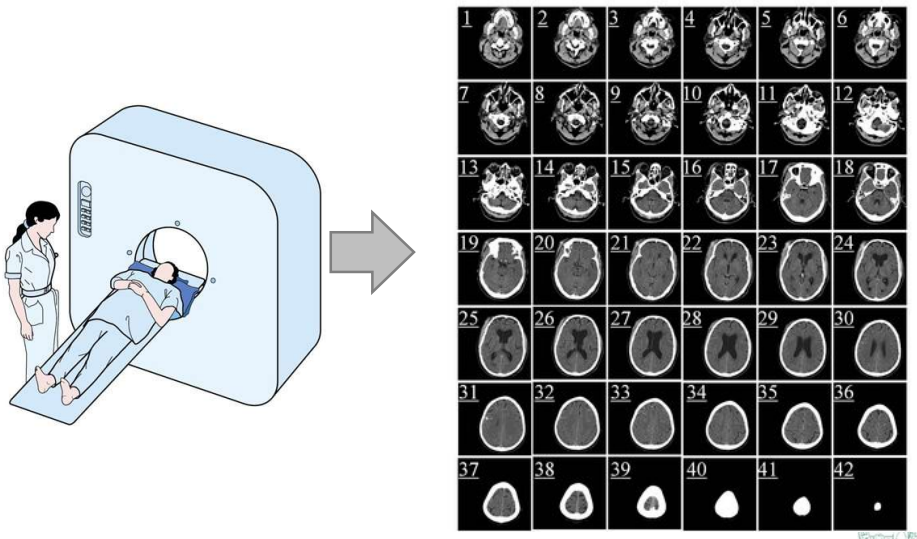



```
float volume[RES_X] [RES_Y] [RES_Z]
```

50

Se 1 voxel = 1 float (valori di densità)

✓Output naturale di **CT scans**



51

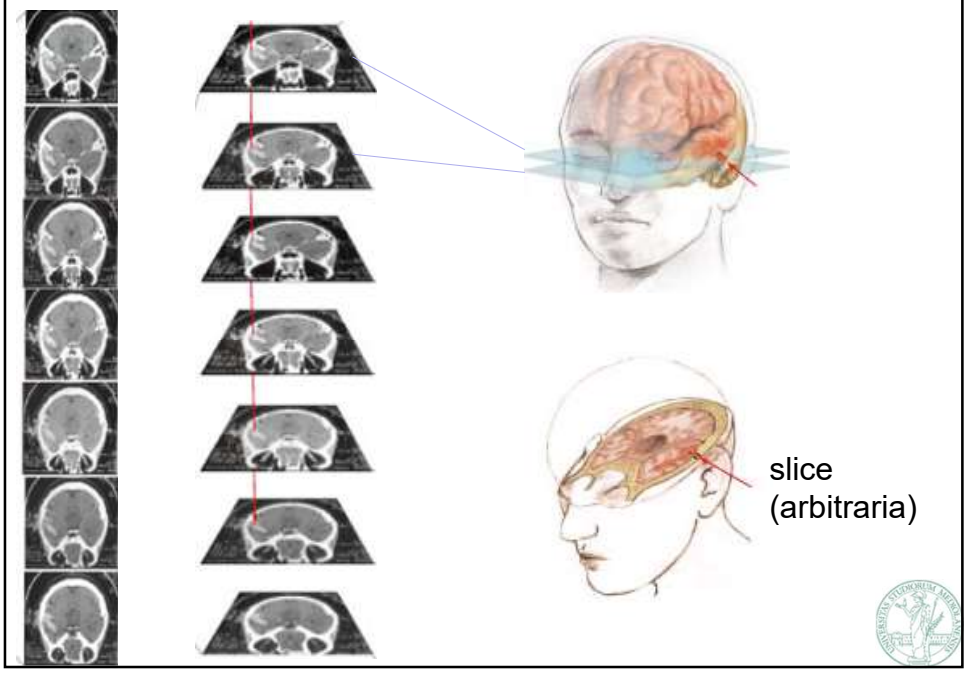
Dati voxel con intensità scalare per voxel

Output “naturale” di importanti tecniche di acquisizione 3D usate in campo diagnostico per la medicina, come:

- ✓ **Tomografia Assiale Computerizzata** (**TAC**, o **CT**)
  - ⇒dato voxel ottenuto processando serie di proiezioni radiografiche da molte angolazioni
  - ⇒attraverso appositi algoritmi di ricostruzione volumetrica
  - ⇒ogni voxel misura: coefficiente di attenuazione dei raggi X
- ✓ oppure, da **Risonanza Magnetica**
  - ⇒principio analogo, diversa fisica
  - ⇒ogni voxel misura: risposta del tessuto al campo magnetico
- ✓ oppure, da **PET scan**
  - ⇒nuovamente, principio analogo, ma fisica diversa
  - ⇒ogni voxel misura: attività del tracciante radioattivo (≈ concentrazione / attività metabolica locale)

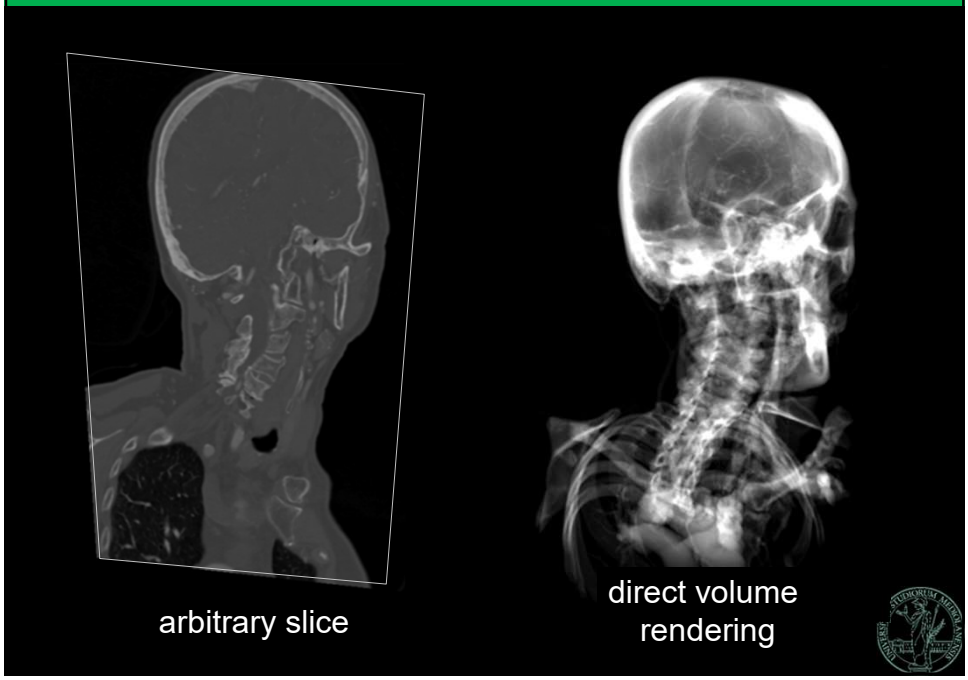
53

Se 1 voxel = 1 float (valori di densità)



55

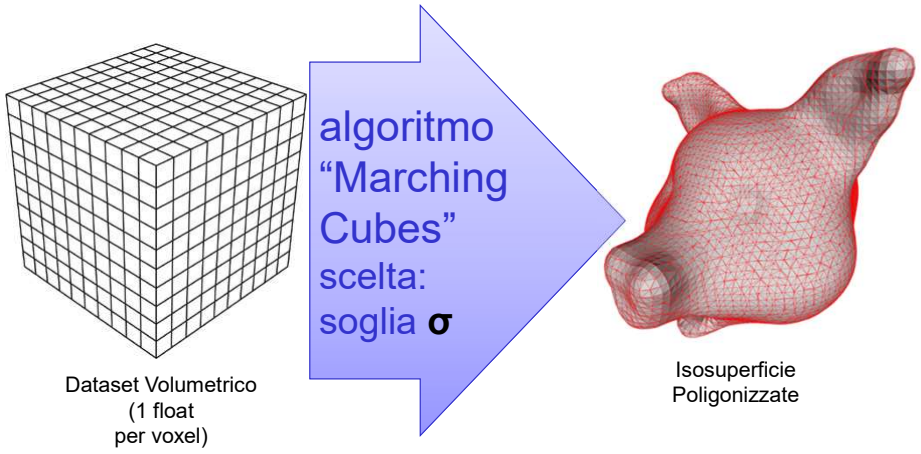
Voxelized volumes con voxel = 1 float fra 0 e 1



56

Poligonizzazione di un modello volumetrico

o segmentazione



58

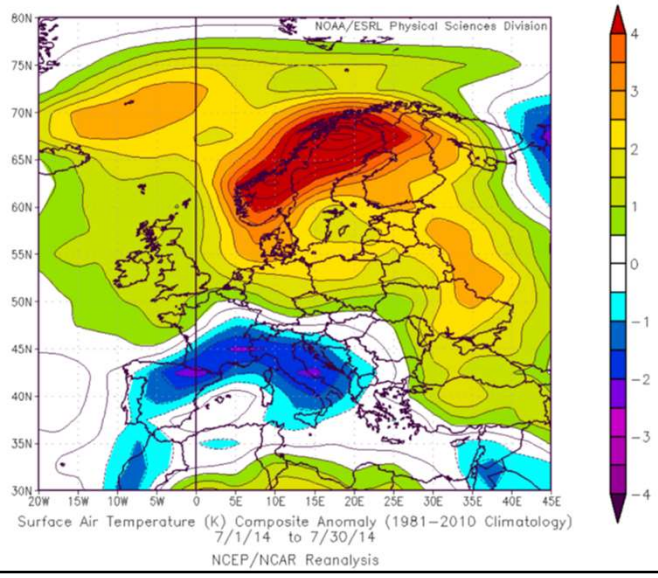
Isolinee e isosuperfici

- ✓ Su un piano (2D):
  - ⇒ ogni punto del piano ha un valore scalare (esempio: pressione, o altezza – «height field»)
  - ⇒ considero 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** » (isolinea = linea di isovalori)
- ✓ Su un volume (3D):
  - ⇒ ogni punto dello spazio ha un valore scalare (esempio: densità, pressione, temperatura…)
  - ⇒ considero 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** » (iso-superficie = …)

59

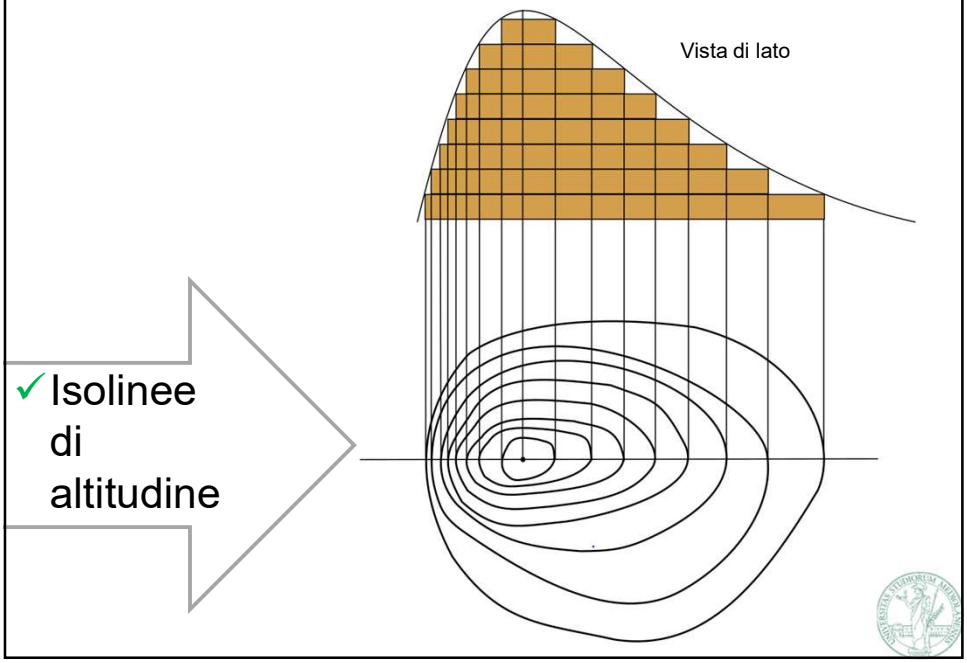
Esempi familiari di isolinea (in 2D)

✓Isolinee di pressione



60

Esempi familiare di isolinea (in 2D)



✓Isolinee di altitudine

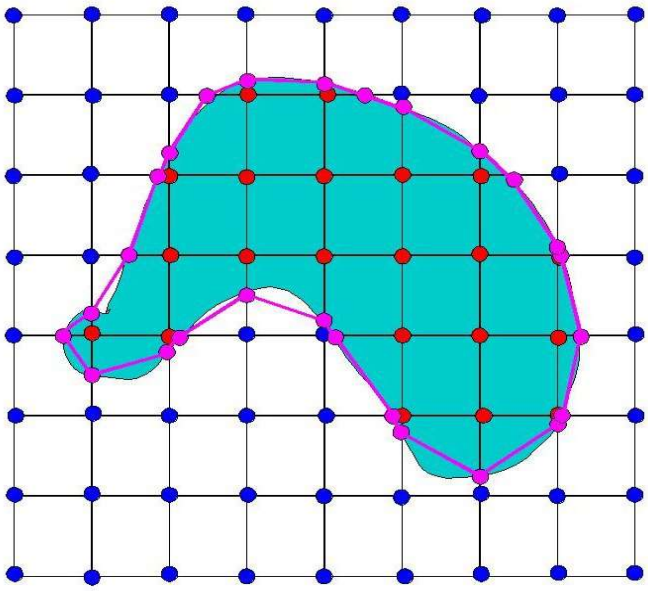
61

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
- ✓ Soluzione: algoritmo «**marching cubes**»
- ✓ Per spiegarlo, vediamo prima l'analogo in 2D: l'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**, cioè una linea chiusa di valore σ (appossimata da segmenti) che racchiuda tutti i pixel di valore superiore a σ

62

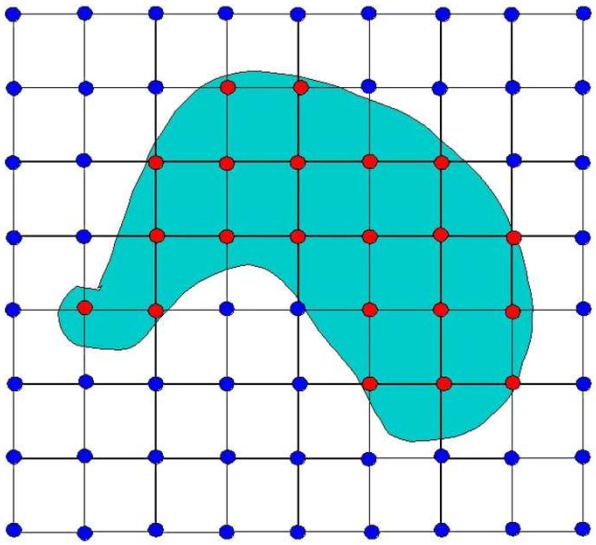
Algoritmo marching squares (per 2D)



63

Algoritmo marching squares (per 2D)

✓Sogliare voxels



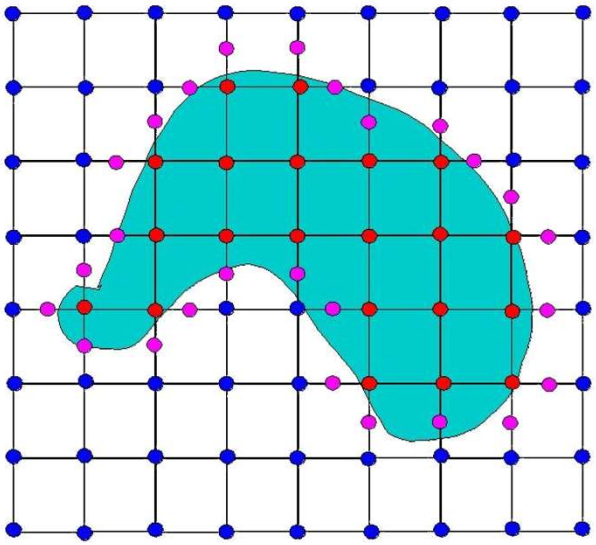
voxel con valore $< \sigma$

voxel con valore $\geq \sigma$

64

Algoritmo marching squares (per 2D)

✓Trovare intersezioni



voxel con valore $< \sigma$

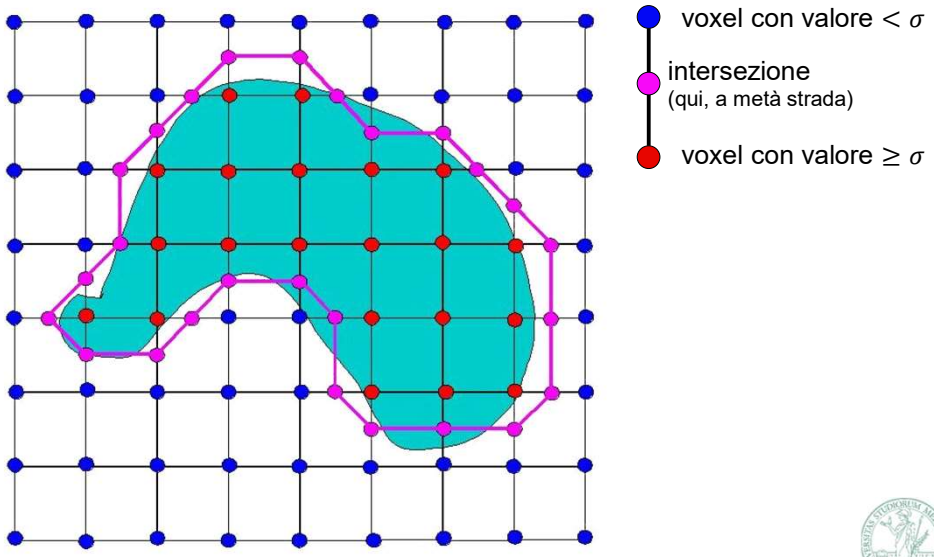
intersezione
(qui, a metà strada)

voxel con valore $\geq \sigma$

65

Algoritmo marching squares (per 2D)

✓Unire intersezioni creando segmenti

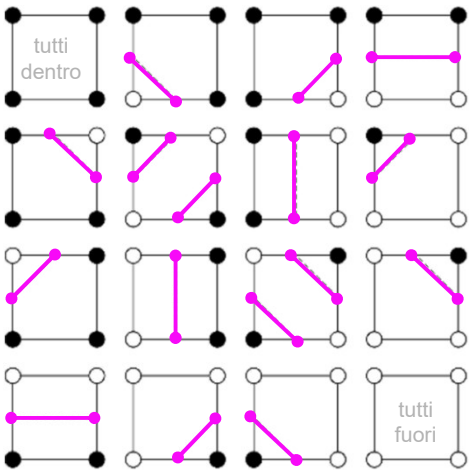


66

Algoritmo marching squares (per 2D)

Come trovare
i segmenti che connettono
le intersezioni?

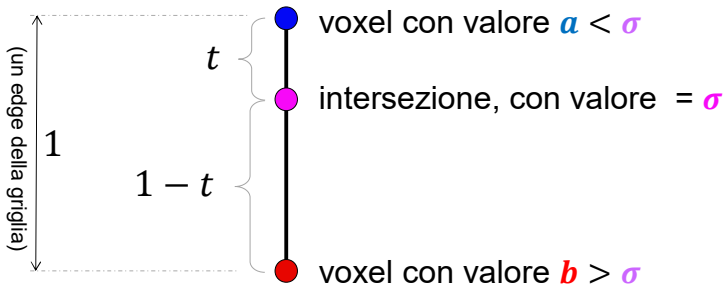
- ✓ Consideriamo un quadrato fra 4 voxel
- ✓ Ogni suo vertice è «dentro» $< \sigma$ o «fuori» $\geq \sigma$
- ✓ Per ogni combinazione, (e sono solo $2^4 = 16$) decido, una volta per tutte, i segmenti da costruire per unire le intersezioni
- ✓ ottengo questa tabella:



67

Algoritmo marching squares (per 2D)

✓Come trovare le intersezioni



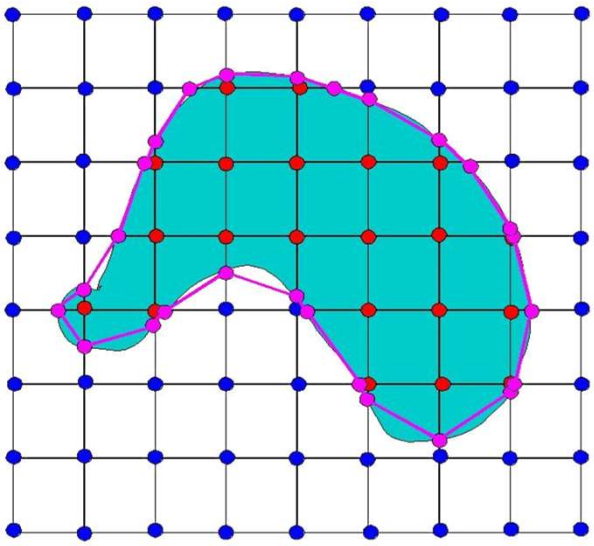
✓Ipotesi: segnale interpolato linearmente:

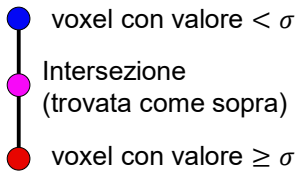
$$a(1-t)+b\,t=\sigma \quad \underset{\text{QUINDI}}{\Leftrightarrow} \quad t=\frac{\sigma-a}{b-a}$$

68

Algoritmo marching squares (per 2D)

✓Risultato

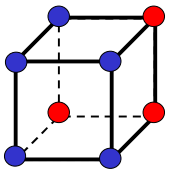




69

Generalizzando a 3D: algoritmo marching cubes

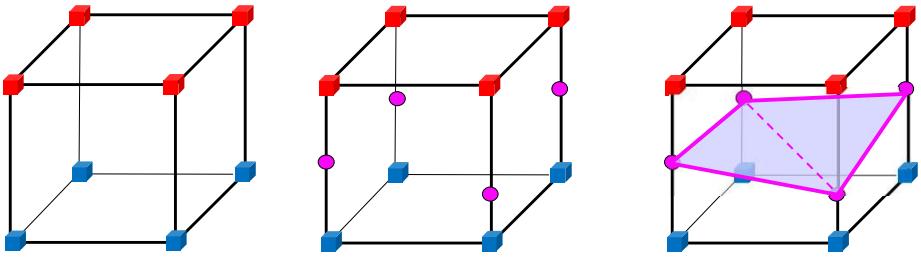
- ✓ Ogni voxel della griglia è dentro o fuori
  - ⇒ valore ≥ oppure < della soglia prescelta
- ✓ Ogni edge della griglia (orientato lungo la X, Y o Z), che connetta un vertice dentro ad uno fuori, ha una intersezione con l'isosuperficie cercata
  - ⇒ la si trova come nel caso 2D
  - ⇒ per ciascuna intersezione, 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 decidere, in ciascun caso, quali poligoni creare per connettere i vertici sui lati della griglia



70

Generalizzando a 3D: algoritmo marching cubes

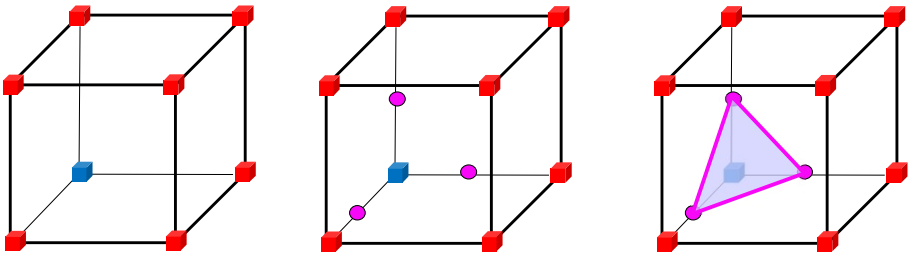
- ✓ Esempio di uno dei 256 casi



71

Generalizzando a 3D: algoritmo marching cubes

✓Esempio di uno dei 256 casi



7 Voxel sotto soglia

1 Voxel sopra soglia

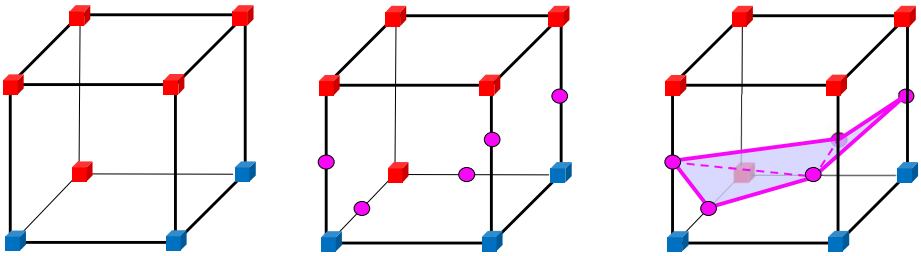
3 Intersezioni (calcolate nei segmenti)

Connesse da un triangolo

72

Generalizzando a 3D: algoritmo marching cubes

✓Esempio di uno dei 256 casi



5 Voxel sotto soglia

3 Voxel sopra soglia

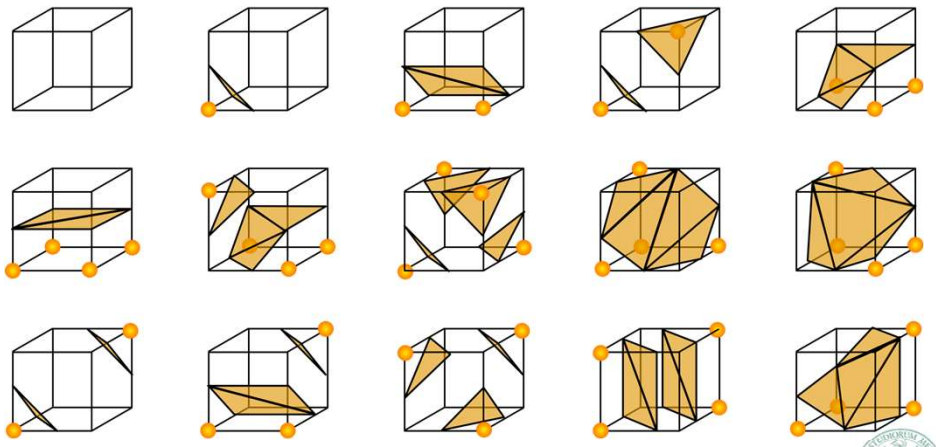
5 Intersezioni (calcolate nei segmenti)

Connesse da tre triangoli

73

Marching cube table

- ✓ La tabella ha 256 casi
  - ⇒ sono tutti analoghi ad uno di questi 15 (per simmetria)



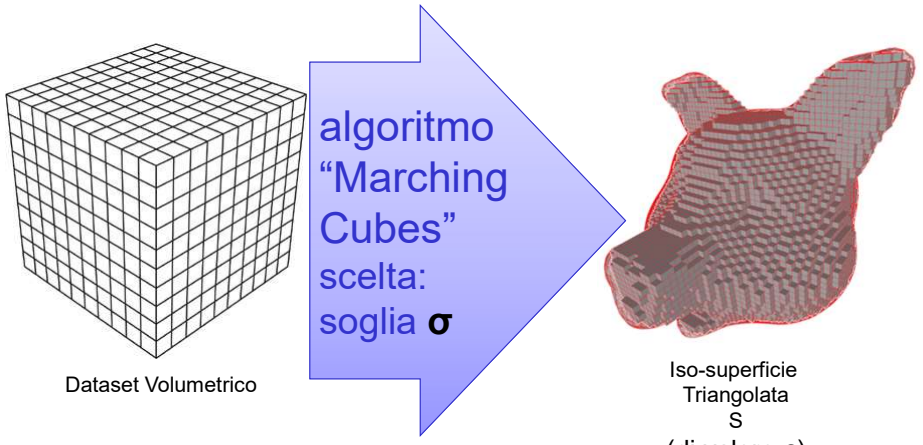
74

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

75

Da modello voxelizzato a Mesh poligonale

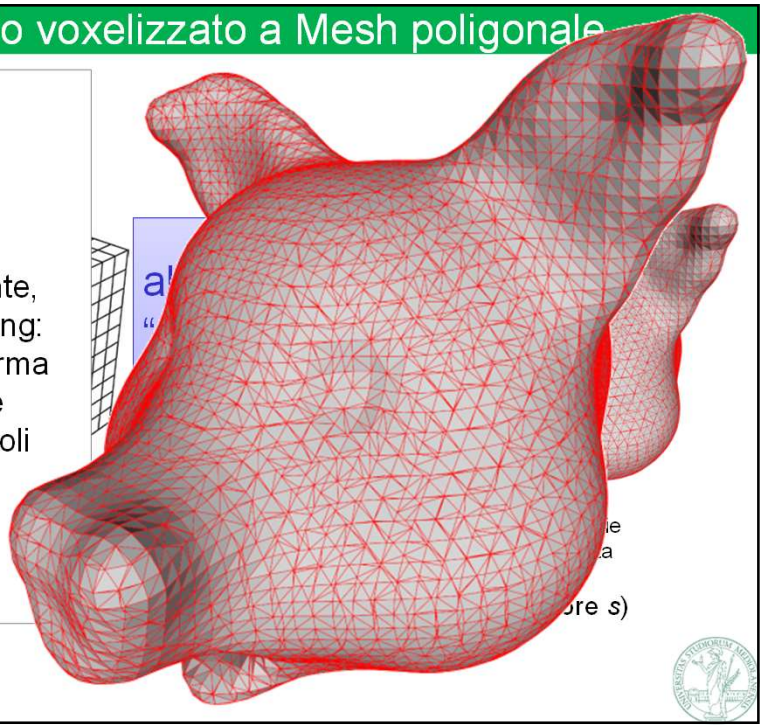


76

Da modello voxelizzato a Mesh poligonale

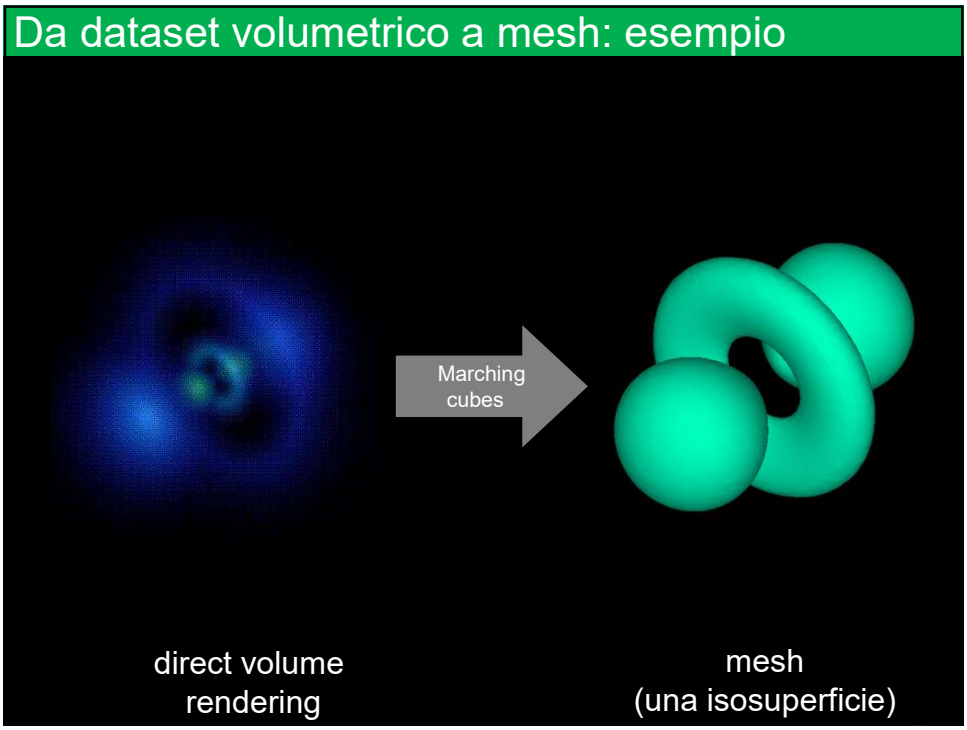
Output:
mesh chiusa,
two-manifold,
ben orientata

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



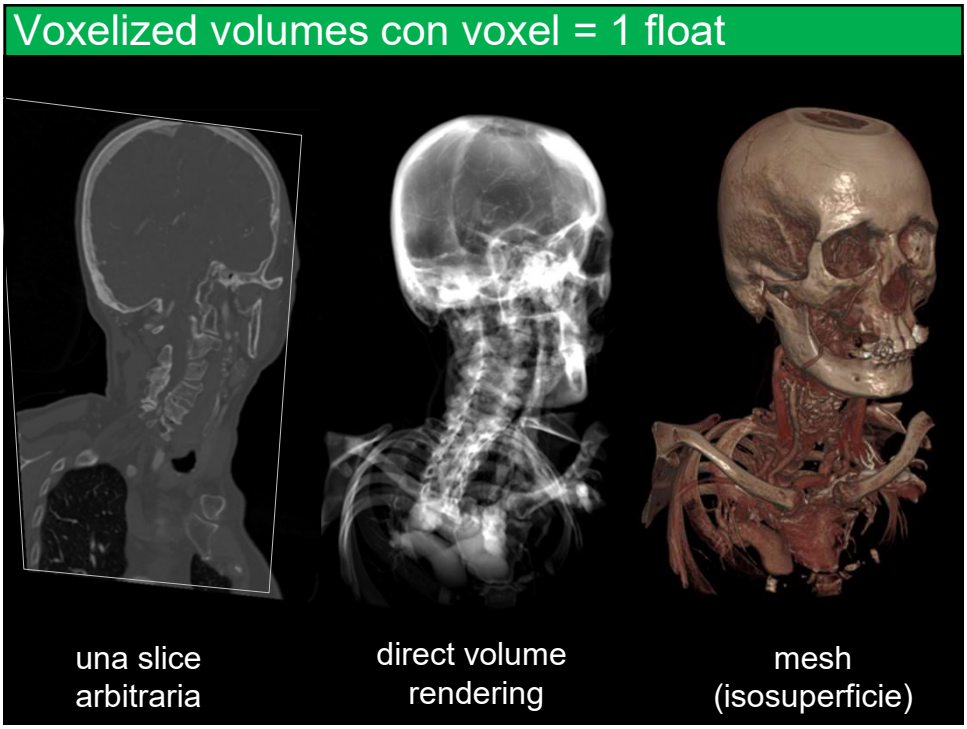
77

Da dataset volumetrico a mesh: esempio



78

Voxelized volumes con voxel = 1 float



80

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

Poisson Reconstruction

81

Da Nuvola di punti a Triangle Mesh passando per…

		ELEMENTI DISCRETI			CONTINUI
		regolari *«a griglia»*	semi-regolari o irregolari		
			elementi simpliciali	elementi non simpliciali	
SUPERFICIALI	2-manifold *«rappresenta una vera superficie»*	**Height Field** **Range Scan**	**Triangle Mesh**	**Polygonal Mesh** **Quad-Mesh** **Quad dominant Mesh**	**Subdivision surface** **Parametric Surface** **(come B-spline)**
	non-manifold *«non rappresenta una sup»*	**Set di Range**	**Point Cloud**		
VOLUMETRICI	(3-manifold)	**Voxels** **Solid Textures**	**Tetra Mesh**	**Hexa Mesh**	**Implicit model** **(es. CSG)**

82

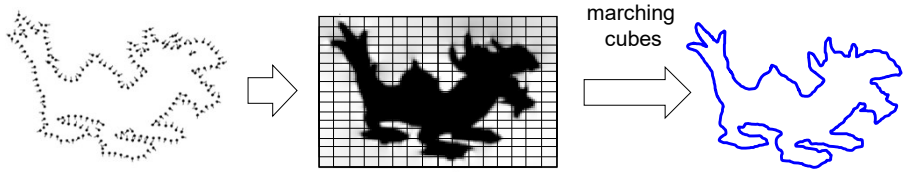
Strutture volumetriche come dataset intermedio

- ✓ «Poisson reconstruction»
  - ⇒ Un algoritmo per convertire una point cloud in una mesh triangolare che consiste nel è passare attraverso a un struttura volumetrica a voxel
- ✓ Idea:
  - ⇒ 1. Stendere una griglia volumetrica (o «lattice») nel volume coinvolto
  - ⇒ 2. Memorizzare una «signed distance function» nel volume (1 valore scalare per voxel, che memorizza una stima della distanza dalla superficie, negativo se il punto è dentro la superficie)
  - ⇒ 3. estrarre iso-superficie (attraverso marching cubes)
- ✓ Nel passo 2: ogni punto della nuvola di posizione $\mathbf{p}$ e normale $\hat{\mathbf{n}}$ «vuole» che…
  - ⇒ $f(\mathbf{p}) = \mathbf{0}$ (cioè, il valore della funzione in $\mathbf{p}$ sia 0)
  - ⇒ $\nabla f(\mathbf{p}) = \hat{\mathbf{n}}$ (cioè, il gradiente della funzione in $\mathbf{p}$ sia il vettore $\hat{\mathbf{n}}$)
  - ⇒ Si tratta quindi di computare un volume di voxel che rappresenti una funzione f che aderisca il meglio possibile a queste richieste, per tutti i punti
  - ⇒ cioè, una funzione che sia una SDF

83

Poisson Reconstruction

Esempio in 2D



Point Cloud

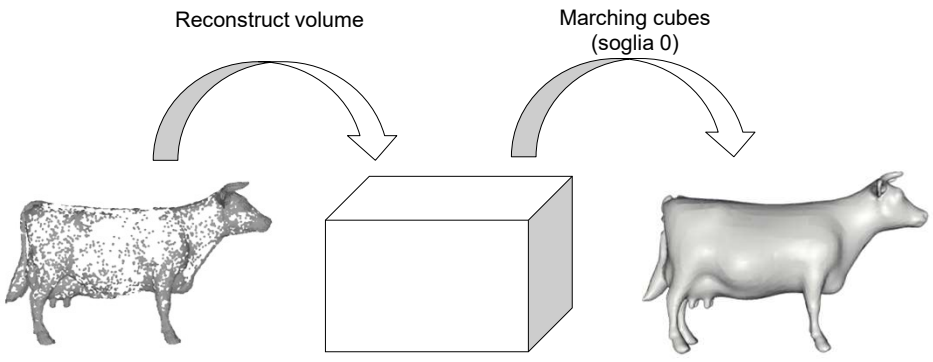
SDF
Signed Distance Field
Griglia 2D di voxel
(1 voxel = 1 scalar)

Tri-Mesh
(isosurface)

Images by Micheal Kazhdan

85

Poisson Reconstruction



Point Cloud

SDF
Signed Distance Field
Griglia 3D di voxel
(1 voxel = 1 scalar)

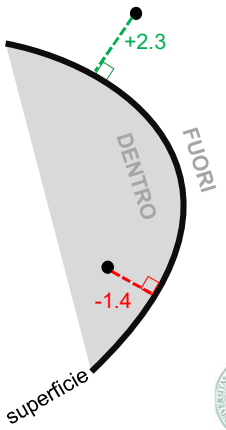
Tri-Mesh
(isosurface)

Images by H. Hoppe

86

Signed distance field

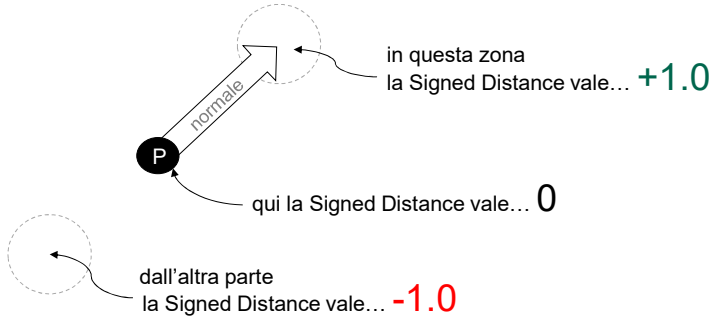
- ✓ Per definizione, la distanza (da qualcosa) è un valore reale *positivo*
- ✓ Definiamo però **Distanza con segno** (**signed distance**) di un punto da una superficie chiusa un numero il cui valore assoluto è la distanza, e il cui segno riflette se il punto in questione sia dentro (neg) oppure fuori (pos)
- ✓ Un **campo** di distanze con segno (**Signed Distance Field** – **SDF**) è una struttura dati che per ogni punto di un volume modella la distanza con segno di quel punto (da una superficie chiusa)
- ✓ Memorizziamo un **SDF** come una volume di voxel in cui per ogni voxel registriamo la sua Signed Distance
- ✓ E' un modo (dispendioso) di memorizzare una superficie chiusa



87

Da nuvola di punti a SDF

- ✓ Ogni punto della nuvola (provvisto come sappiamo di normale) fornisce un'indicazione sul valore della SDF in quel punto nel suo intorno:



- ✓ L'idea della Poisson Reconstruction è costuire un SDF (di voxel) che «mette insieme», e in qualche modo media fra loro, tutte le indicazioni di questo tipo

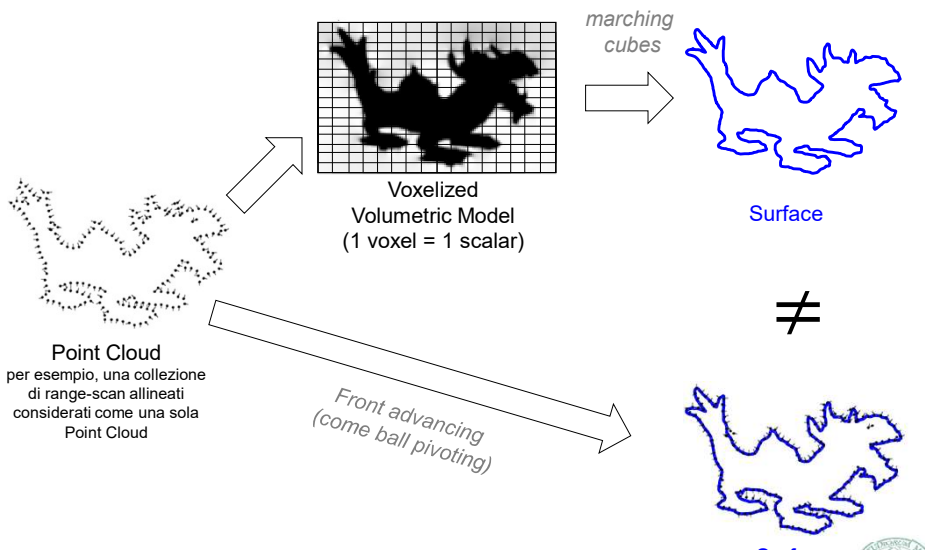
88

Da nuvola di punti a mesh: sommario

- ✓ Modi per convertire una nuvola di punti in una mesh
  - ⇒ Modo diretto: algoritmi di Front Advancing (come Ball Pivoting) o triangolazioni di Delaunay aggiungono una connettività di triangoli per unire i punti della nuvola con triangoli (potenzialmente, scardandone alcuni) -- il modo che abbiamo visto nella prima lez sulle mesh
  - ⇒ Modo indiretto: si converte la nuvola di punti in un dataset volumetrico di voxel (che campiona nel volume una stima della *Signed Distance Function - SDF*) e si estrare da questo dataset una mesh poligonale (con algoritmo Marching Cubes con soglia 0)

89

Un paragone fra due approcci



90

Da nuvola di punti a mesh

- ✓ Nei **metodi diretti** (ball-pivoting, o altri front advancing, o triangolazioni di Delaunay) si mantengono i punti della point-cloud inalterati, come vertici della mesh
- ✓ Questo è problematico, quando la point-cloud presenta difetti come:
  - ⇒ rumore o outliers
  - ⇒ difetti di allineamento (se la nuvola di punti è ottenuta allineando e unendo due nuvole separate, come spesso è il caso)
  - ⇒ densità diverse non volute, accidentali (per es, due point-cloud parziali sovrapposte presentano molti più campioni nelle parti ripetute)
  - ⇒ parti mancanti
- ✓ Sono tutti difetti comuni nelle point-cloud generate da acquisizioni 3D

91

Da nuvola di punti a mesh

- ✓ Con la **Poisson Reconstruction**…
  - ⇒ vengono prodotti nuovi vertici che in sostanza *mediano* le posizioni dei punti nella nuvola di input, abbattendo rumore e difetti
  - ⇒ viene sempre prodotta una mesh two-manifold, chiusa, e ben orientata
- ✓ Ma…
  - ⇒ la mesh prodotta è molto irregolare; (anche se i punti di input fossero molto ben distribuiti)
  - ⇒ anche se la nuvola di punti avesse avuto una risoluzione adattiva, la mesh prodotta perde questa caratteristica;
  - ⇒ i vertici vengono ricampionati: è uno svantaggio, quando i punti originali sono molto accurati;
  - ⇒ è molto oneroso in termini di computazione e memoria, a meno che non venga scelta una risoluzione molto bassa, che produce mesh a risoluzione bassa; (è la solita «*curse of dimensionality*» della rappresentazione voxel)
  - ⇒ vediamo ora un modo generale per rimediare a quest'ultimo problema (per qualsiasi struttura basata su voxel – non solo per questo uso)

92

Dataset volumetrici a griglia adattivi

- ✓ Il problema della «curse of dimensionality» è legato ad una caratteristica del dato a voxel: la sua risoluzione non è *adattiva*
- ✓ Esistono strutture dati **gerarchiche** che sono più compatte, grazie alla loro risoluzione adattiva
- ✓ Una delle più diffuse è l'**oct-tree** (o anche **octree**)
  - ⇒ Il soggetto della prossima lezione!
  - ⇒ Vediamo prima una sua versione 2D: il «**quad-tree**»
  - ⇒ Cioè: in questi primi esempi, ipotizziamo di voler rappresentare, in forma di quad-tree, un dataset "voxelizzato" (ma in 2D) con 1 bit per voxel (1 = pieno , 0 = vuoto) (in sostanza, un'immagine in bianco e nero)

93