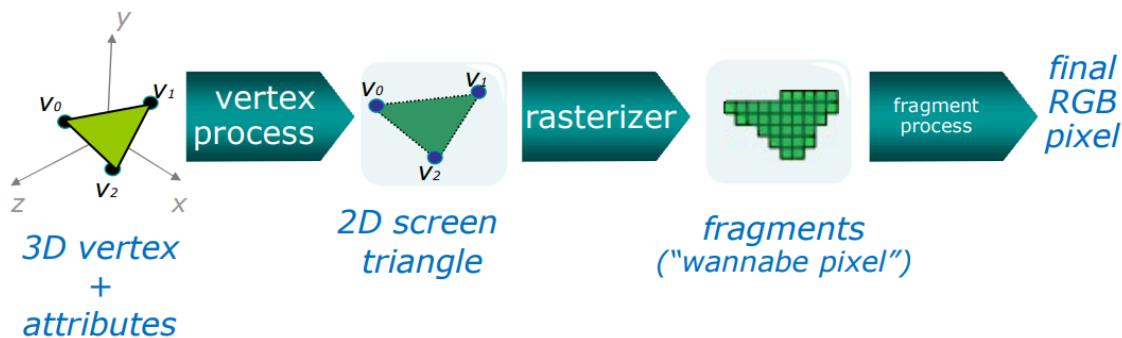


Lezione 8 03/04/2025

2D e 3D Data

Quando partiamo da qualcosa di digitale, sappiamo tutto di questo triangolo nel mondo 3D. Poi c'è un processo che genera il rendering finale (che vedremo), poi una serie di operazioni che rendono questo processamento della geometria un'immagine rappresentabile tramite pixel, e poi abbiamo un altro processo che ci dà l'effetto visivo che abbiamo voluto attribuire al triangolo.



Mondo reale

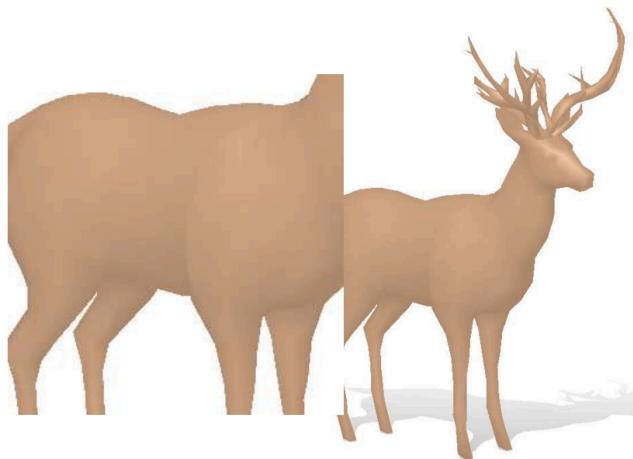
Il mondo reale è **non rigido, deformabile**. Non sappiamo la posa in cui troveremo questo cervo.

- ◆ Several elements we interact with are deformable

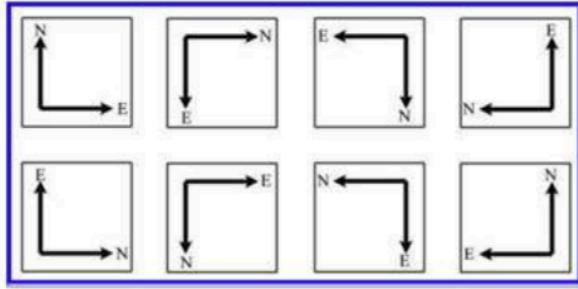


Il mondo reale è **continuo**, la superficie del cervo ha valori ovunque. Non ci sono punti di discontinuità. Questo invece è molto difficile da processare per un elaboratore, dobbiamo avere una discretizzazione di solito.

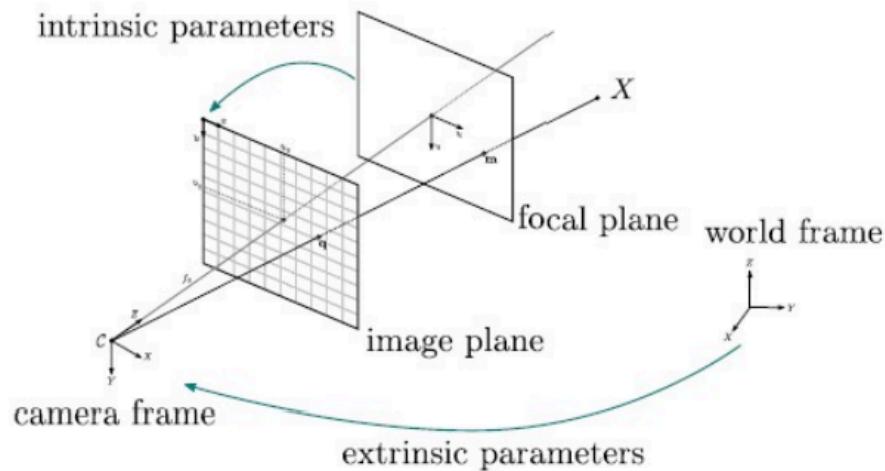
- ◆ The world is continuous



Sulle immagini non usiamo il **sistema di riferimento** centrato al centro, ma invece questo.

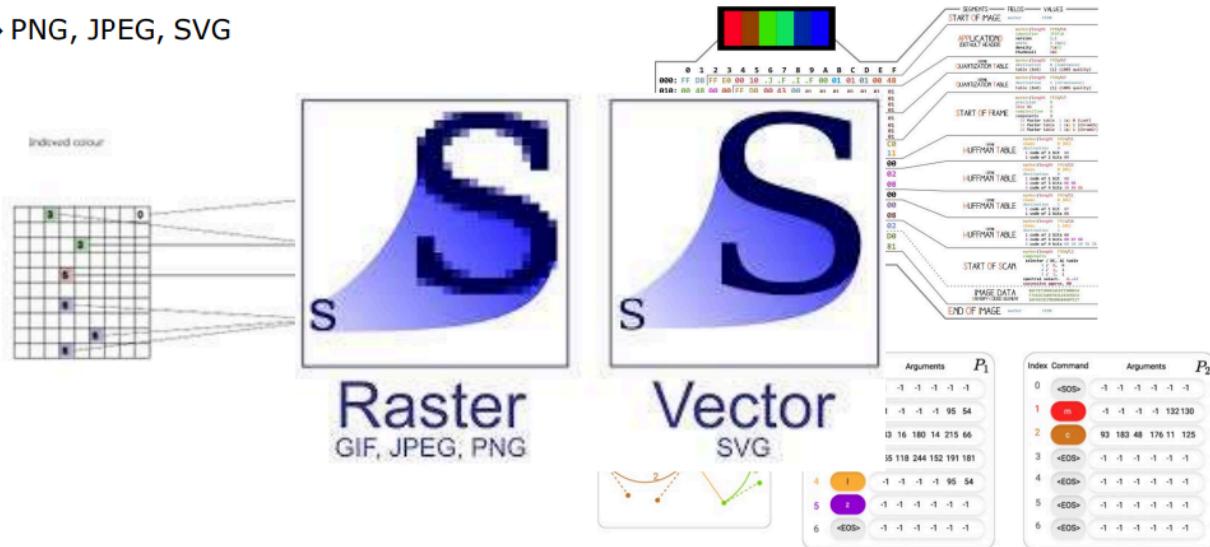


La camera calibration fa capire qual è la trasformazione che porta il sistema di riferimento mondo nel sistema di riferimento immagine (non lo vediamo).



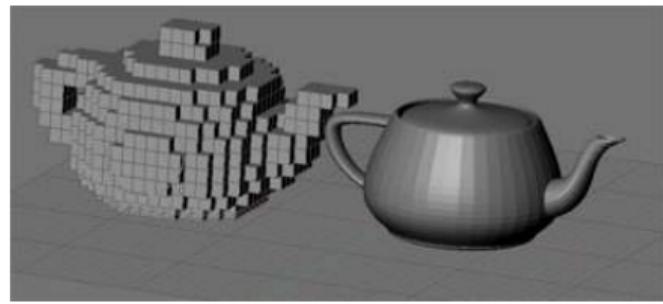
le immagini **vettorizzate** nascono sempre da una discretizzazione, ma si guarda all'immagine in modo continuo usando degli strumenti. Quelle **rasterizzate** sono quelle fatte da pixels.

◆ PNG, JPEG, SVG

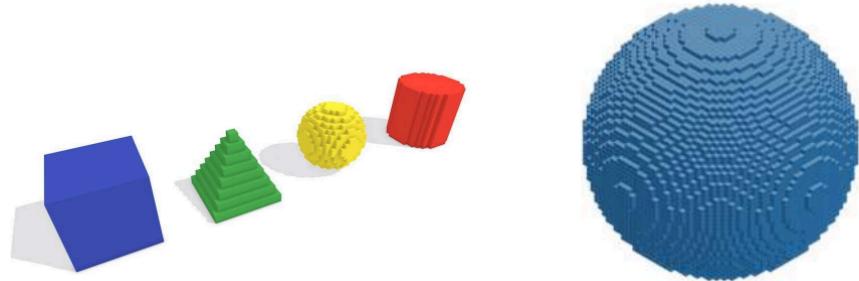


3D Modeling

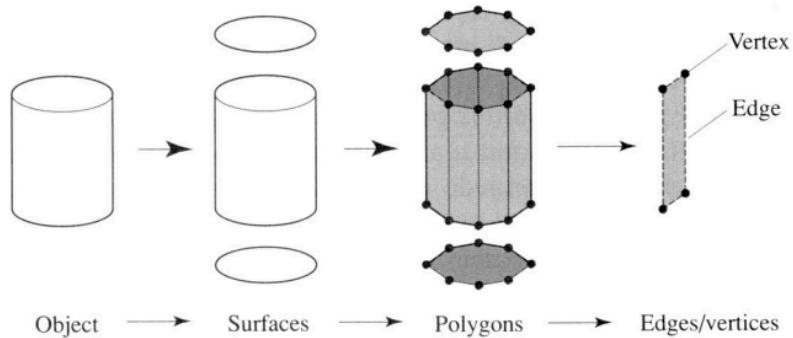
Come rappresentiamo i dati 3D? Innanzitutto bisogna fare una scelta, vogliamo rappresentare anche il **volume** dell'oggetto, oppure rappresentare solo il **contorno**.



Questa è la cosa più vicina ai pixel che posso utilizzare per rappresentare una superficie/volume



Nella **boundary representation** discretizziamo, creiamo una mesh scomponendo l'oggetto in una serie di poligoni (facce)

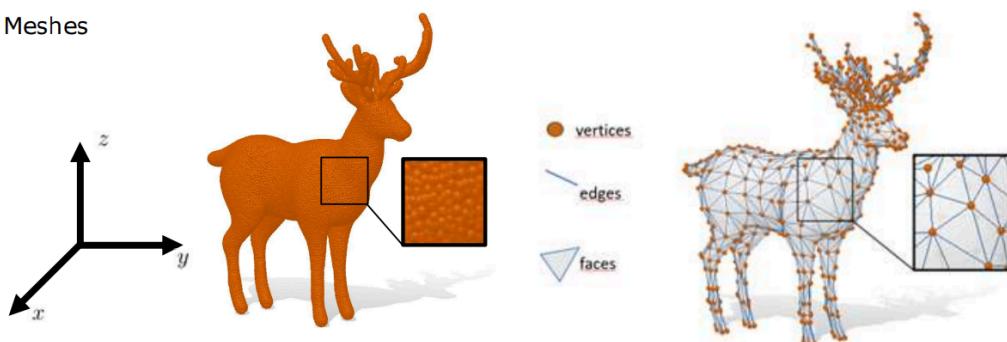


◆ Different representation are possible!

- Continuous surfaces
 - SDF
 - Implicit representations
- Point clouds
 - With ior without normals
- Meshes
 - Traingular meshes
 - Quad-meshes
 - Polygonal meshes
- Volumetric
 - Voxels
 - Tetrahedral meshes

Quelle più usate sono **point cloud** e **mesh**.

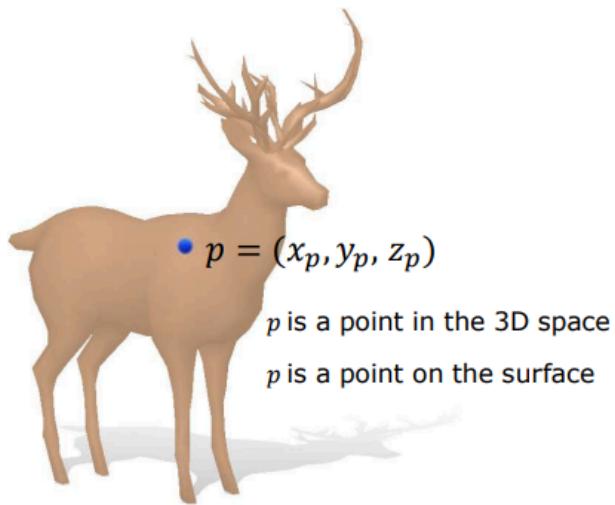
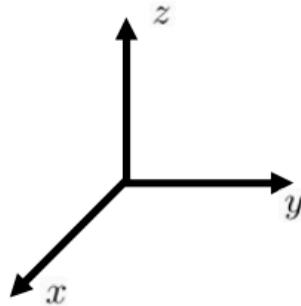
- ◆ Point clouds
- ◆ Meshes



Ogni volta che abbiamo un oggetto 3D dobbiamo piazzare un punto di riferimento.

◆ Different reference system

Arbitrary reference system x, y, z

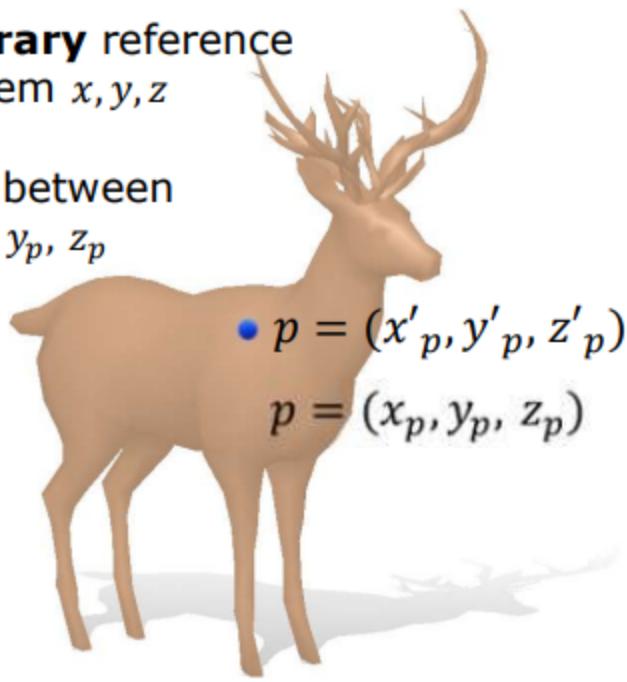
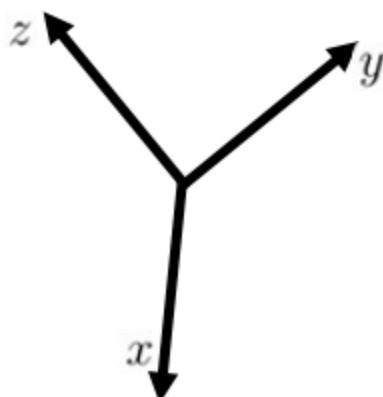


Fisso un sistema di coordinate. p è un punto dello spazio 3D, ma è anche un punto che appartiene alla superficie. Quindi dichiaro le coordinate, che mi dicono qualcosa sia della superficie che dello spazio 3D.

Se cambio il sistema di riferimento, cambiano i valori che attribuisco allo stesso punto.

...an **arbitrary** reference system x, y, z

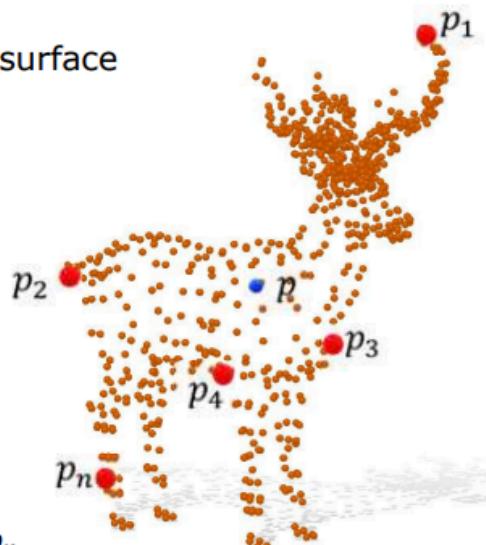
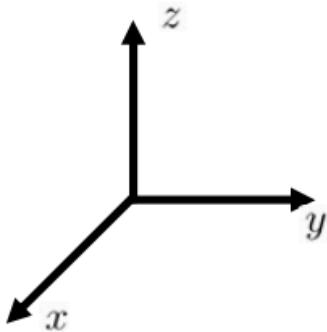
What is the relation between x'_p, y'_p, z'_p and x_p, y_p, z_p



Potrei scegliere una quantità infinita di punti sull'oggetto, ma non possiamo avere un quantità infinita, e non posso prenderne troppi altrimenti non è più possibile processarli con un pc.

Di solito selezioniamo un sottoinsieme di punti, che siano abbastanza per capire che oggetto è l'oggetto 3D.

Given a set of n points sampled on a surface
anda fixed reference system



We can order the points $p_1, p_2, p_3, p_4, \dots, p_n$

Quindi ora possiamo organizzare i punti di questa collezione di punti, dando un'**ordinamento**.

Possiamo mettere i dati in una matrice V , che è **piena di dipendenze**:

- dipende dal sistema di riferimento
- l'ordine è scelto arbitrariamente
- avevamo deciso di selezionare un certo numero di punti, non tutti

Queste dipendenze creano problemi.

Meshe

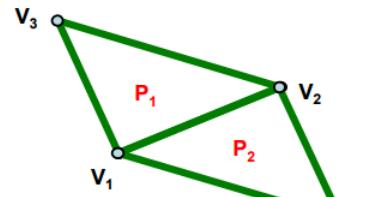
Le mesh sono delle strutture composte da **vertici** (punti in R^3), gli **edge** che sono segmenti che collegano i vertici, e i **poligoni** che sono forme chiuse composte da edge che si connettono.

La mesh più popolare è quella **triangolare**, perchè i triangoli sono i poligoni più semplici. Quello che possiamo comporre con un poligono, possiamo anche comporlo con una serie di triangoli.

Possiamo rappresentare la mesh triangolare come una composizione di punti, tenendo conto però anche delle composizioni in triangoli. Ci sono diverse rappresentazioni possibili:

◆ Explicit representation

$$\begin{aligned} P_1 &= \{(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3)\} \\ P_2 &= \{(x_1, y_1, z_1), (x_4, y_4, z_4), (x_2, y_2, z_2)\} \\ &\dots \end{aligned}$$

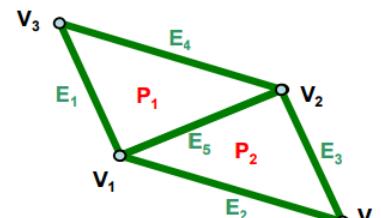


◆ Vertices' list

$$\begin{array}{ll} V = \{V_1, V_2, V_3, V_4\} & V_1 = (x_1, y_1, z_1) \\ & V_2 = (x_2, y_2, z_2) \\ P_1 = \{V_1, V_2, V_3\} & V_3 = (x_3, y_3, z_3) \\ P_2 = \{V_1, V_4, V_2\} & V_4 = (x_4, y_4, z_4) \end{array}$$

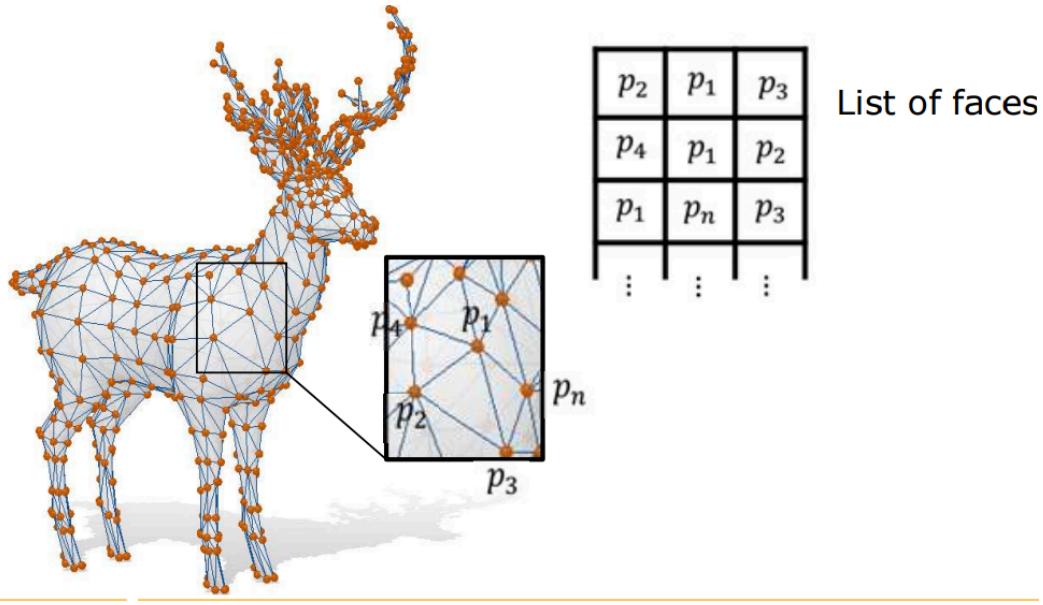
◆ Edges' list

$$\begin{array}{ll} V = \{V_1, V_2, V_3, V_4\} & \\ E_1 = \{V_3, V_1, P_1, \lambda\} & V_1 = (x_1, y_1, z_1) \\ E_2 = \{V_1, V_4, P_2, \lambda\} & V_2 = (x_2, y_2, z_2) \\ E_3 = \{V_4, V_2, P_2, \lambda\} & V_3 = (x_3, y_3, z_3) \\ E_4 = \{V_2, V_3, P_1, \lambda\} & V_4 = (x_4, y_4, z_4) \\ E_5 = \{V_1, V_2, P_1, P_2\} & \\ \\ P_1 = \{E_1, E_5, E_4\} & \\ P_2 = \{E_2, E_3, E_5\} & \end{array}$$



Nell'ultima, un edge può essere condiviso da una faccia o 2.

La rappresentazione più usata è quella della **lista di facce**.

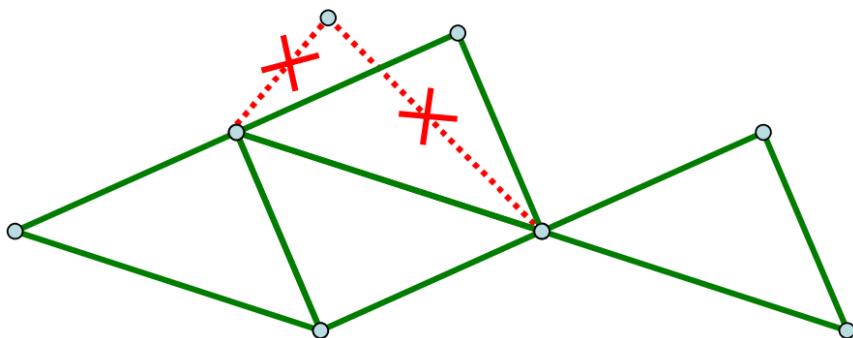


La sequenza dice che se parto da p_2 , avrò un edge che lo collega a p_1 , poi da p_1 vado in p_3 , e da p_3 vado in p_2 .

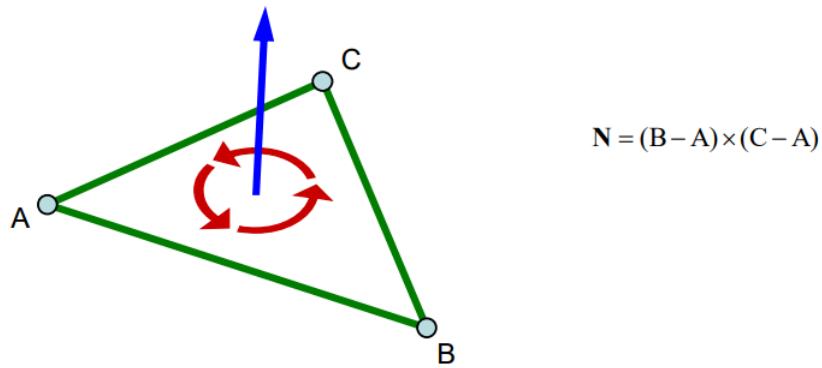
Quindi devo salvare una matrice di valori delle coordinate dei punti e una matrice delle facce come sequenze di punti che la definiscono, implicitamente dicendo che due punti di fila nella lista indicano che ci sia un edge che li collega.

Le mesh, essendo superfici, hanno delle limitazioni:

- i poligoni possono condividere al più un edge

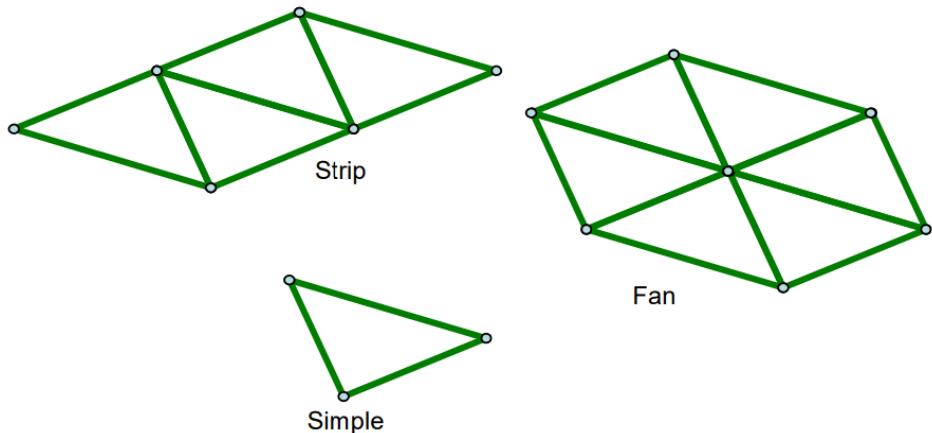


Le mesh triangolari ci possono dire qualcosa sulla **normale**. In particolare, l'ordine con cui percorro il triangolo

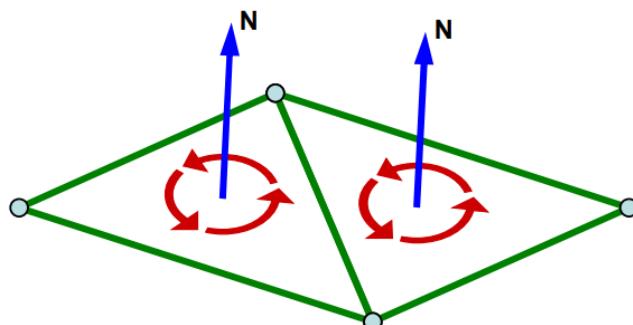


In generale si usa l'andamento **antiorario**.

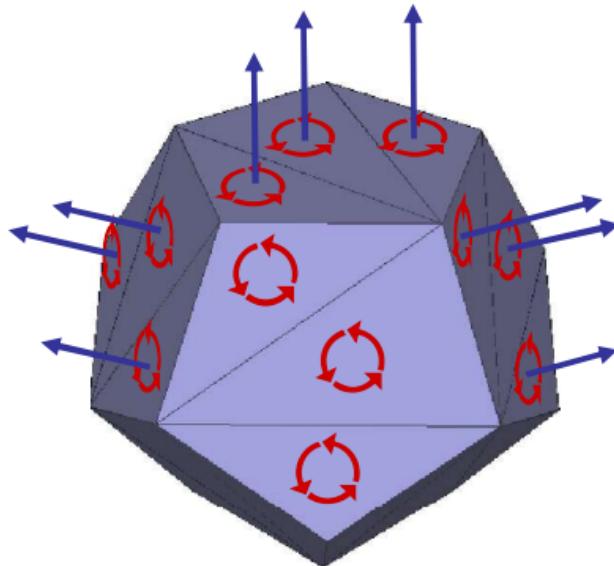
Le mesh triangolari e poligonali creano il problema di "quante ne attacco assieme?".



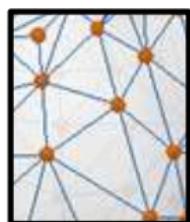
Due facce sono **compatibili** se le normali hanno la stessa orientazione, o meglio se esiste un esterno della superficie e un interno. Questi triangoli sono compatibili:



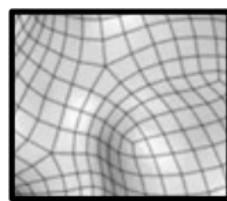
Un **oggetto chiuso**, senza bordo, è rappresentato da una mesh chiusa, che ha la proprietà che tutti i suoi poligoni hanno edge che sono condivisi esattamente da due facce. E tutte le normali puntano verso "fuori".



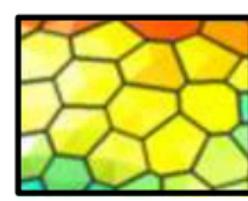
Non esistono solo mesh triangolari:



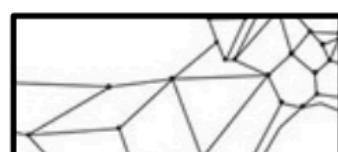
Triangular



?



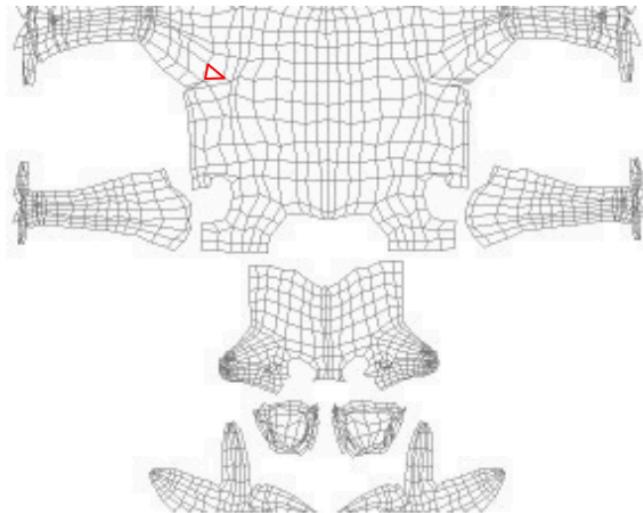
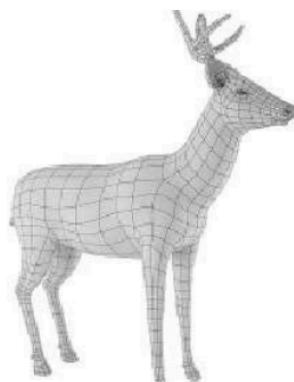
?



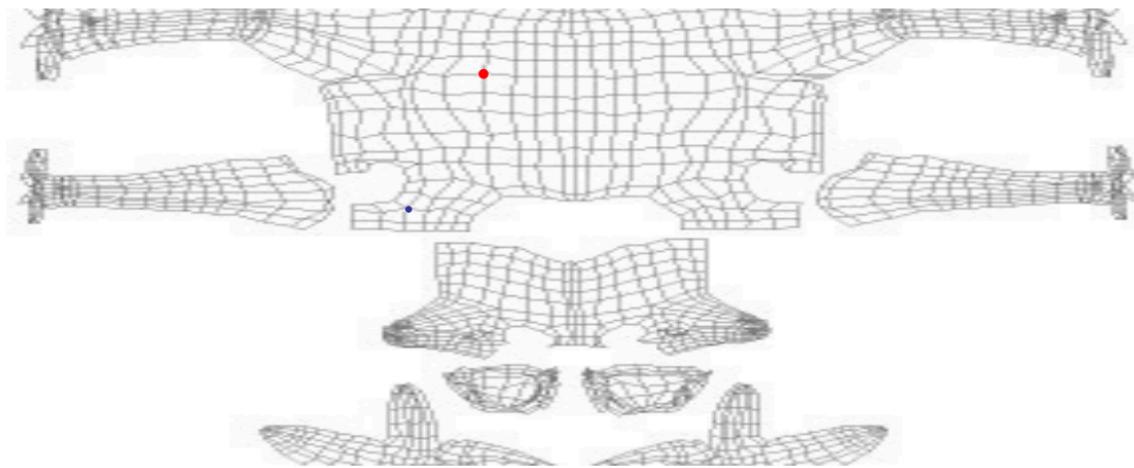
?

La quarta non è una mesh, perchè ogni edge deve collegare due vertici.

Di questa mesh esiste questo "spiatellamento", con cui posso fare una **texture**.



In una mesh posso anche usare un mix tra triangoli, quadrilateri...



In questo caso il punto rosso condivide 4 facce, quello nero invece 5, questa è sempre una variabile.

Le mesh sono tutte approssimazioni di una superficie continua, non è detto che rappresentando con mesh diverse io riesca ad avere l'uguaglianza di quello che rappresento.

Noi vogliamo una **rappresentazione** che ci permetta di rappresentare al meglio la struttura di questo dato, e ce la renda agile sia per capire la sua **topologia** (proprietà geometriche dell'oggetto) che per manipolare la sua **geometria**.

Vogliamo una rappresentazione che **minimizzi lo spreco di memoria** e che **supporti le operazioni** che vogliamo fare.

Ci sono molti formati, noi ci focalizziamo su .fbx e .obj che sono i più utilizzati.

Un file obj è fatto così:

◆ obj

```
1 #####
2 #
3 # OBJ File Generated by Meshlab
4 #
5 #####
6 # Object MooseObj_no_color.m
7 #
8 # Vertices: 7000
9 # Faces: 14000
10 #
11 #####
12 v -46.310894 0.950512 2.334025
13 v -46.283062 0.959372 1.005501
14 v -45.630074 0.259103 2.213509
15 v -46.620777 1.938330 1.901046
16 v -37.007906 -0.254051 -5.455070
17 v -39.753323 -6.043702 -5.617625
18 v -39.795357 -4.102954 -5.397791
19 v -40.907753 -3.103874 -3.842500
20 v -39.015361 -3.085747 -5.791756
21 v -40.564487 -2.284369 -4.559308
22 v -40.631439 -3.859112 -3.997926
23 v -40.281166 -5.515858 -2.739056
24 v -37.954430 -7.732079 -4.724075
25 v -36.821983 -8.597969 -3.481880
26 v -42.503063 -1.555553 -0.448015
27 v -41.008232 -4.094211 -2.823807
28 v -42.032499 -2.221395 -1.073404
29 v -43.131897 -0.672374 -0.160728
30 v -41.857307 -0.126238 -1.071174
31 v -42.370098 0.253158 -1.428216
```

A .obj file could contains

- Vertixes
- Faces
- Textures
- Normals

We will see that one could also desire to store other data such as:

- Animations
- Material
- Lighting
- ...

C'è un'intestazione che ci da delle proprietà dell'oggetto, come il numero di vertici e di facce. Poi parte con una serie di righe che hanno v e poi i valori delle coordinate.

Generalmente in questo oggetto metto la lista dei **vertici**, che hanno una posizione e ordine. Ad un certo punto, compare una lettera diversa, e inizia l'elenco dei 3 vertici che compongono ciascuna **faccia**, poi ci sono le **normali**.

Ci possono anche essere tipi diversi di dati obj.

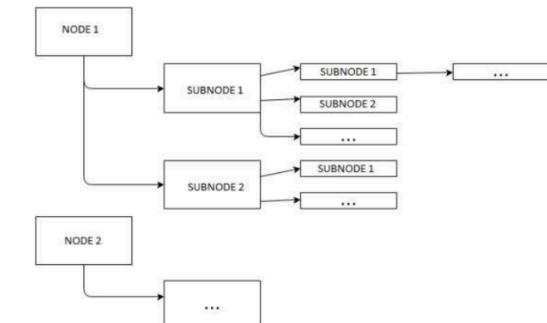
Posso anche inserire le **texture** nei file obj. Potremmo anche voler salvare **animazioni, materiali, luci...**

Tipicamente il formato che viene usato nei videogames quando vogliamo inserire anche queste informazioni, sono gli **fbx**. Contengono l'oggetto 3D, le informazioni sull'animazione, e altre proprietà. Sono usate anche per animazioni nei film e visual effects.

- ◆ An FBX file is a 3D file format that contains 3D object data and animation data. FBX files are commonly used in film, gaming, and VFX.

- ◆ What can FBX files contain?

- mesh,
- material,
- texture,
- skeletal animation data,
- scene information,
- lighting.



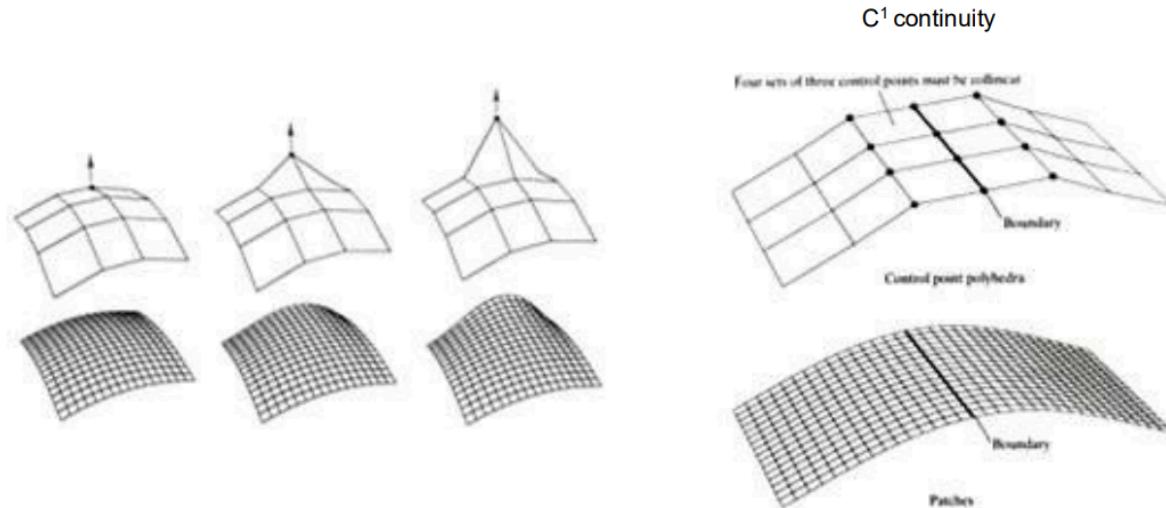
◆ Object node
Model subnode
Vertices subnode

Come è organizzato un file fbx? Hanno una **struttura a nodi**, per esempio questo object node, che ha come subnode il model subnode, che ha come sotto nodo il vertices subnode.

Per alcuni tipi di visualizzazioni, per esempio per permettere uno zoom in infinito su una superficie, può essere utile una rappresentazione continua per esempio quella delle **curve parametriche**.

Definiamo una serie di punti di controllo rispetto ai quali esiste una funzione che ci dice, rispetto ad un quadratino, come ci comportiamo all'interno del quadratino.

Questa è la **patch di Bezier** che è una patch con 16 punti di controllo per rappresentare una patch.

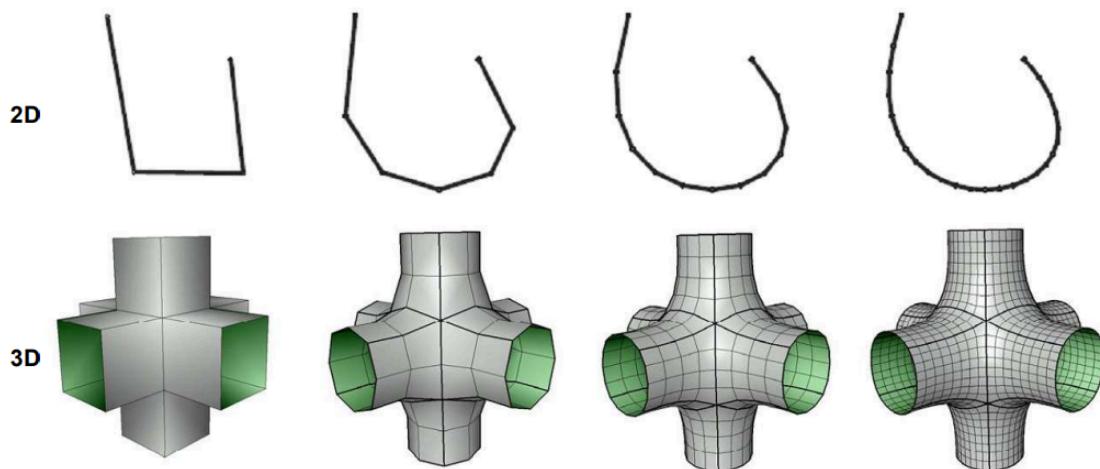


Spostando un punto di controllo, cambiamo le variabili di una funzione che dice come è fatta la patch in quella zona.

Nell'esempio sopra se alzo quel punto di controllo, la superficie intorno si muove in modo continuo di conseguenza.

Dobbiamo sapere che esistono.

Per rappresentare la geometria con una discretizzazione possiamo fare una **suddivisione**: partiamo da una mesh con pochi vertici e pochi edge. Definiamo un'algoritmo che in maniera iterativa va ad aggiungere nuovi vertici e nuovi edge. Di solito si ottiene uno **smoothing**.

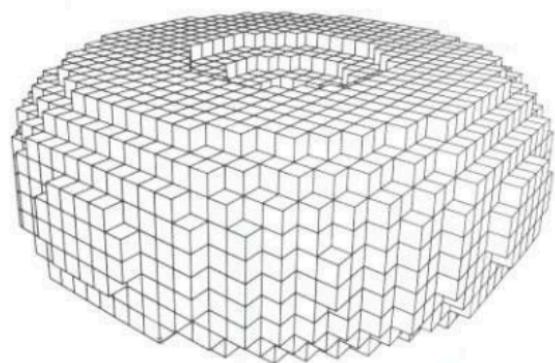


Esistono le rappresentazioni **volumetriche**, usate in alcuni videogames.

L'idea è che sono pixel volumetrici, lo spazio 3D è diviso in celle regolari e a ciascuna cella possiamo dare alcune proprietà

◆ Voxel (Volumetric Pixel)

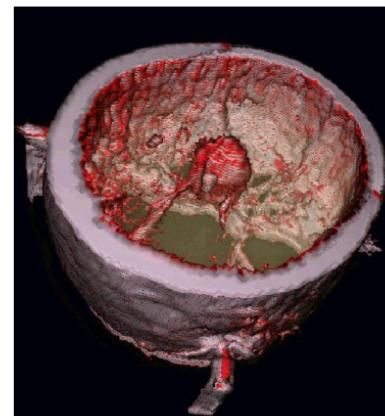
- The 3D space is divided into a regular grid
- Each cell represents a base element of information
 - Color
 - Position
 - Density
 - Temperature
 - ...



Vengono usati anche nella medicina:

◆ Voxel (Volumetric Pixel)

- Many acquisition devices produce volumetric data
 - Computerized Axial Tomography
 - Magnetic Resonance



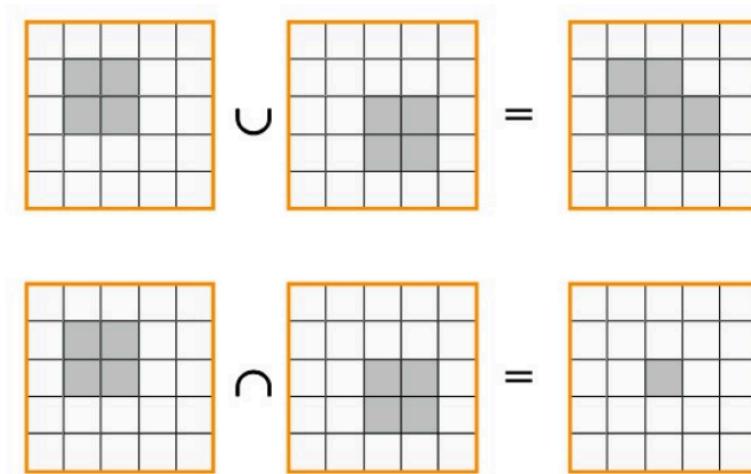
I voxel hanno problemi di bassa scalabilità, se vogliamo rappresentare un oggetto 3D dobbiamo prendere un cubo e dividerlo in cubi, quindi se vogliamo prendere una risoluzione su ogni asse pari a 1000, abbiamo subito un'esplosione di dimensioni.

- ◆ With n cells per dimension, we need to store $O(n^3)$ data!!
 - Eg 1000 cells = 1 billion Voxels

Un'ottimizzazione che si può fare è verificare quali di questi cubi mi sono utili, escludendo alcuni e facendo densità diverse in zone diverse. (Oct-tree)

Questo però non risolve tutti i problemi.

Di pro, i voxel, nonostante siano discreti, hanno le operazioni come intersezione e unione:



I voxel hanno anche bisogno di strumenti aggiuntivi per fare rendering.

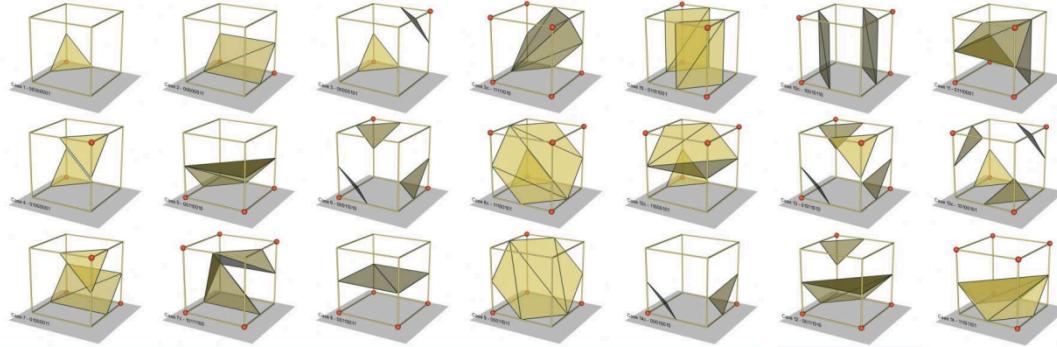
Possiamo vederli facendo ray casting.

Il vantaggio è che ci possono dare informazioni anche su come è fatto all'**interno** un oggetto.

I **marching cubes** sono una via di mezzo tra i voxel e le mesh, dove i vertici del cubo diventano più importanti del cubo. C'è un processo che ci fa capire la superficie all'interno di ciascun cubo, e dopo aver individuato quali tra le possibilità è la superficie che passa per quel cubo, gli viene assegnata una mesh triangolare che corrisponde alla stima della superficie:

◆ Marching cubes

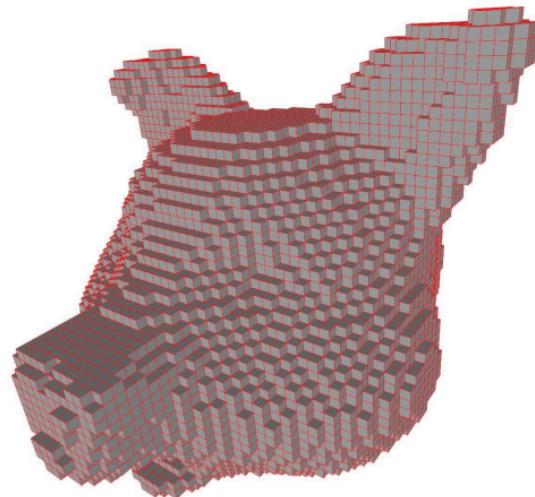
- Different configurations
- +Strategies to cope with ambiguous cases



Il rendering sui voxel può essere fatto interessandoci solamente alle facce.

◆ Rendering via Cuberille

- Voxels are cubic pixels (modelled with meshes)
- Classic rendering techniques



Per esempio minecraft era (forse ancora è) fatto così.

◆ Voxels representation

- Advantage

- Simple, intuitive, non ambiguous
- All the objects have the same representation complexity
- Some devices naturally generate volumetric voxel data
- Objects can be easily manipulated with boolean operations

- Disadvantage

- Voxels are an approximation
- Carefully designed data structure
- Large storage space

Al posto dei cubi possono essere usati anche tetraedri (non ci sono grossi risultati al momento):

