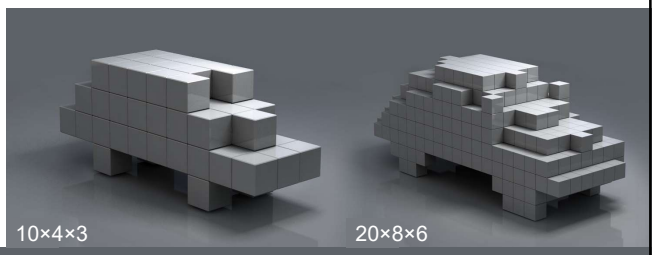
## Occupazione spaziale dei dataset voxelizzati

- ✓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 quello visto sopra, con un 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»

36

## Risoluzione di un dataset di voxel

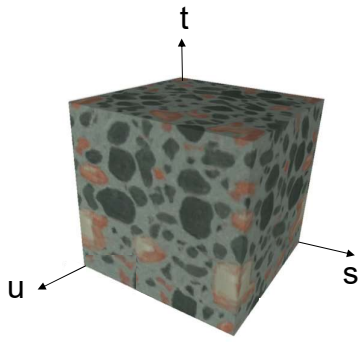
- ✓ risoluzione: un intero per lato res = (X,Y,Z)
- ✓ La risoluzione non è adattiva
- ✓ Esempio di piramide di livelli di dettaglio:





37

## Se Voxel = 1 colore RGB: “texture volumetriche”



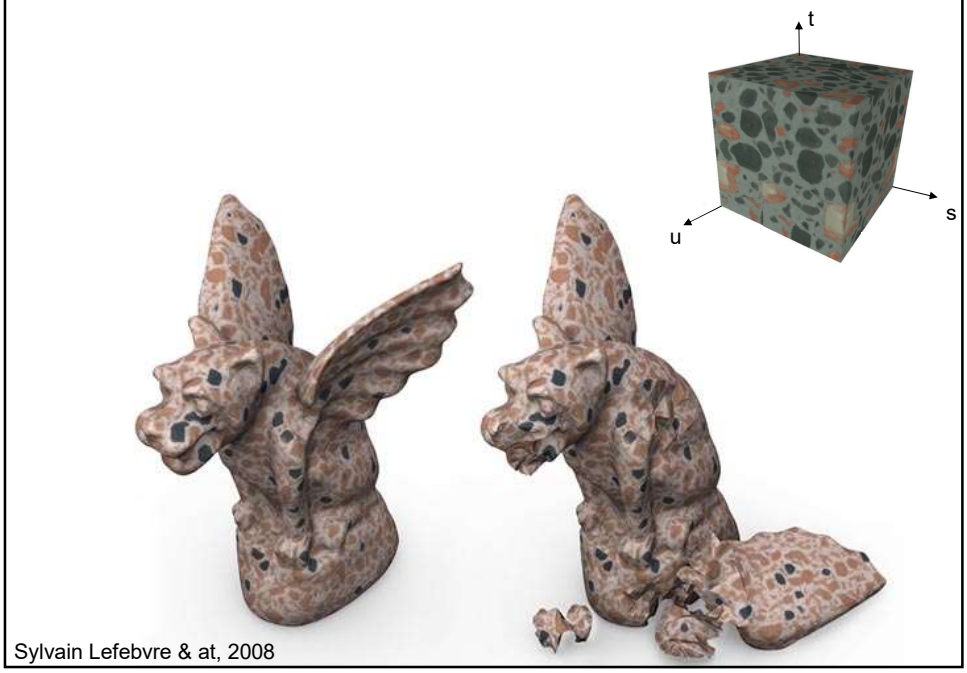

43

## Volumetric Textures (o “solid Textures”)

- ✓ 1 texel = 1 voxel
  - ⇒«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
  - ⇒come gli oggetti sono colorati all’interno
  - ⇒utile per modelli che si possono rompere
  - ⇒utile per pattern volumetrici 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?

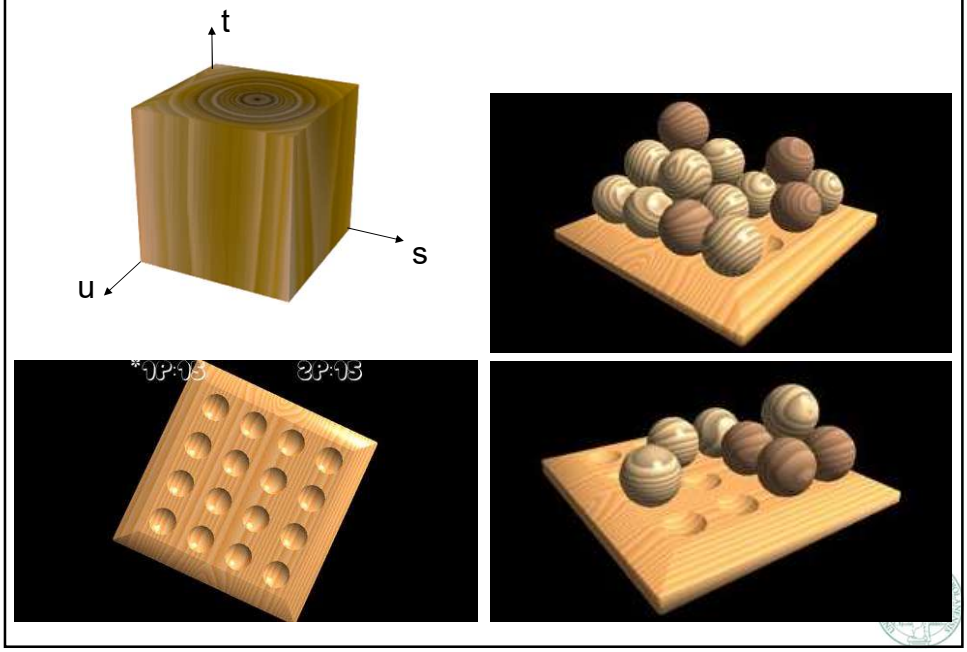
44

## Un *colore* per voxel: “texture volumetriche”



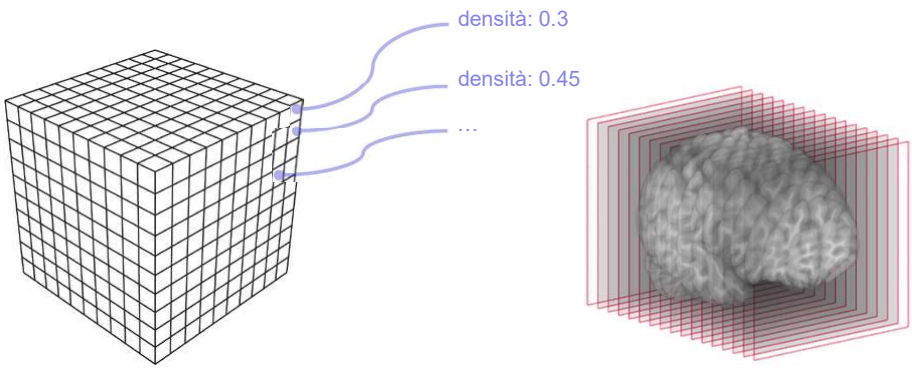

45

## Volumetric Textures (o “solid Textures”)




46

## Voxel = 1 scalare (es fra 0.0 e 1.0)

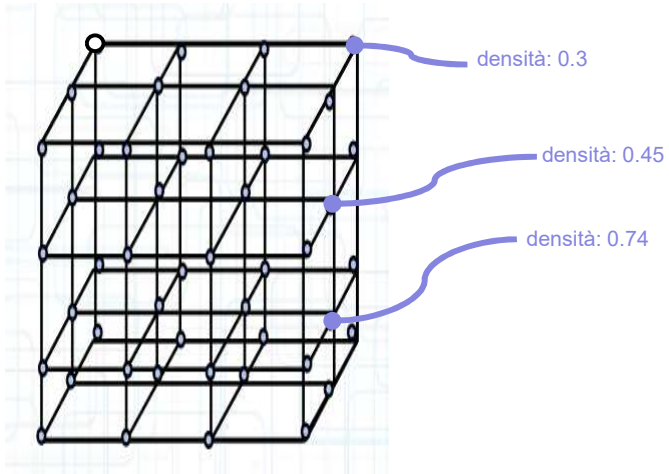



Esempio di File format: **DICOM**
(medicina)

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

47

## Voxelized models: uno scalare per voxel

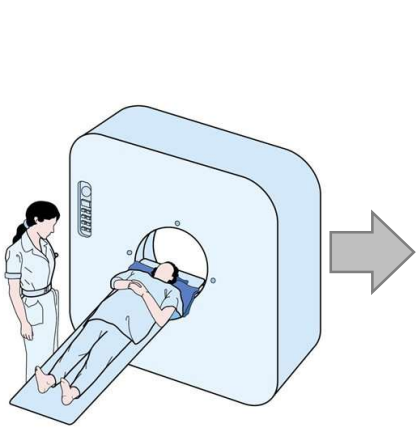


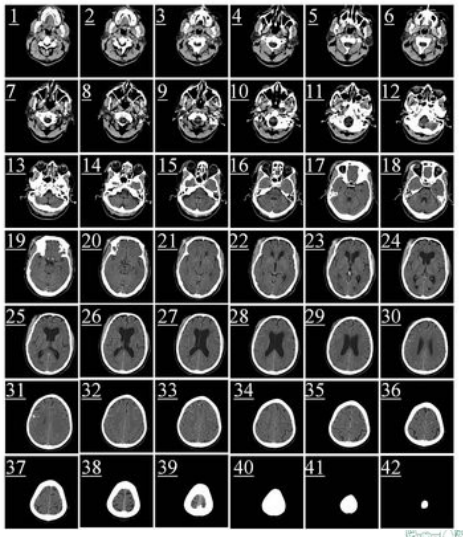

```
float volume[RES_X] [RES_Y] [RES_Z]
```

48

## Se 1 voxel = 1 float (valori di densità)

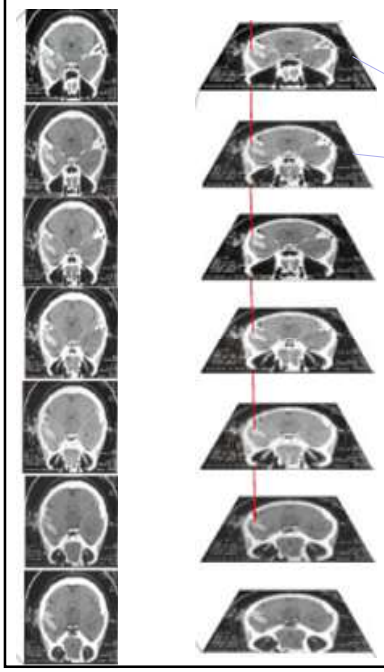
✓Output naturale di **CT scans**

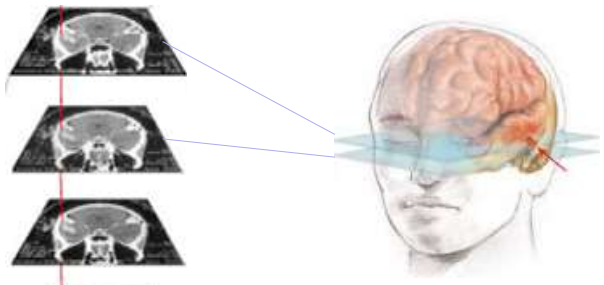





49

## Se 1 voxel = 1 float (valori di densità)

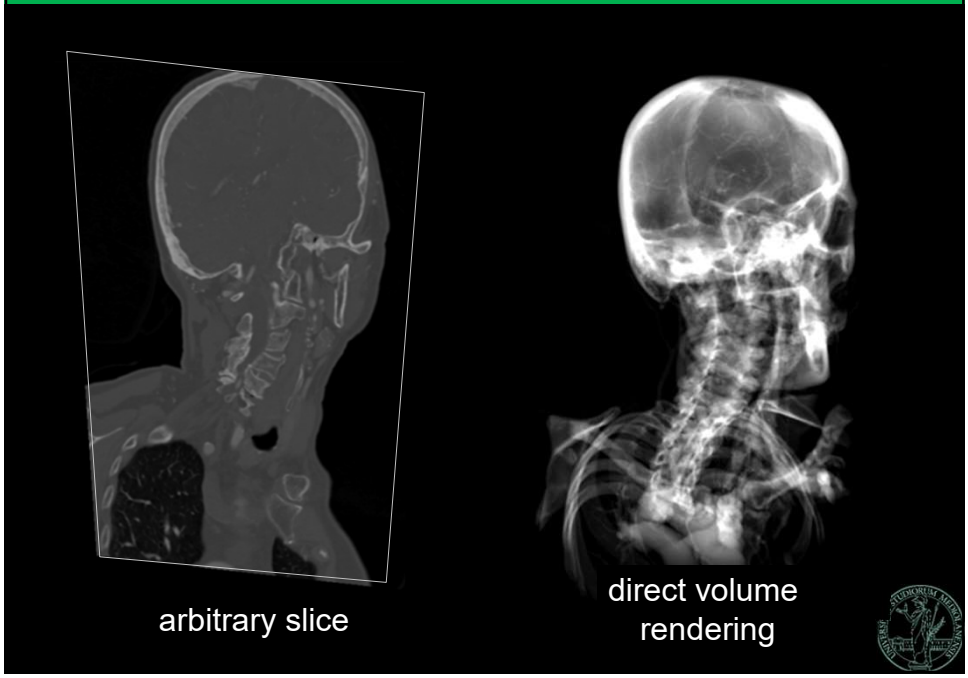




slice
(arbitraria)

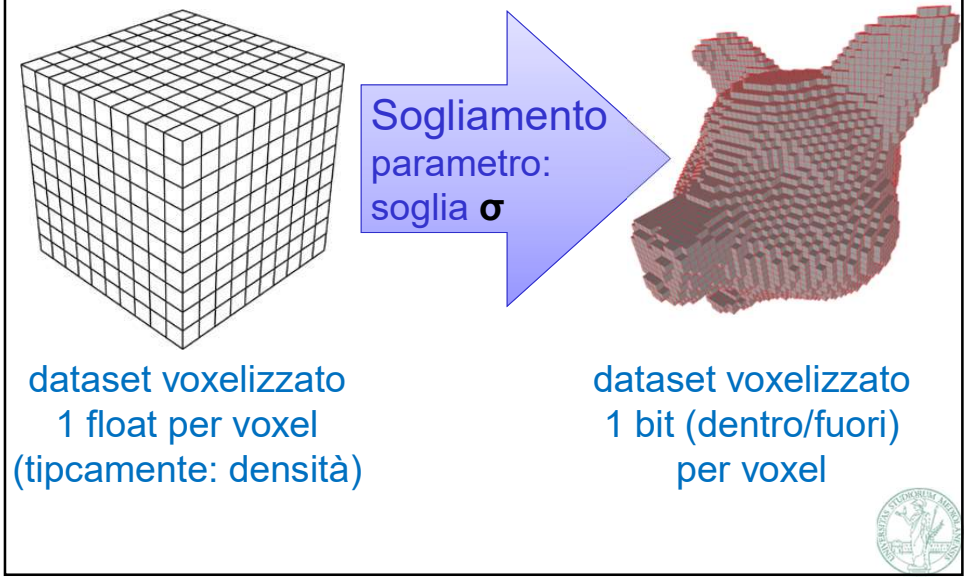
52

## Voxelized volumes con voxel = 1 float fra 0 e 1




53

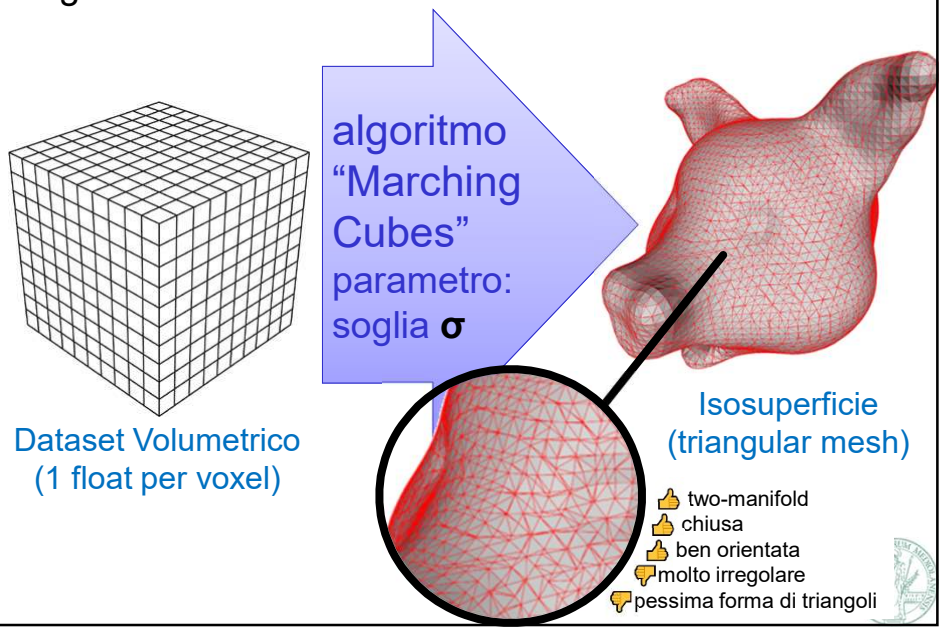
## Soglia sulla densità




54

## Poligonizzazione di un modello volumetrico

o segmentazione




55

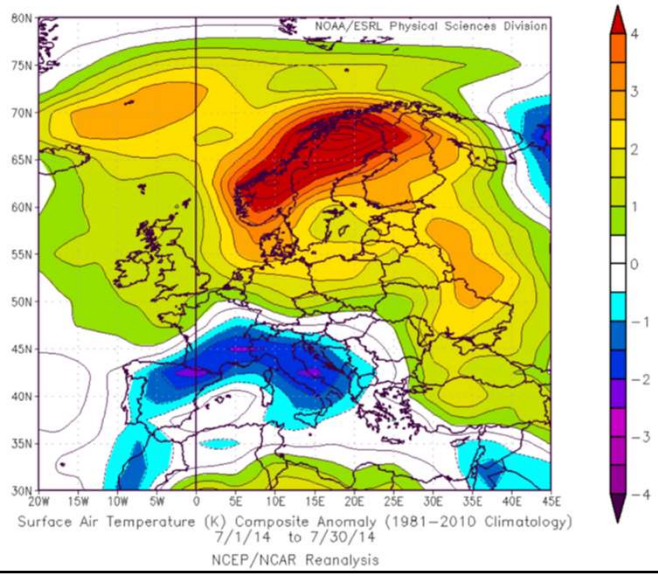
## Isolinee e isosuperfici

- ✓ 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 curva … che racchiude tutti i valori di valore > **σ** e i cui punti hanno valore tutti **σ**
  - ⇒ è detta 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… che racchiude tutti i valori di valore > **σ** e i cui punti hanno tutti valore è **σ**
  - ⇒ è la « **iso-superficie di valore σ** »

56

## Esempio di isolinea

✓ Isolinee di pressione




57

## Poligonizzazione di un modello volumetrico

✓ Obiettivo:
- ⇒ dato un modello volumetrico in cui un voxel = un valore scalare (per es, densità, temperatura…)
- ⇒ produrre una tri-mesh 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: una superficie chiusa (approssimata da triangoli) data da tutti i punti nel volume che valgono esattamente σ

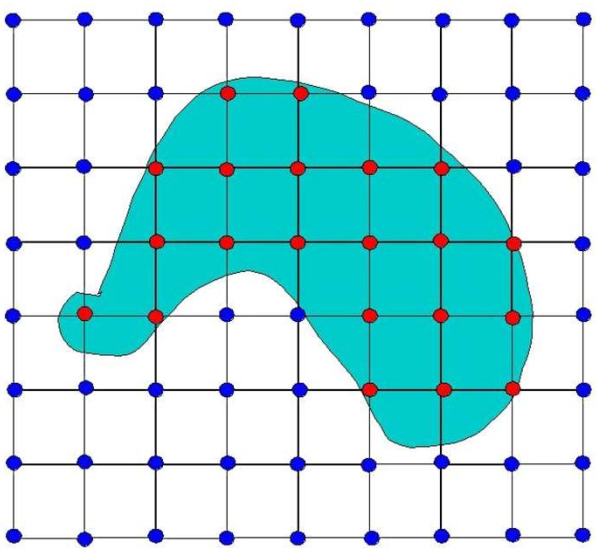
✓ Algoritmo: «**marching cubes**»

✓ Vediamo prima un analogo in 2D: algoritmo «**marching squares**»
- ⇒ data un'immagine rasterizzta in cui un pixel = un valore scalare (per es, denistà, temperatura…)
- ⇒ produce una **iso-linea** , la linea chiusa di valore σ (appossimata da segmenti) che racchiude tutti i pixel di valore superiore a σ

58

## Algoritmo marching squares (per 2D)

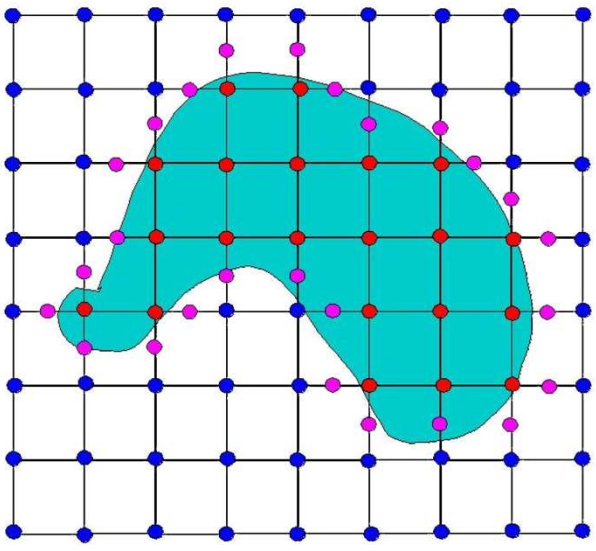
✓Sogliare voxels




60

## Algoritmo marching squares (per 2D)

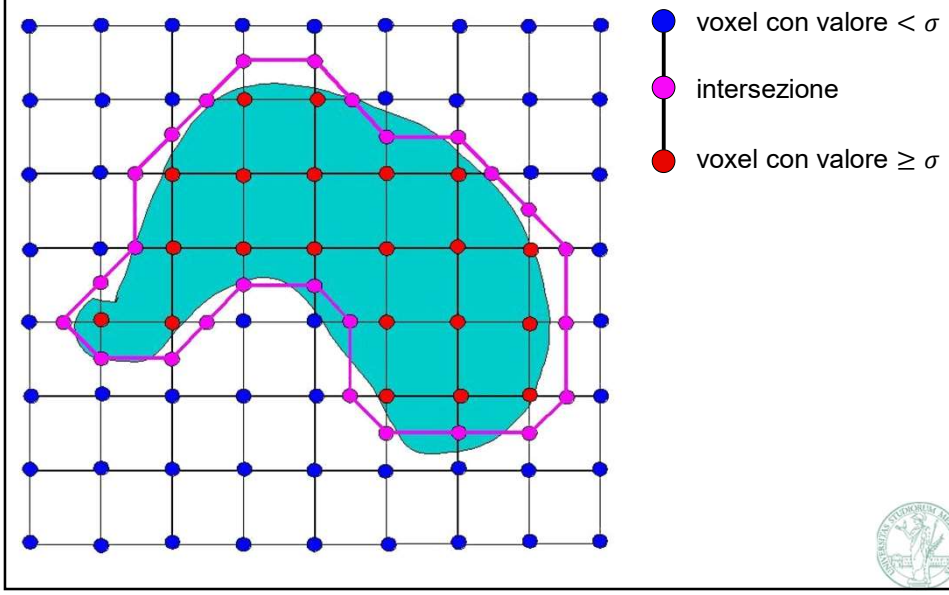
✓Trovare intersezioni




61

## Algoritmo marching squares (per 2D)

✓Unire intersezionicreando segmenti



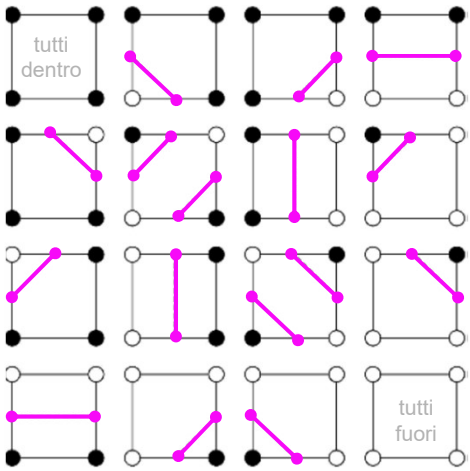

62

## 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

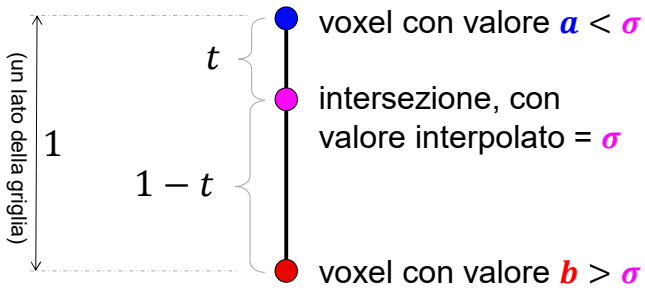
✓ ottengo questa tabella:




63

## Algoritmo marching squares (per 2D)

✓Come trovare le intersezioni




✓Ipotesi: segnale interpolato (linearmente):

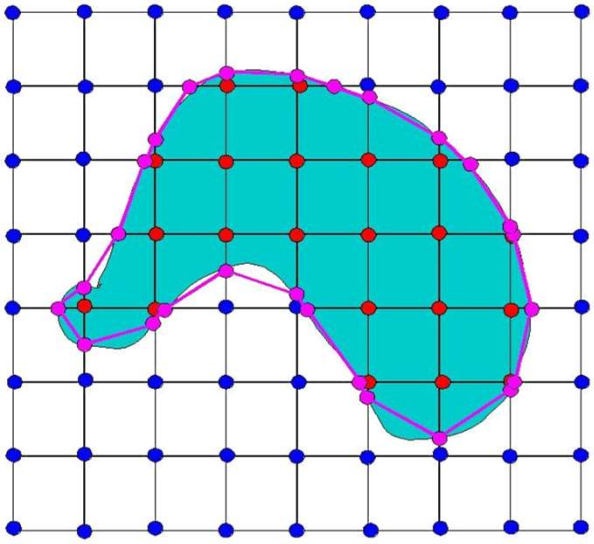
$$a(1-t)+bt=\sigma \quad \Leftrightarrow \quad t=\frac{\sigma-a}{b-a}$$

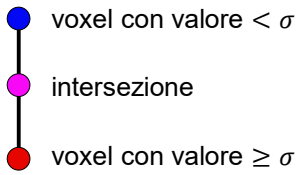
Verificare!

64

## Algoritmo marching squares (per 2D)

✓Variante: trovare intersezioni e unirle

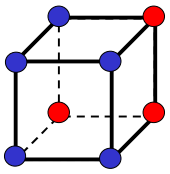





65

## Generalizzando a 3D: algoritmo marching cubes

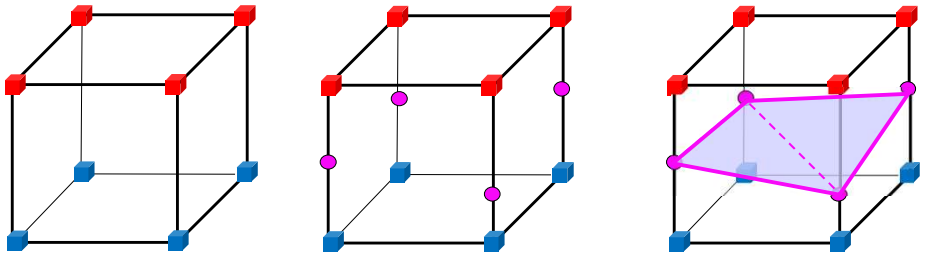
- ✓ Ogni voxel della griglia è dentro o fuori
  - ⇒ valore > o ≤ della soglia prescelta $\sigma$
- ✓ Ogni edge della griglia (orientato lungo la X, Y o Z), che connetta un vertice dentro ad uno fuori, ha un'intersezione con l'isosuperficie cercata
  - ⇒ la si trova trovata come nel caso 2D
  - ⇒ per ciascuna intersezione, creo un vertice della mesh
  - ⇒ ho ottenuto la **geometria** della mesh!
- ✓ Come ottengo la sua **connettività**?
  - ⇒ Scompongo la griglia in cubetti 1x1x1
  - ⇒ Ogni cubo ha 8 voxel ai vertici, ciascuno o dentro o fuori ➔ $2^8 = 256$ casi possibili
  - ⇒ Uso una tabella per descrivere, per ciascun caso, quali triangoli creare per connettere i vertici sui lati del cubo



66

## Generalizzando a 3D: algoritmo marching cubes

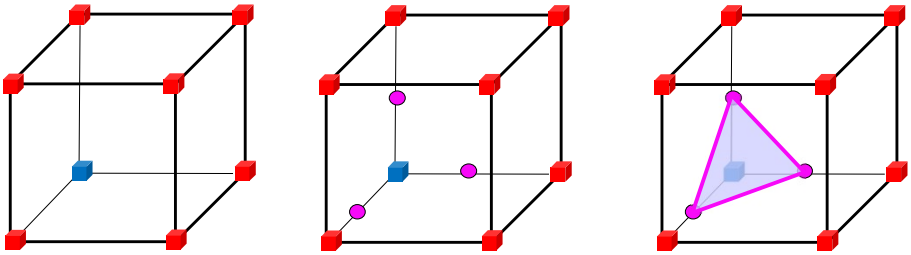
- ✓ Esempio di uno dei 256 casi




67

## Generalizzando a 3D: algoritmo marching cubes

✓Esempio di uno dei 256 casi



7 Voxel sopra soglia

1 Voxel sotto soglia

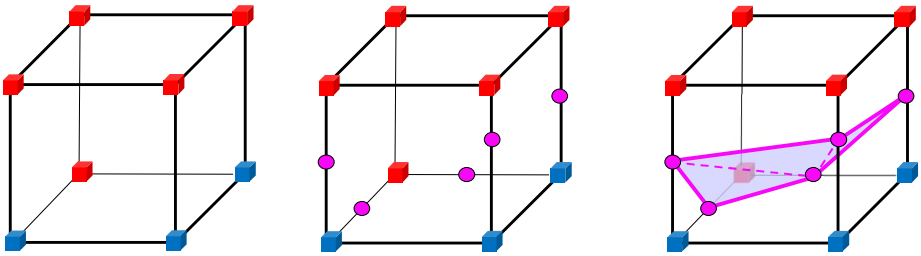
3 Intersezioni (calcolate sugli edge)

Connesse da un triangolo

68

## Generalizzando a 3D: algoritmo marching cubes

✓Esempio di uno dei 256 casi



5 Voxel sopra soglia

3 Voxel sotto soglia

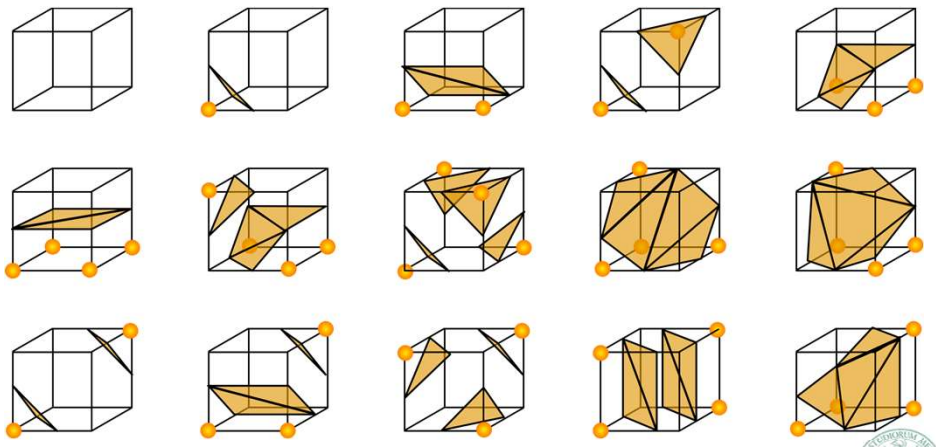
5 Intersezioni (calcolate sugli edge)

Connesse da tre triangoli

69

## Marching cube table

- ✓ Altri esempi
  - ⇒ Tutti gli altri sono analoghi a uno di questi 15, per simmetria



70

## Marching cubes

- ✓ Problema: efficienza.
  - ⇒ *Curse of dimensionality*: numero di cubi da analizzare cubico (con la risoluzione linare)
- ✓ 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:
  - ⇒ Trovo un primo cubo non-vuoto → lo processo
  - ⇒ Passo a processare i cubi non-vuoti vicini (ciascuna delle 256 configurazioni prevede quali dei sei cubi vicini saranno non vuoti)
  - ⇒ Continuo fino ad aver completato la mesh

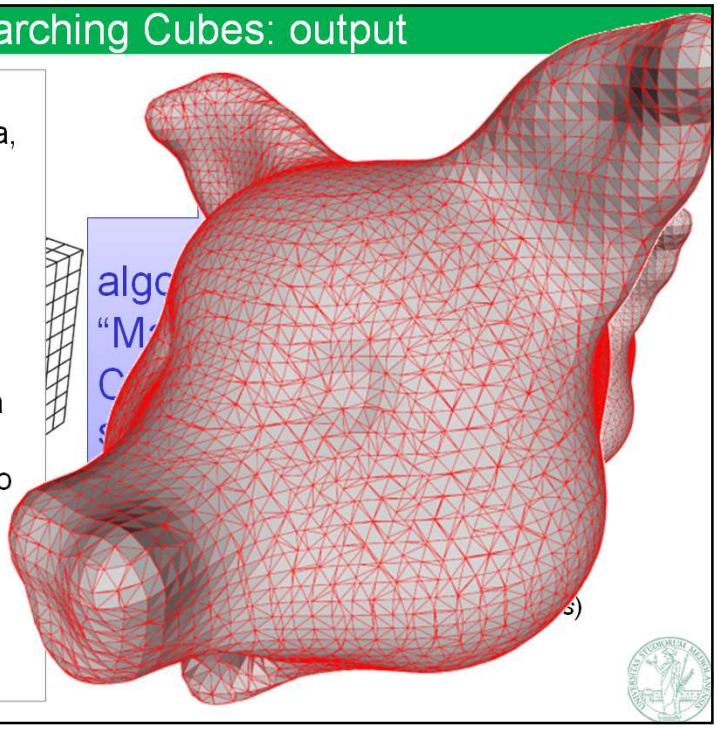
71

## Algoritmo Marching Cubes: output

Output:
una mesh chiusa,
two-manifold,
ben orientata.

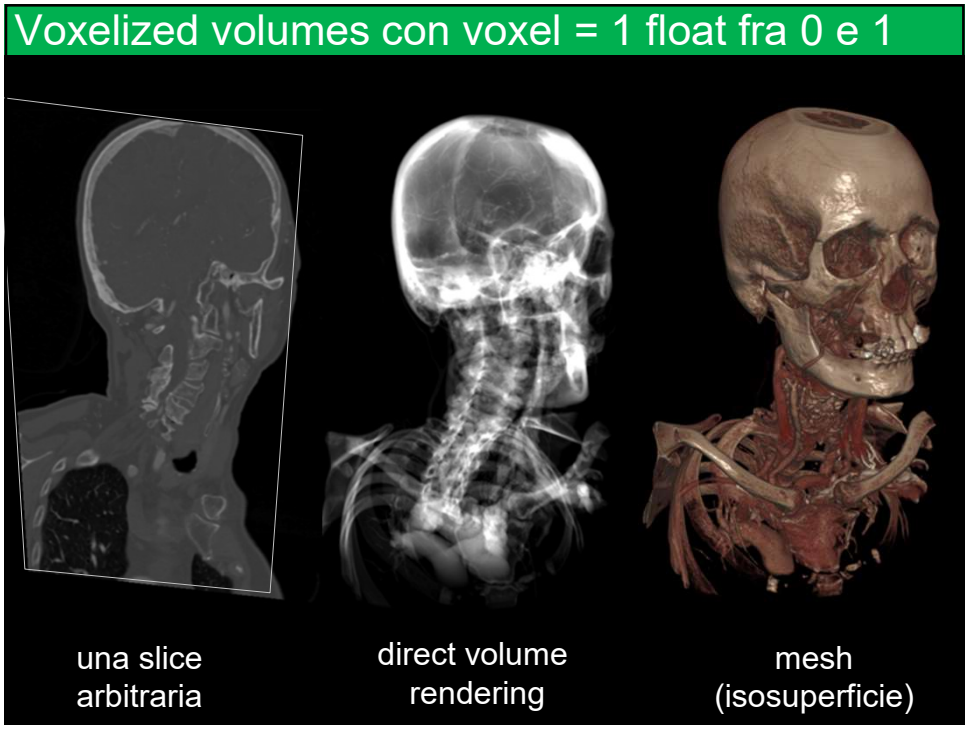
Ma tipicamente,
cattivo meshing:
molto irregolare,
triangoli di forma
lunga e stretta,
triangoli piccoli, o
degeneri

Un remeshing
può essere
necessario.



73

## Voxelized volumes con voxel = 1 float fra 0 e 1



una slice arbitraria

direct volume rendering

mesh (isosuperficie)

76