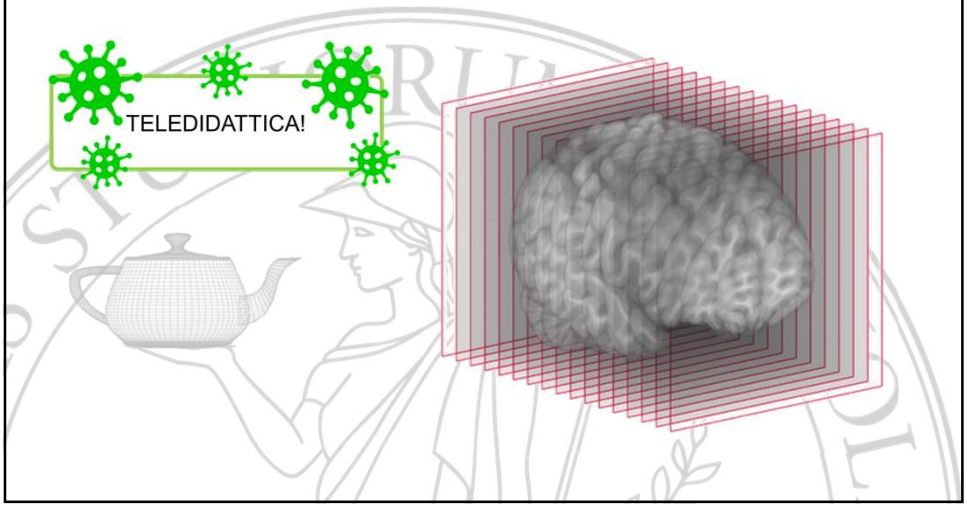
Marco Tarini - Computer Graphics 2020/2021
Università degli Studi di Milano

Rappresentazioni Volumetriche 2/3



31

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

		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** **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

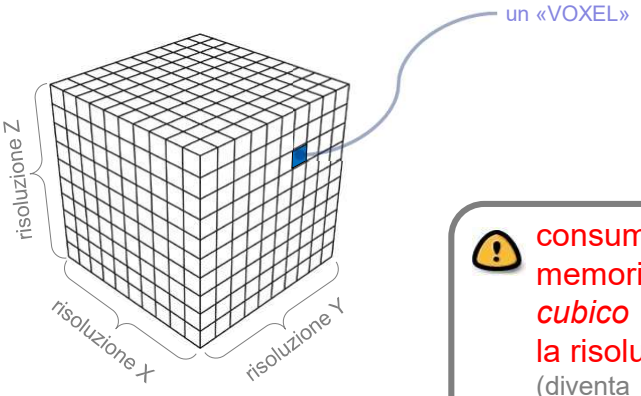
1. Discreti & regolari: dataset voxellizati
   ⇒analogo di un immagine rasterizzata, ma in 3D
   ⇒una griglia 3D regolare di voxel

2. Discreti & irregolari: mesh poliedrali
   ⇒Tetra-mesh, hexa-mesh
   ⇒insieme di poliedri adiacenti faccia a faccia

3. Continui: modelli impliciti
   ⇒rappresentazione basata su funzioni volumetriche
   ⇒superficie: luogo di zeri di questa funzione

33

Modello 3D a voxel (o voxellizato)



Griglia regolare 3D
(o lattice)

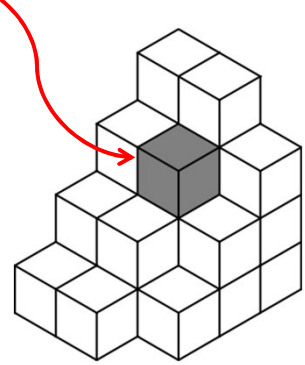
consumo memoria *cubico* con la risoluzione (diventa facilmente ingestibile)

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

34

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



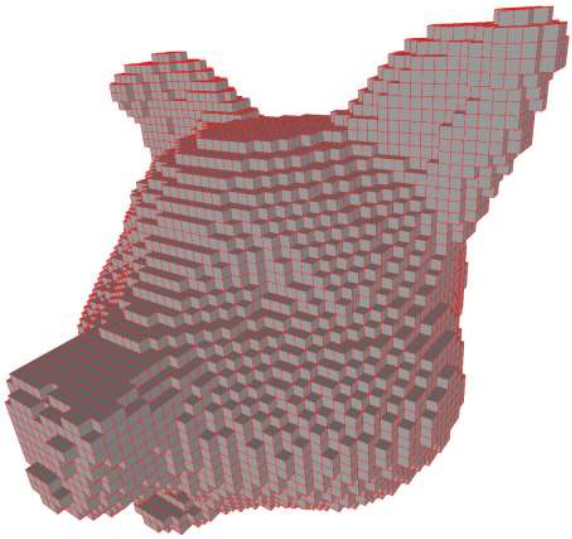
- ✓In codice:

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

Esempio in Java

35

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

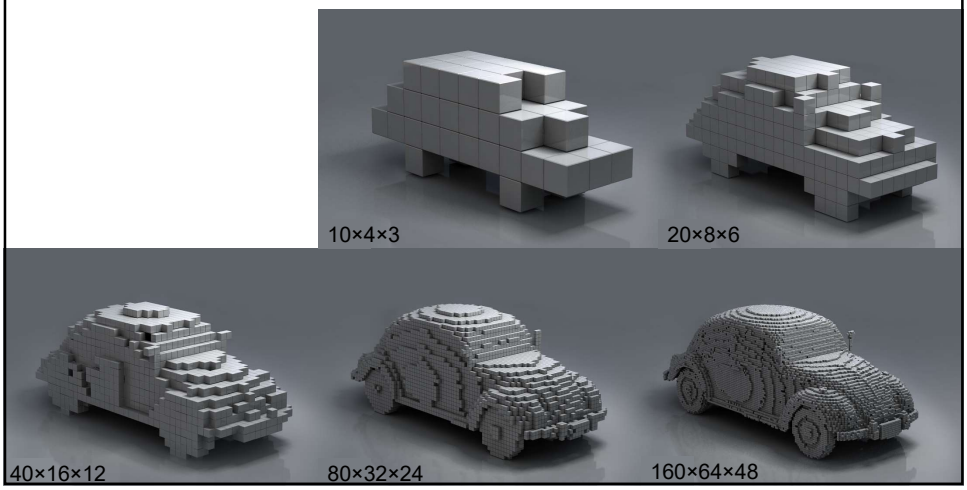


ogni *voxel* è pieno (1) o vuoto (0)

36

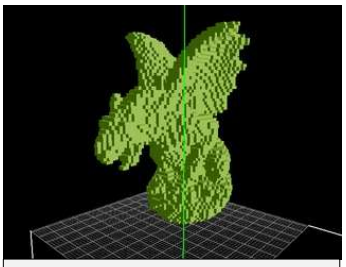
Risoluzione e costo in memoria

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



37

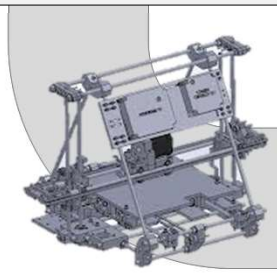
Voxel = 1 bit (pieno / vuoto)



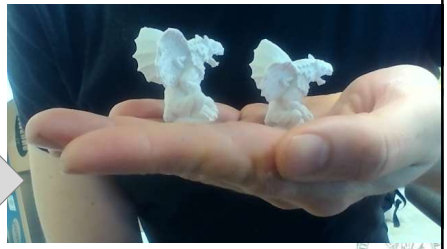
Voxelized dataset
Voxel: pieno/vuoto

✓Input naturale di (alcuni) **3D printing devices**

⇒Rapid prototyping devices per stampa “additiva”



3D printer

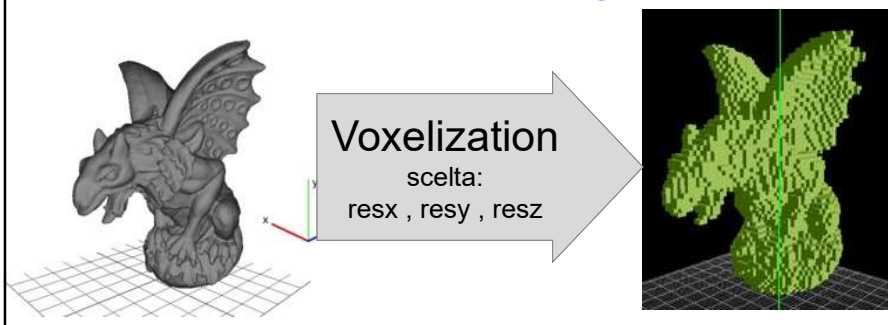


oggetti stampati

38

Voxelized Volumes:

- ✓Input naturale dei **3D printing devices**



Tri Mesh

ben orientata
e chiusa!

Voxelized volume

– 1bit × voxel

to printer

39

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»

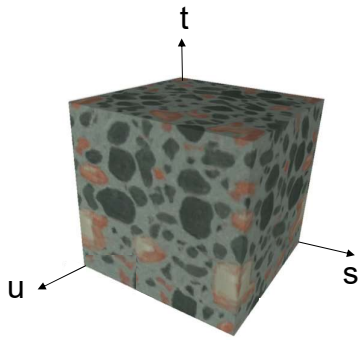
41

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

Esatto, proprio i terreni di minecraft ™

43

Se Voxel = 1 colore RGB: “texture volumetriche”



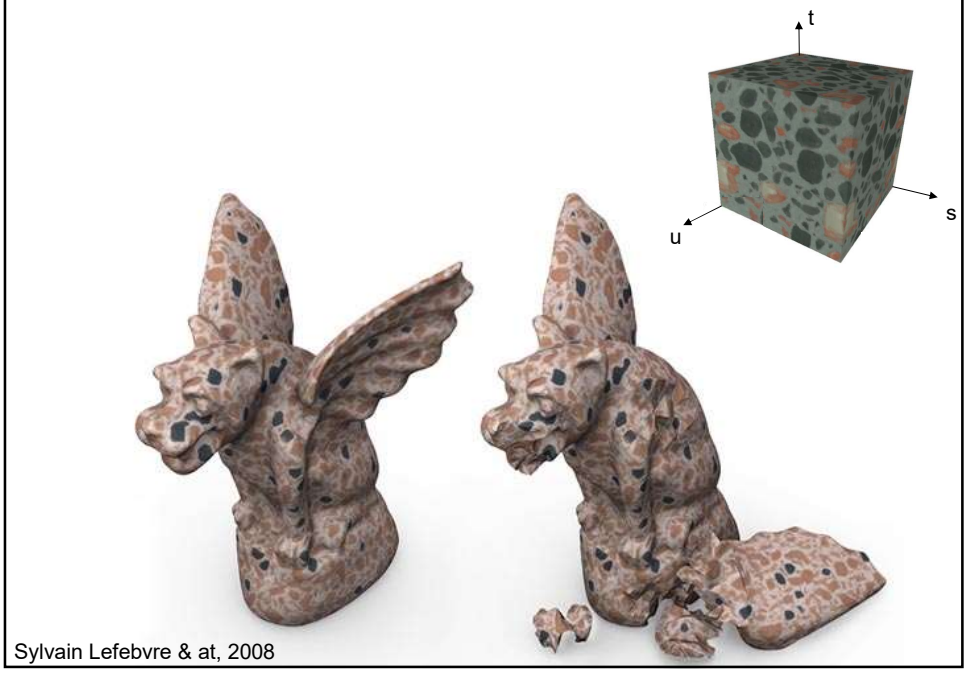
44

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?

45

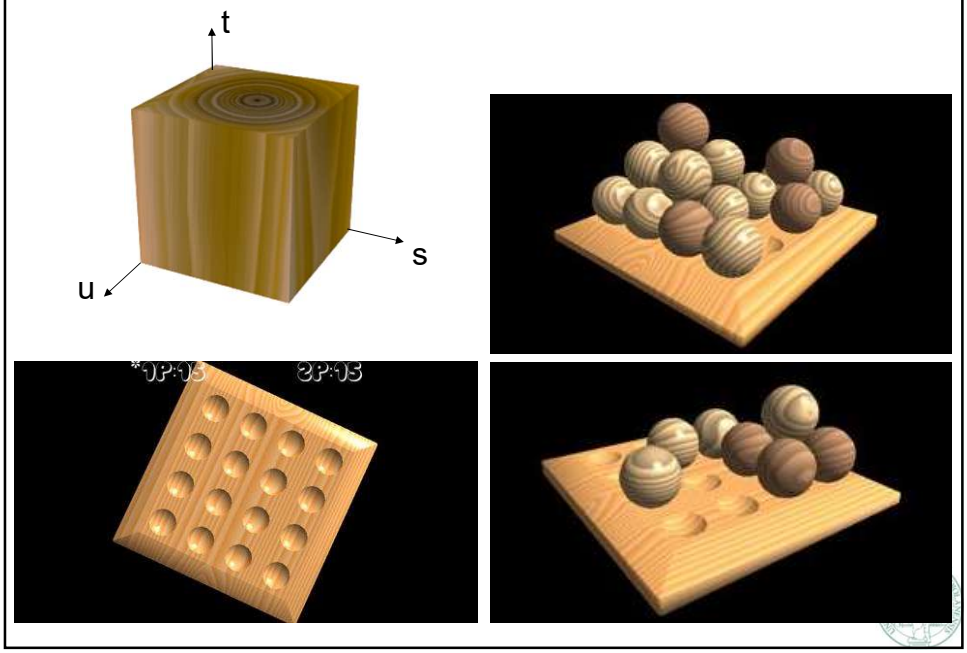
Un *colore* per voxel: “texture volumetriche”



Sylvain Lefebvre & at, 2008

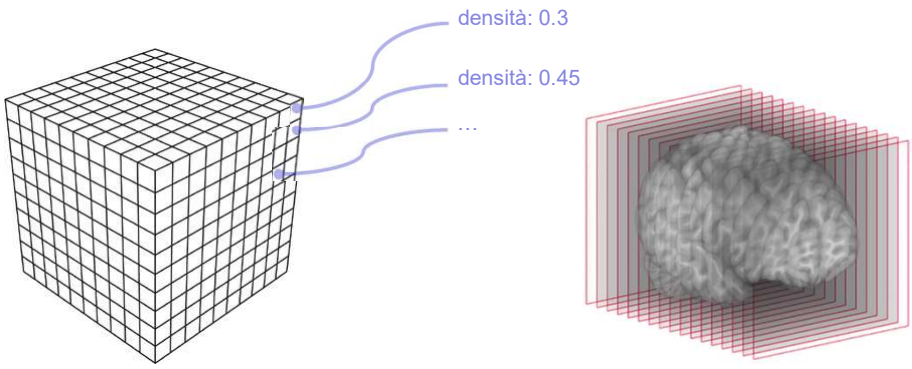
46

Volumetric Textures (o “solid Textures”)



47

Voxel = 1 scalare (es fra 0.0 e 1.0)

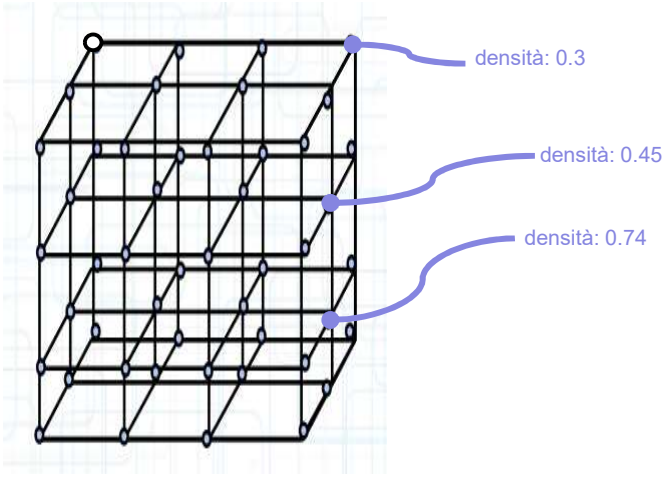


Esempio di File format: **DICOM**
(medicina)

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

48

Voxelized models: uno scalare per voxel

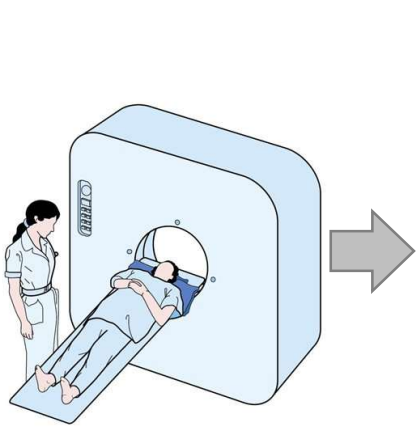


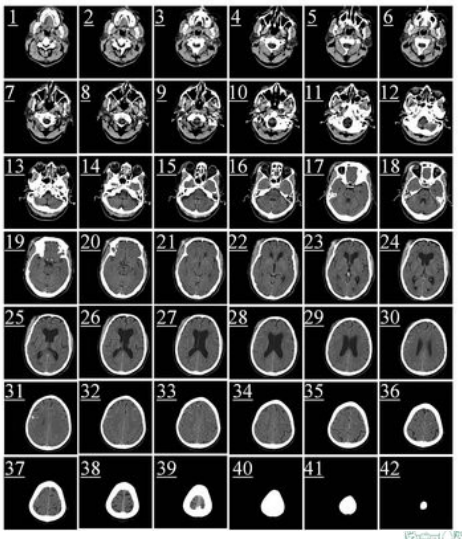

```
float volume[RES_X] [RES_Y] [RES_Z]
```

49

Se 1 voxel = 1 float (valori di densità)

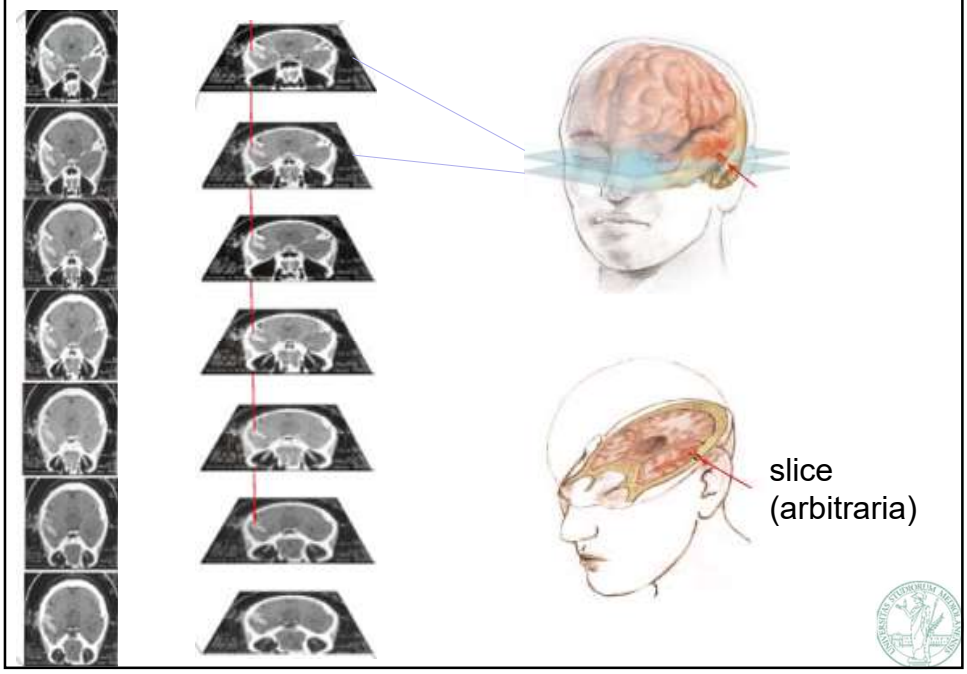
✓Output naturale di **CT scans**





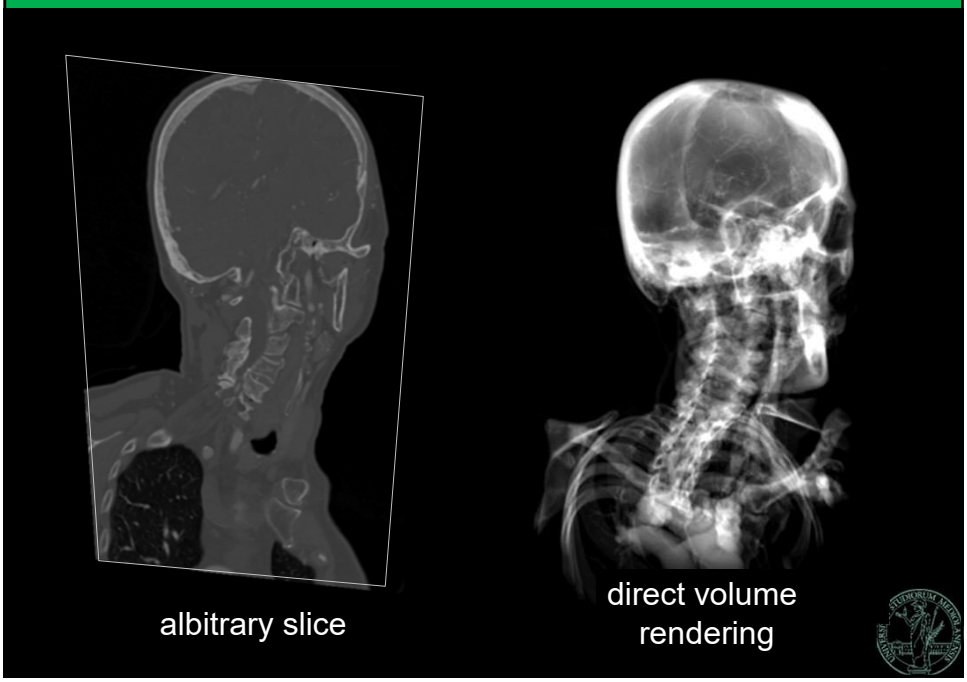
50

Se 1 voxel = 1 float (valori di densità)



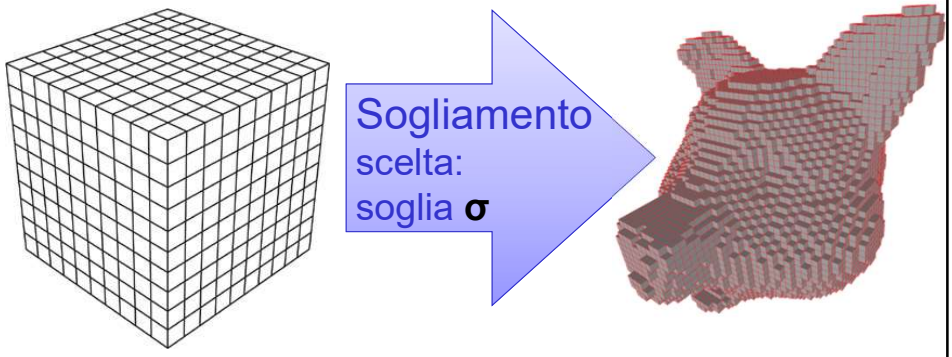
53

Voxelized volumes con voxel = 1 float fra 0 e 1



54

Soglia sulla densità



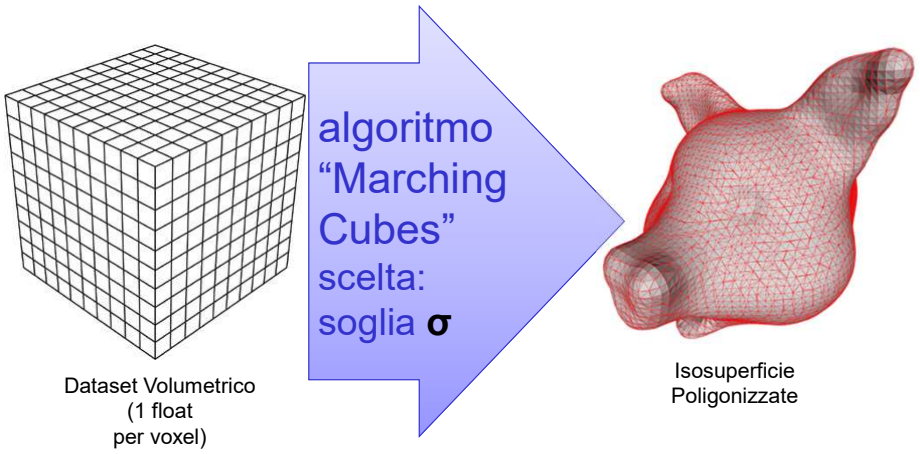
dataset voxelizzato
1 float per voxel
(tipcamente: densità)

dataset voxelizzato
dentro/fuori
per voxel

55

Poligonizzazione di un modello volumetrico

o segmentazione



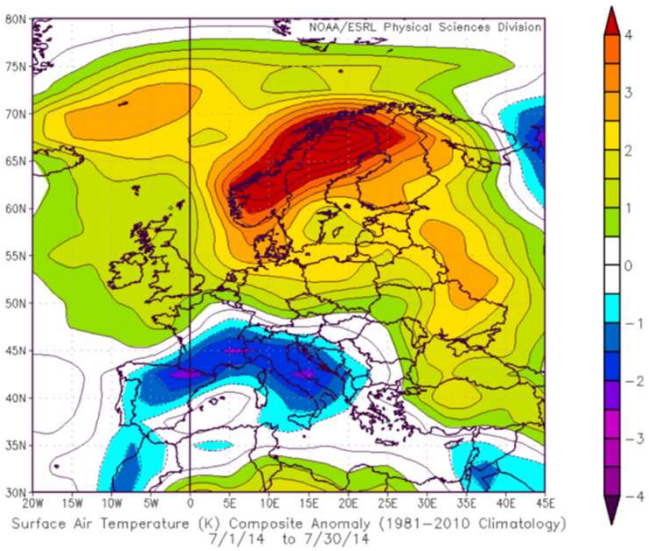
Dataset Volumetrico
(1 float
per voxel)

Isosuperficie
Poligonizzate

56

Esempio di isolinea

✓Isolinee di pressione



57

Isolinee e isosuperfici

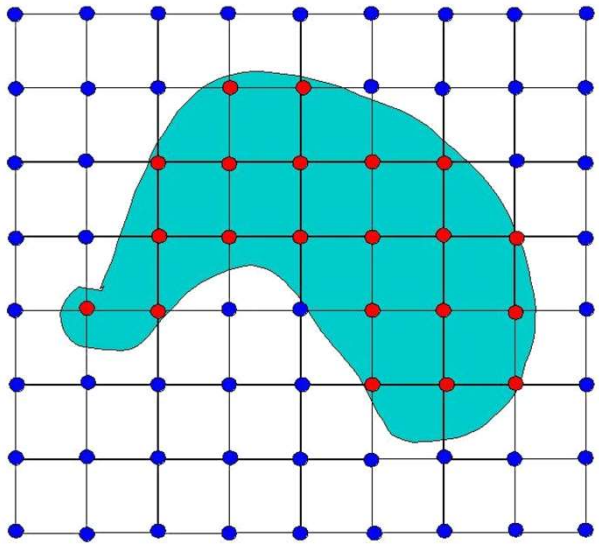
✓Su un piano (2D):
- ⇒ogni punto del piano ha un valore scalare (esempio: pressione, o altezza – height field)
- ⇒prendo tutta la regione 2D con valore > **σ**
- ⇒il bordo di questa regione è una linea … i cui punti hanno valore tutti **σ** e che racchiude tutti i valori di valore > **σ**
- ⇒è la « isolinea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

Algoritmo marching squares (per 2D)

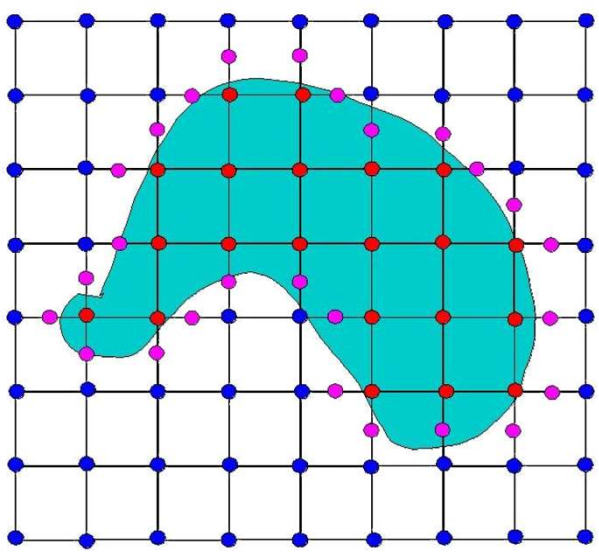
✓Sogliare voxels



60

Algoritmo marching squares (per 2D)

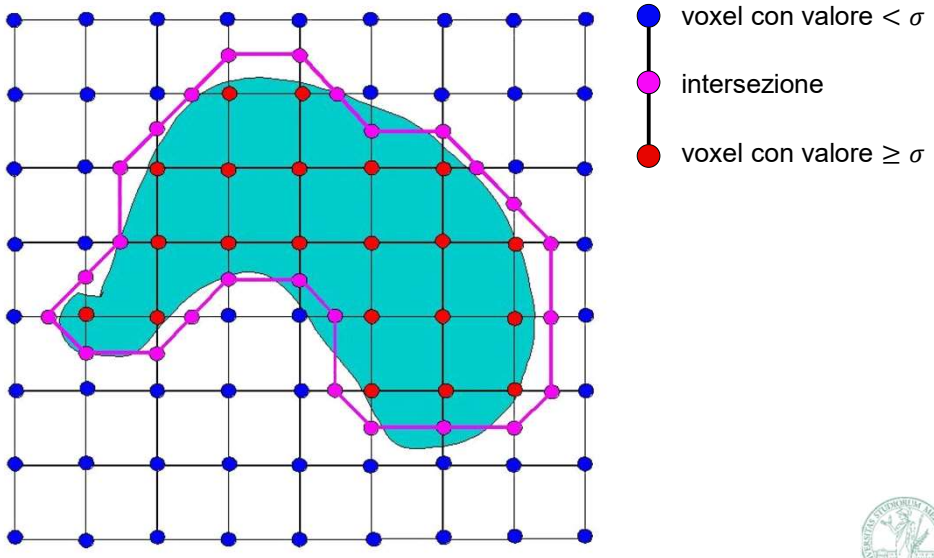
✓Trovare intersezioni



61

Algoritmo marching squares (per 2D)

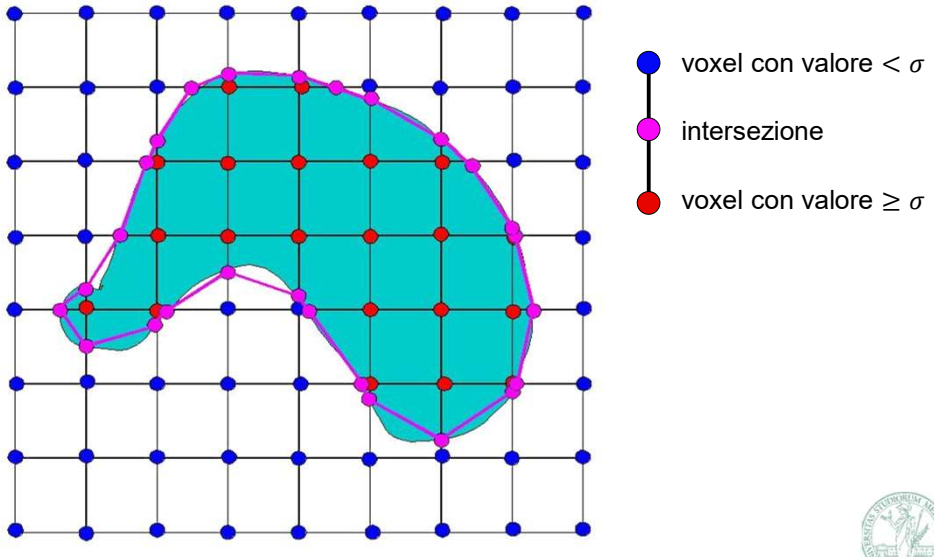
✓Unire intersezionicreando segmenti



62

Algoritmo marching squares (per 2D)

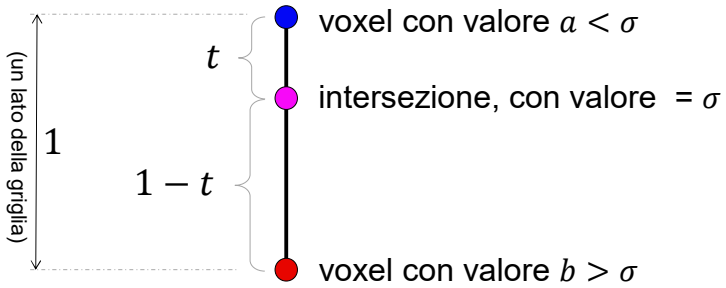
✓Variante: trovare intersezioni e unirle



63

Algoritmo marching squares (per 2D)

✓ Come trovare le intersezioni



✓ Ipotesi: segnale interpolato linearmente:

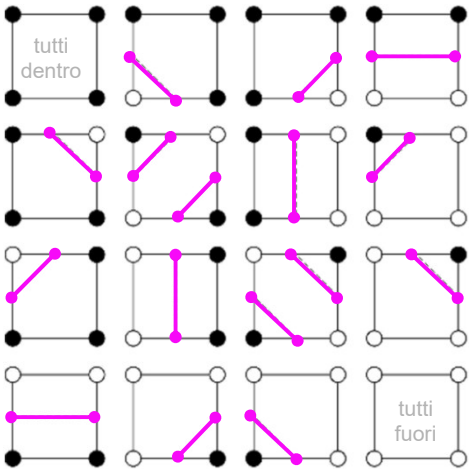
$$a(1-t) + b\,t = \sigma \quad \Leftrightarrow \quad t = \frac{\sigma - a}{b - a}$$

64

Algoritmo marching squares (per 2D)

Come trovare
i segmenti che connettono
le intersezioni?

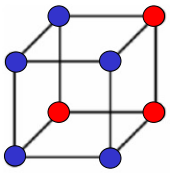
- ✓ Consideriamo un quadrato fra 4 voxel
- ✓ Ogni suo vertice è «dentro» $< \sigma$ o «fuori» $\geq \sigma$
- ✓ Per ogni combinazione, (e sono solo $2^4 = 16$) decido, una volta per tutte, i segmenti da costruire per unire le intersezioni
- ✓ ottengo questa tabella:



65

Generalizzando a 3D: algoritmo marching cubes

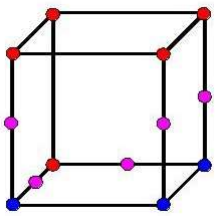
- ✓ Ogni voxel della griglia è dentro o fuori
  - ⇒ valore > o < della soglia prescelta
- ✓ Ogni edge della griglia (orientato lungo la X, Y o Z), che connetta un vertice dentro ad uno fuori, ha una intersezione con l'isosuperficie cercata
  - ⇒ la si trova trovata come nel caso 2D
  - ⇒ per ciascuna intersezione, creo un vertice della mesh
  - ⇒ ho ottenuto la **geometria** della mesh!
- ✓ Come ottengo la **connettività**?
  - ⇒ Scompongo la griglia in cubetti 1x1x1
  - ⇒ 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

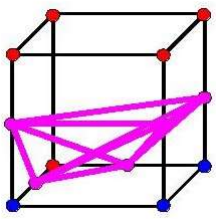


66

Generalizzando a 3D: algoritmo marching cubes

- ✓ Esempio di uno dei 256 casi

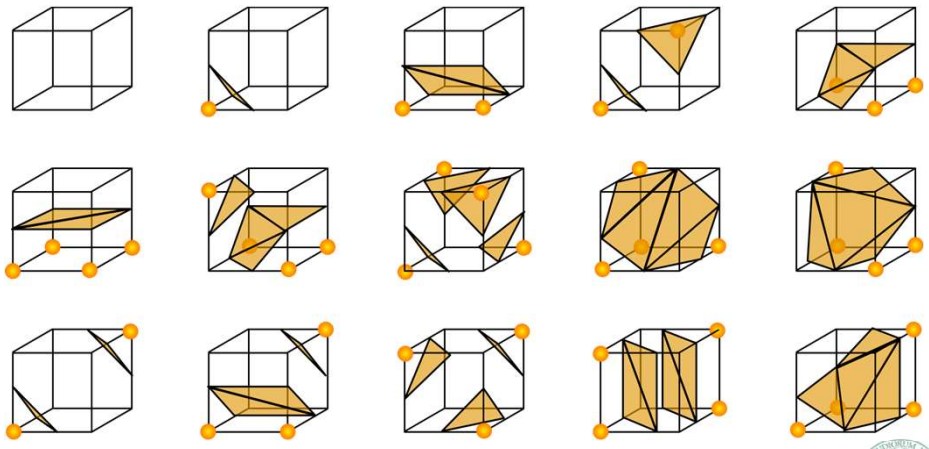




67

Marching cube table

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



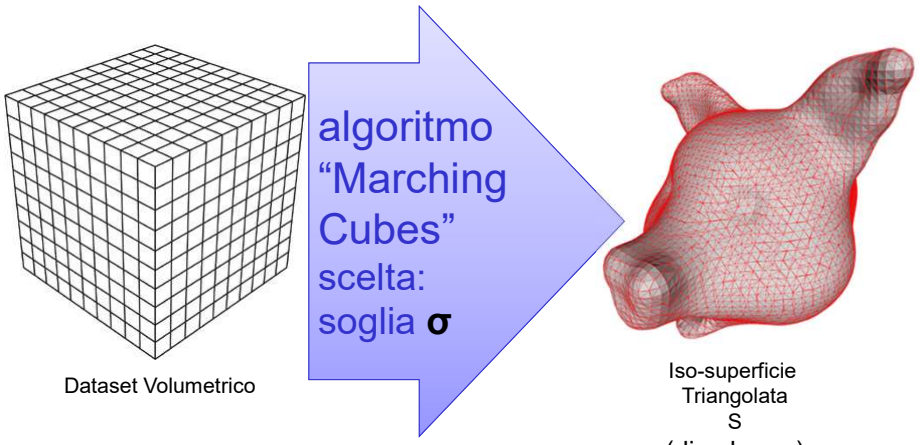
68

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 sono vuoti (niente intersezioni sugli spigoli, ne' facce all'interno)
- ✓Far "marciare" i cubi:
  - ⇒Trovo un cubo non vuoto → lo processo
  - ⇒Passo a processare i cubi non vuoti vicini
  - ⇒Continuo fino ad aver completato la mesh

69

Da modello voxelizzato a Mesh poligonale

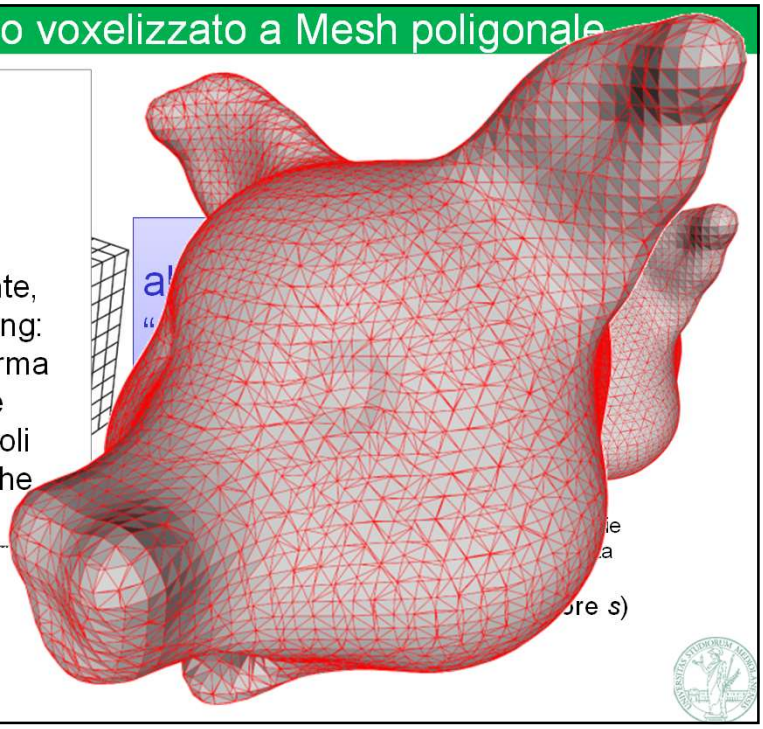


70

Da modello voxelizzato a Mesh poligonale

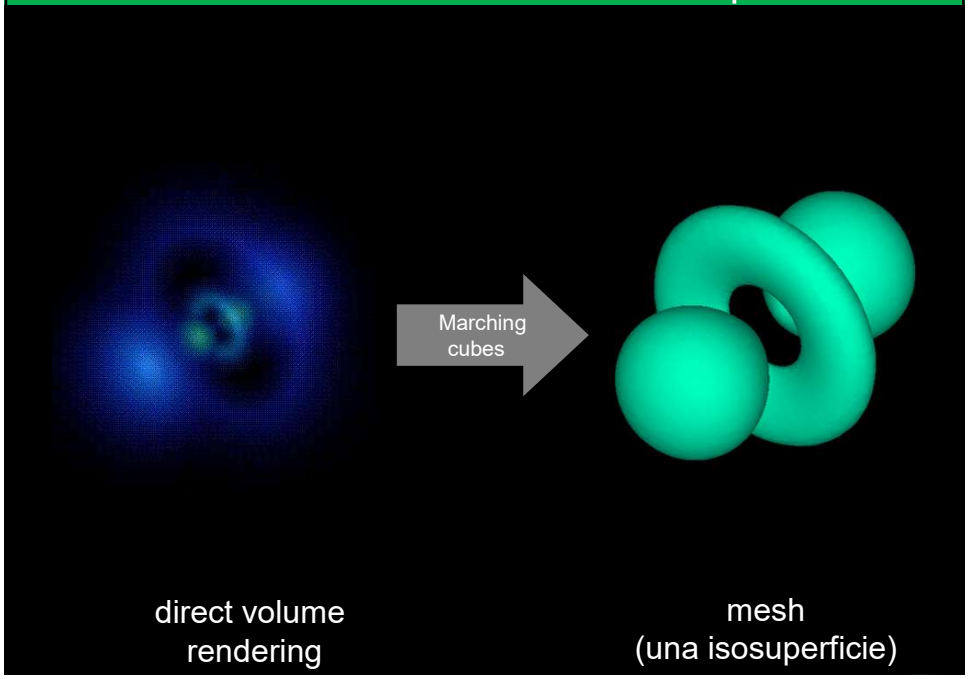
Output:
mesh chiusa,
two-manifold,
ben orientata

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



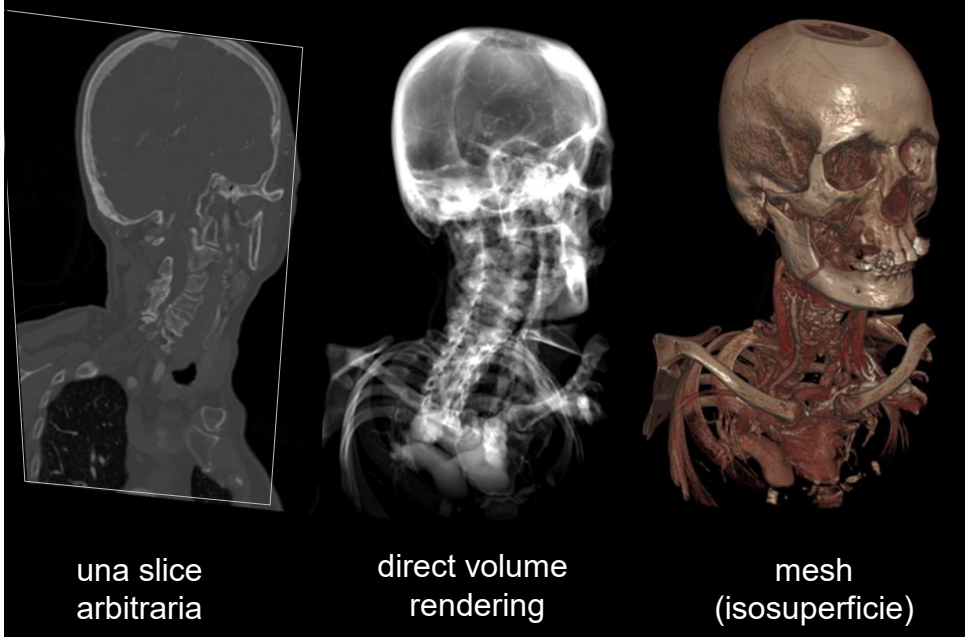
71

Da dataset volumetrico a mesh: esempio



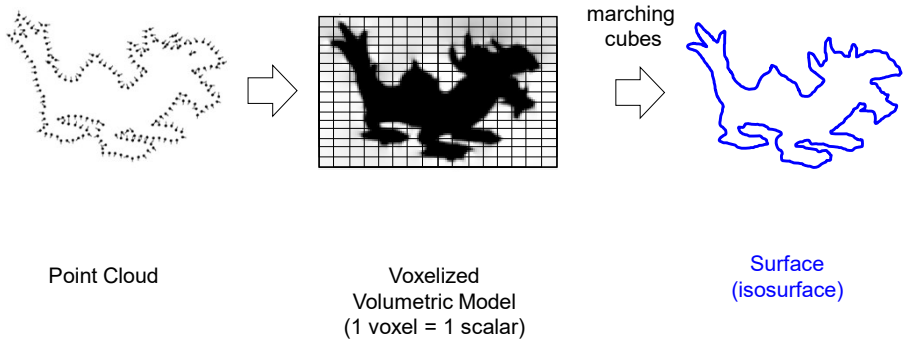
72

Voxelized volumes con voxel = 1 float fra 0 e 1



74

Strutture volumetriche come dataset intermedio



Images by Micheal Kazhdan

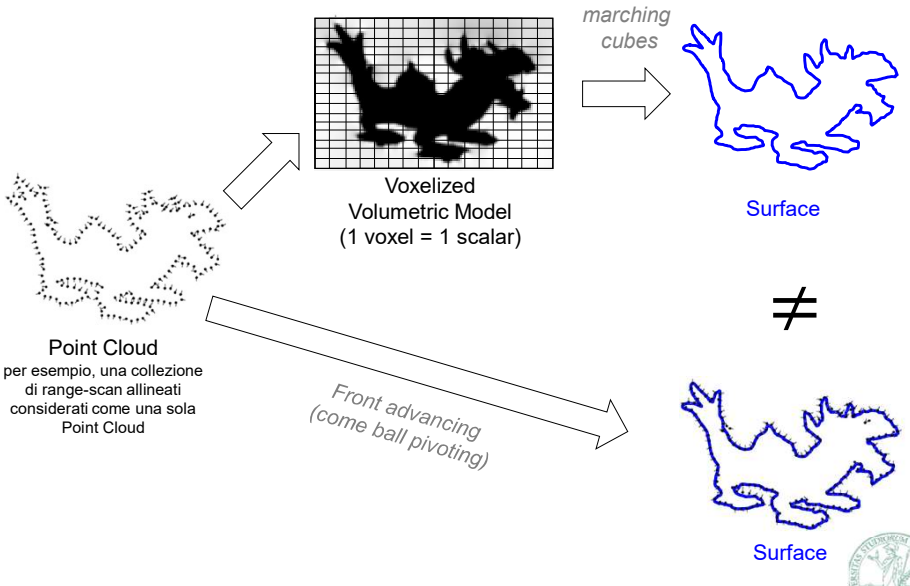
75

Strutture volumetriche come dataset intermedio

- ✓ Un modo comune di convertire una point cloud (e altro) in una mesh 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 isosuperficie (marching cubes)
- ✓ Nel passo 2: ogni punto della nuvola di posizione *p* e normale *n* «vuole» che…
  - ⇒ che il valore della funzione in *p* sia 0
  - ⇒ che il gradiente della funzione in quella posizione sia allineato a *n*

76

Quale metodo è migliore?



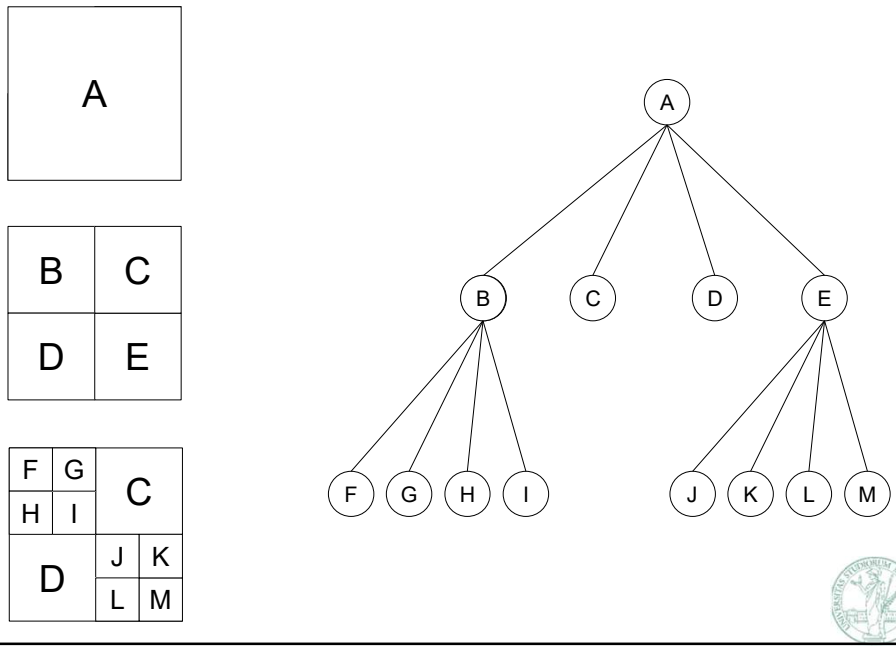
77

Dataset volumetrici a griglia adattivi

- ✓ Il problema della «curse of dimensionality» è che la risoluzione di un dataset di voxel non è *adattiva*
- ✓ Esistono strutture dati più compatte, perchè dotate di risoluzione adattiva
- ✓ Una delle più diffuse è l'oct-tree
  - ⇒ Vediamo prima una sua versione 2D: il quad-tree

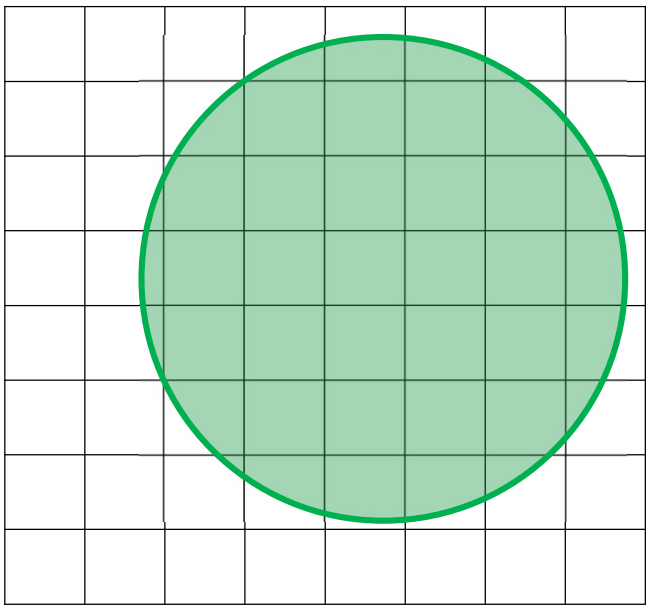
78

Quad Tree



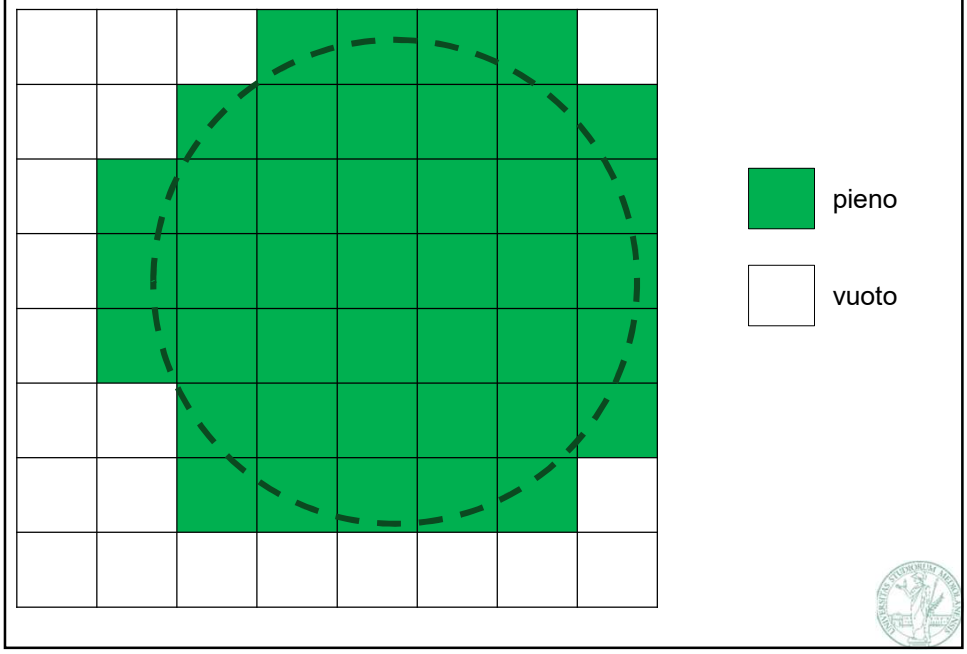
79

Esempio. 1/4: la forma da rappresentare



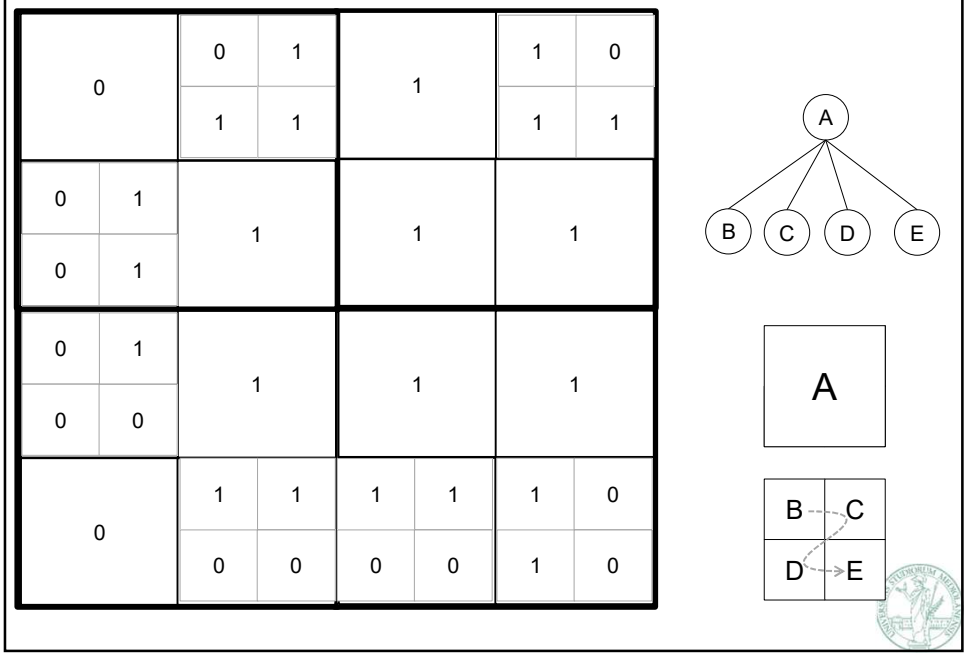
82

Esempio. 2/4: “voxellizzazione” (ma in 2D)



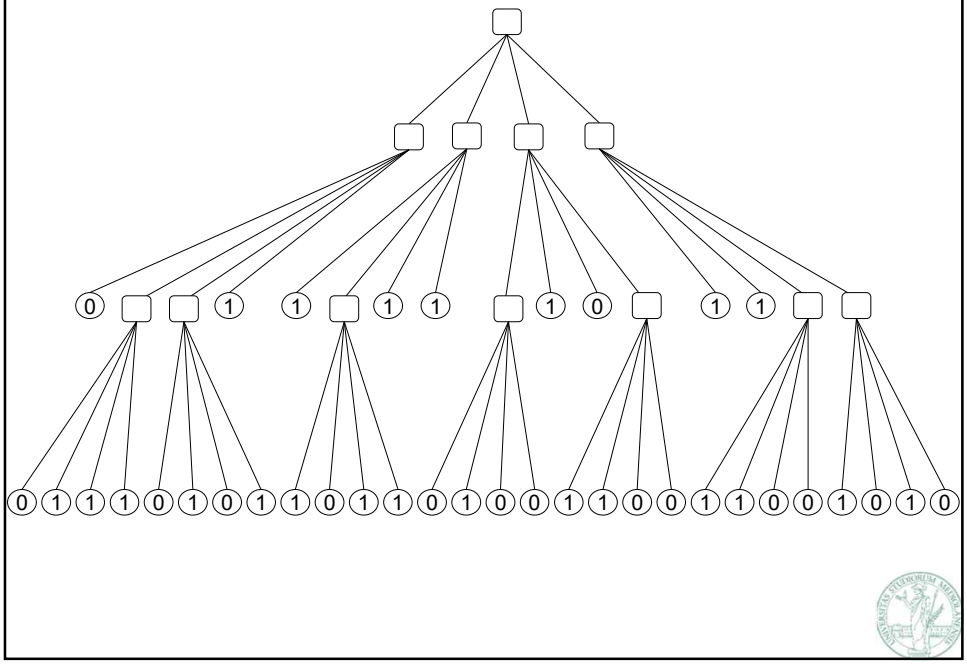
83

Esempio. 3/4: quad-tree (scomposizione)



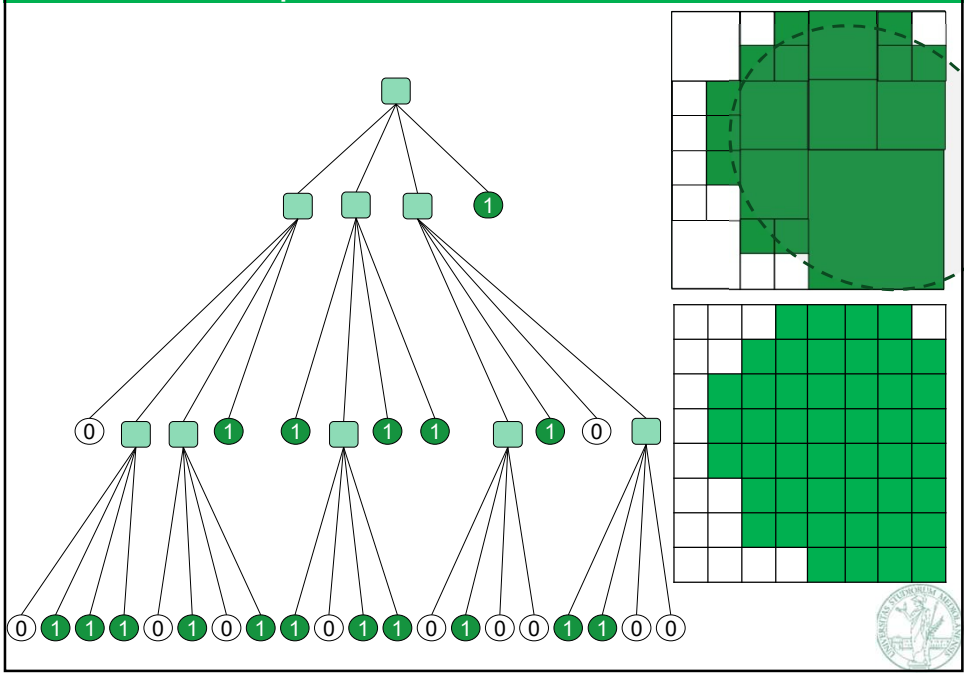
84

Esempio. 4/4: quad-tree (l'albero)



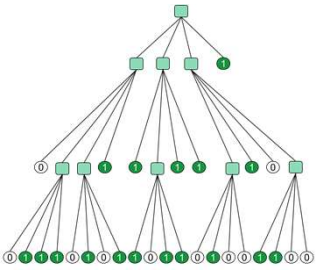
85

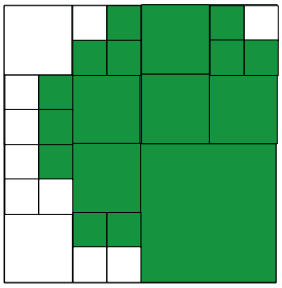
Un altro esempio



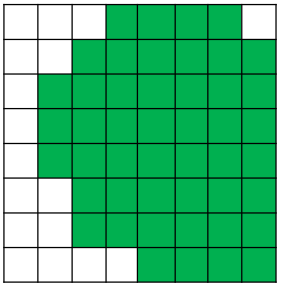
86

Un altro esempio





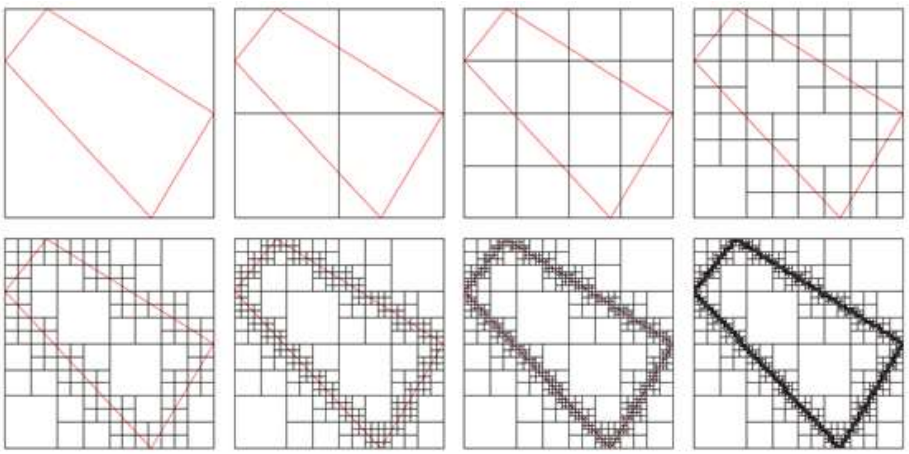
Come quad-tree:
solo 28 foglie
(+9 nodi interni)



Come volume di
voxel equivalente:
8x8 = 64 voxel

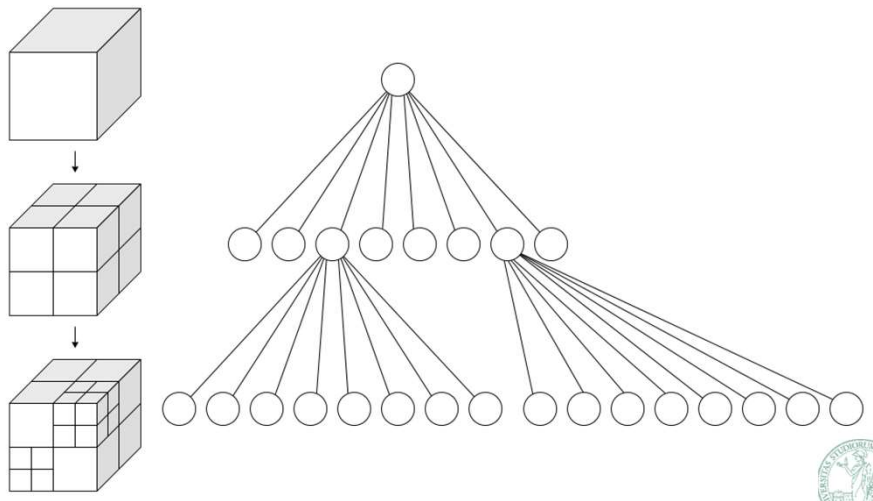
87

Quad tree: esempio che codifica segmenti



88

Oct Tree: concetto simile, per 3D



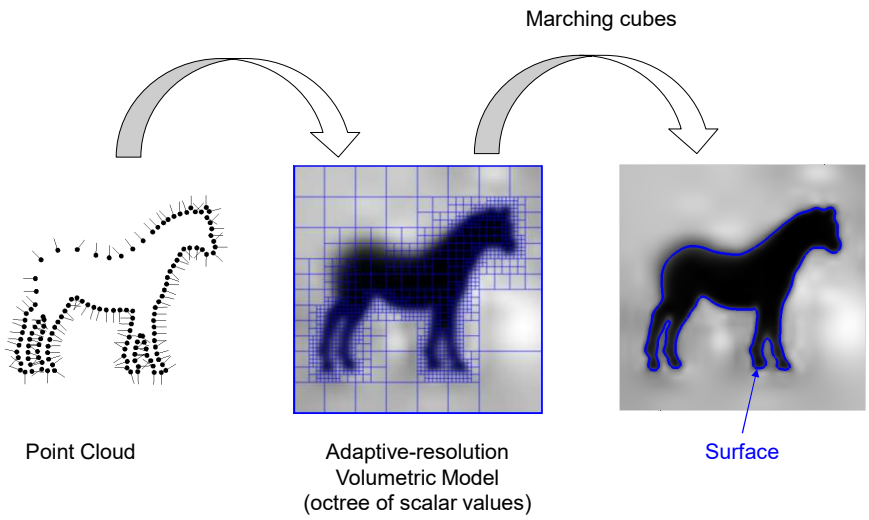
89

Oct-tree: sommario

- ✓ Struttura ricorsiva, ad albero
- ✓ Qui usata per memorizzare efficientemente un volume voxellizzato a 1 bit
  - ⇒ Ci sono molti altri usi in computer graphics
- ✓ Ogni nodo è associato ad un cubo nel volume
  - ⇒ Radice: associato all'intero dataset
  - ⇒ Nodo interni: blocchi che contengono sia 1 che 0
    Hanno 8 figli, uno per ciascun ottavo del padre
  - ⇒ Foglie: rappresentano aree uniformi, di tutti 1 o 0
- ✓ Numero totale nodi: circa proporzionale all'estensione del bordo dell'oggetto rappresentato
  - ⇒ è quindi quadratico, non cubico, con la risoluzione!

90

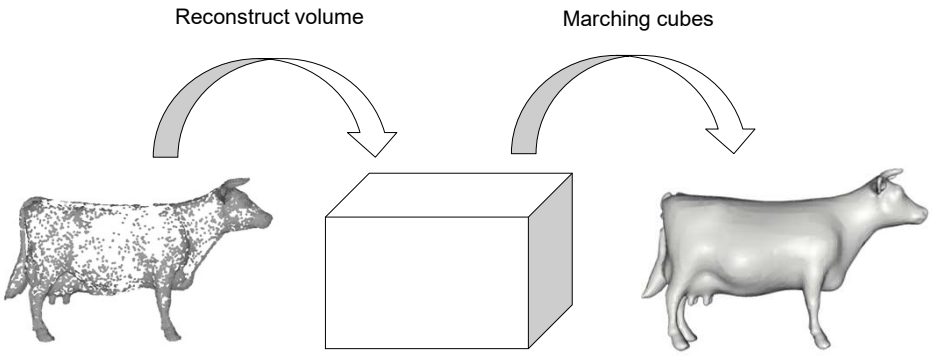
Poisson Surface Reconstruction



Images by H. Hoppe

91

Poisson Surface Reconstruction



Images by H. Hoppe

93