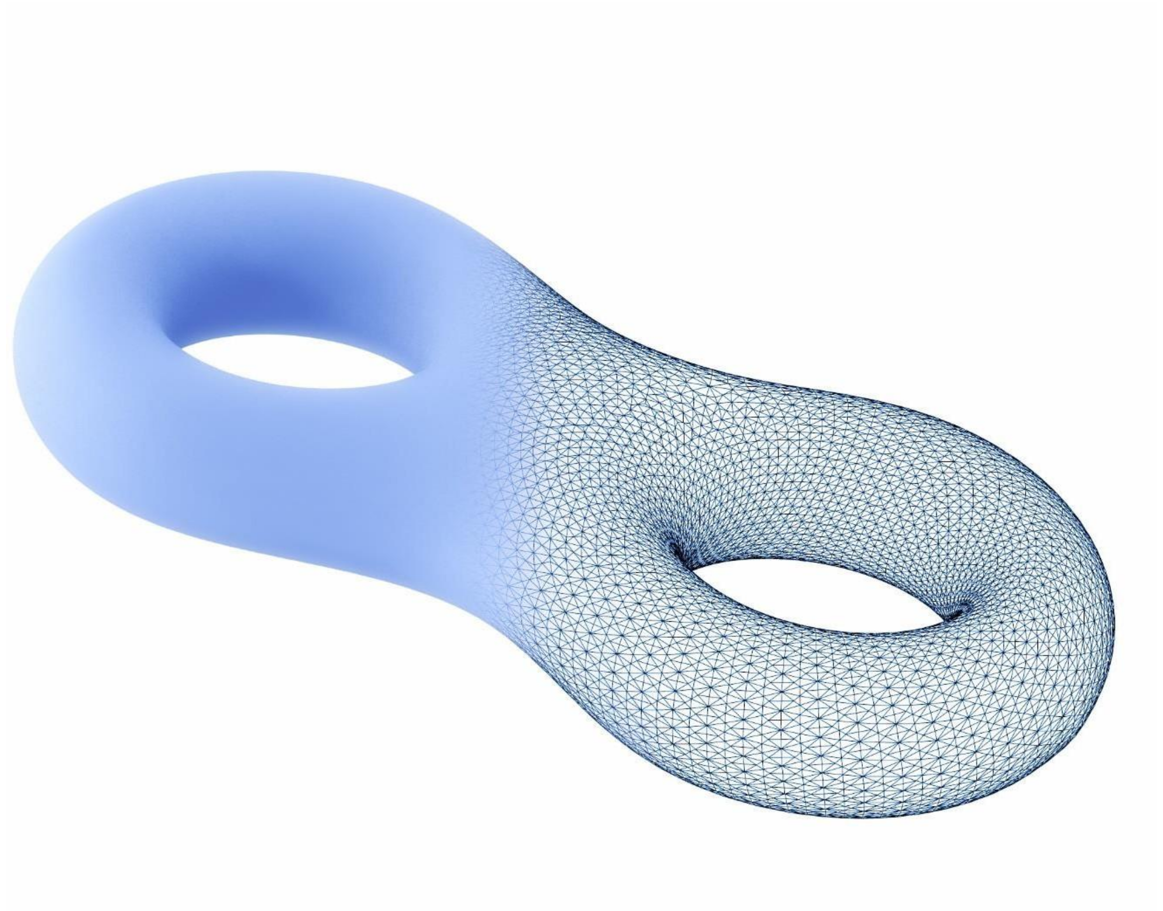
Mesh Generative Models



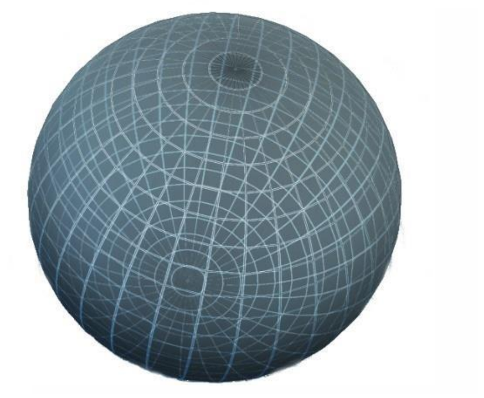
Mustafa Hajij

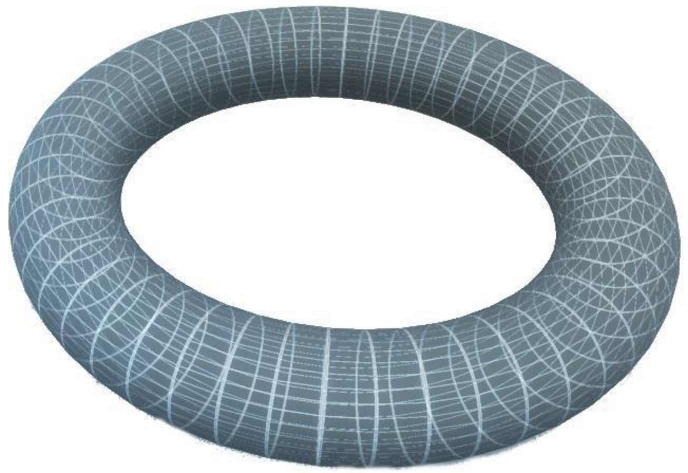
Outline

- The concept of a surface
- Surfaces with boundary
- Topological properties of surfaces
- Triangulated surfaces
- Mesh Generative Models

The concept of a surface

- Consider the following shapes :

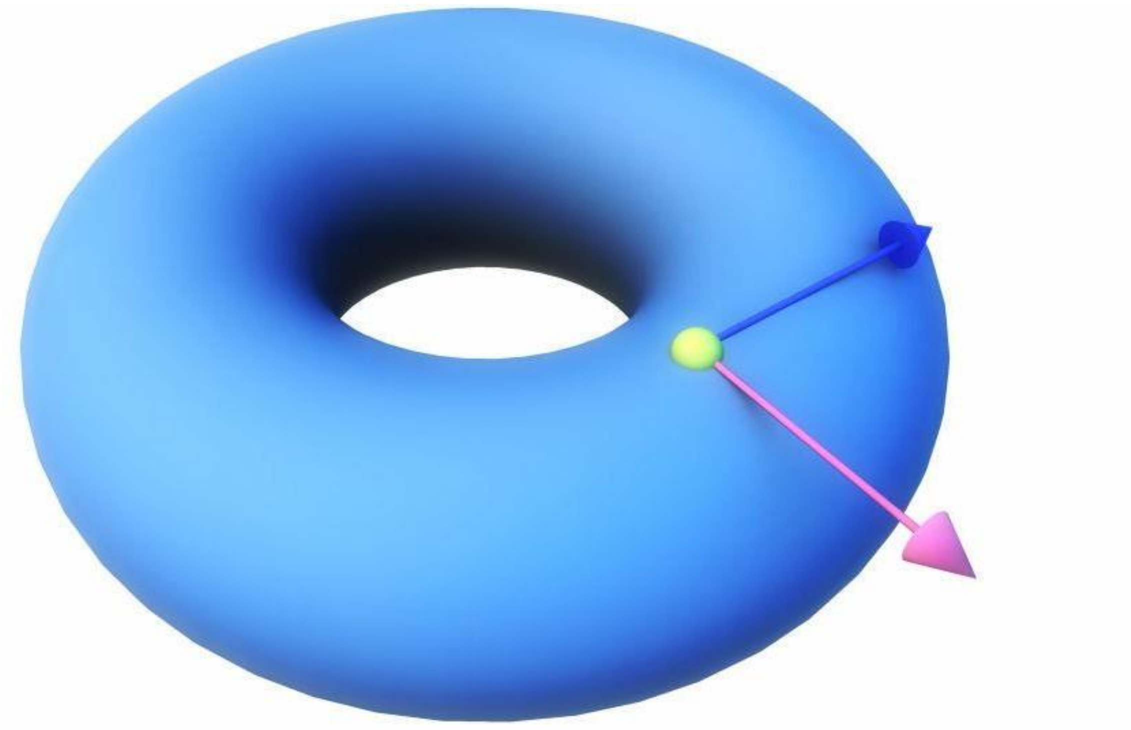




- In all examples that we will consider we will only consider the "surface" of the shape and not what is inside.

The concept of a surface

Surfaces are two dimensional objects, in the sense :



For any point on the surface of the torus, the yellow sphere has only two degrees of freedom in its movement.

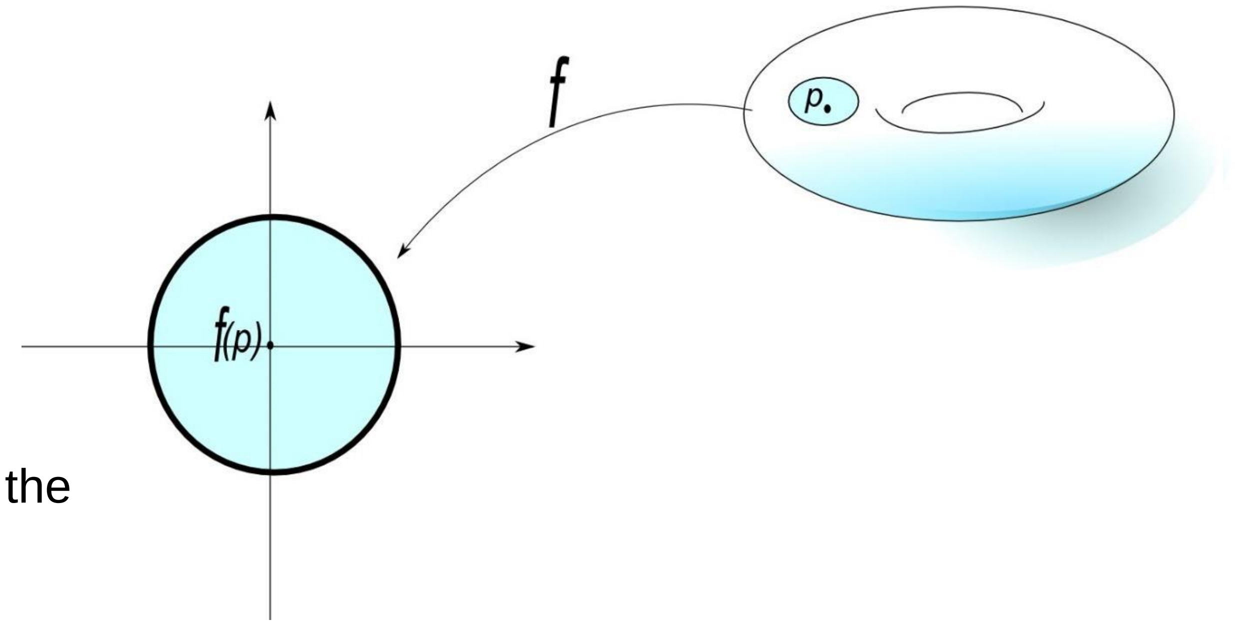
The concept of a surface

Mathematically, a surface is a space that looks locally like a disk.

In other words, for every point *p* on the surface

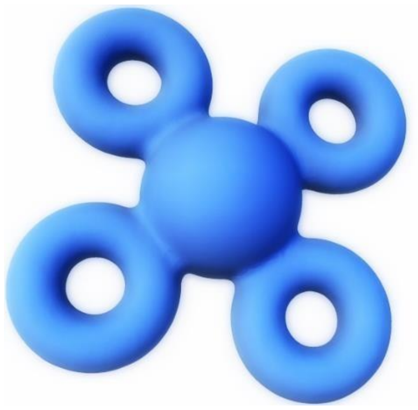
there exists a small disk

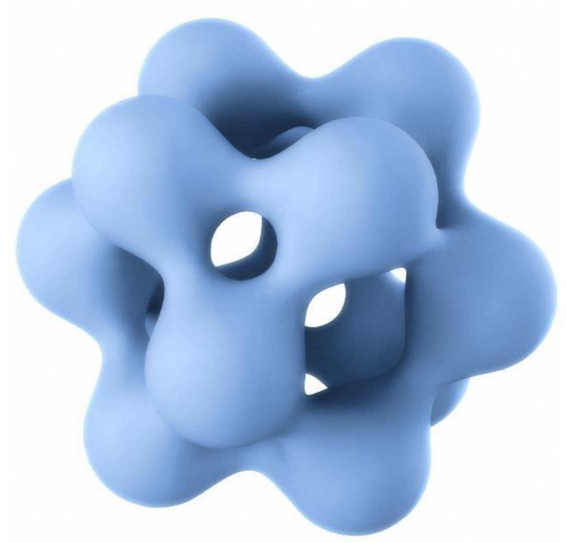
that is mapped via a map *f* to the unit disk in the plane.

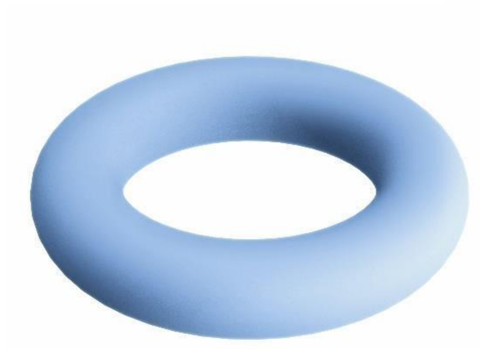


The concept of a surface

So, the following shapes can be considered as surfaces :

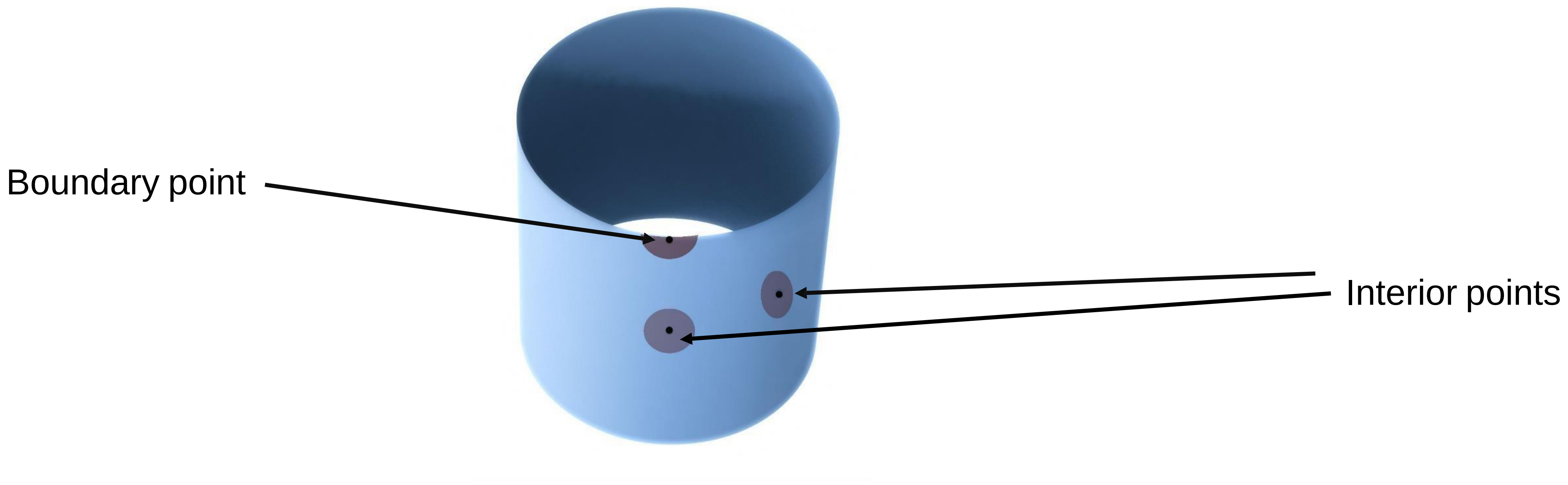






Surfaces with boundary

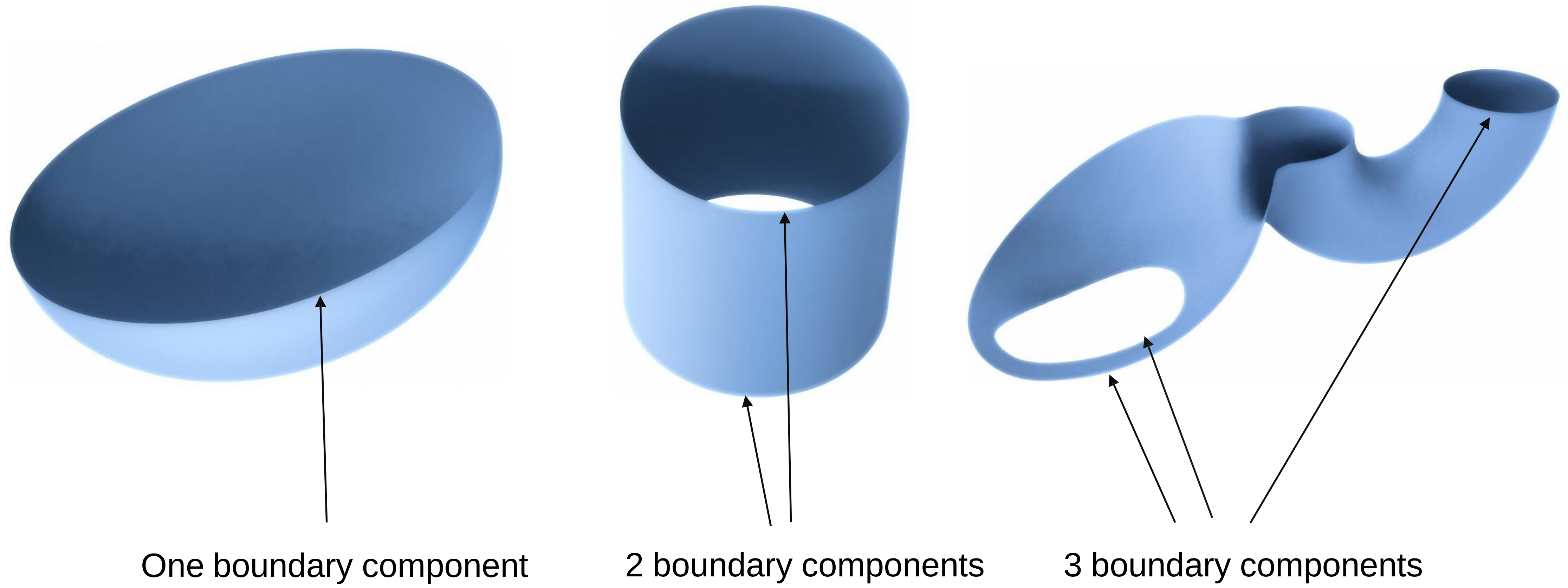
In this context, how do we consider a surface like the one here?



This shape matches our intuition of what a surface is but it does not quit fit with the previous definition since it has “edges”.

Surfaces with boundary

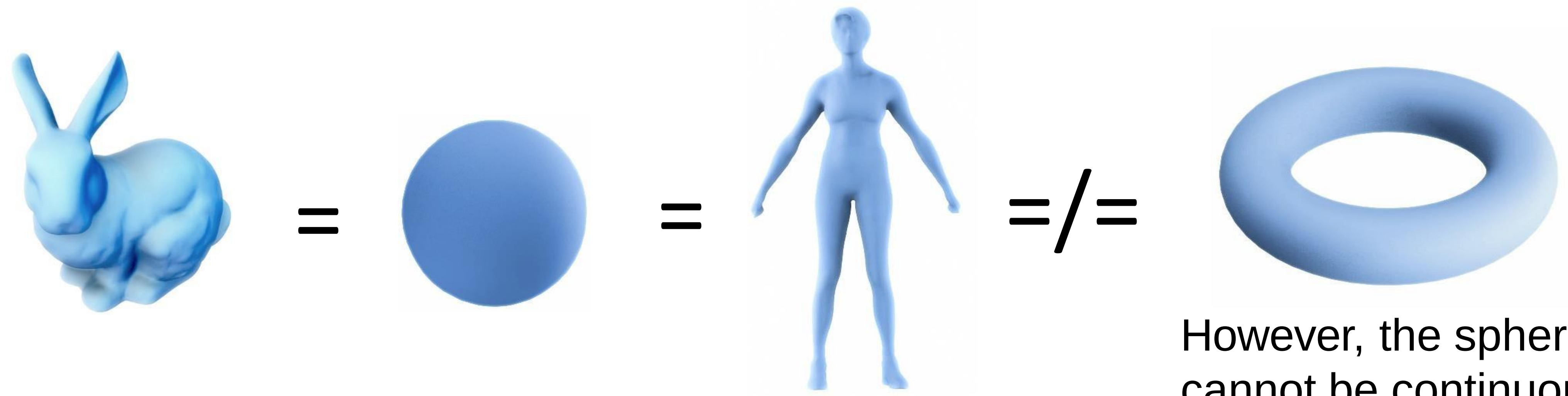
A surface can have more than one boundary components



Topology of surfaces

Roughly speaking, topology of an object studies the way this object is connected.

Topology studies the properties of shapes that do not change under continuous deformations.

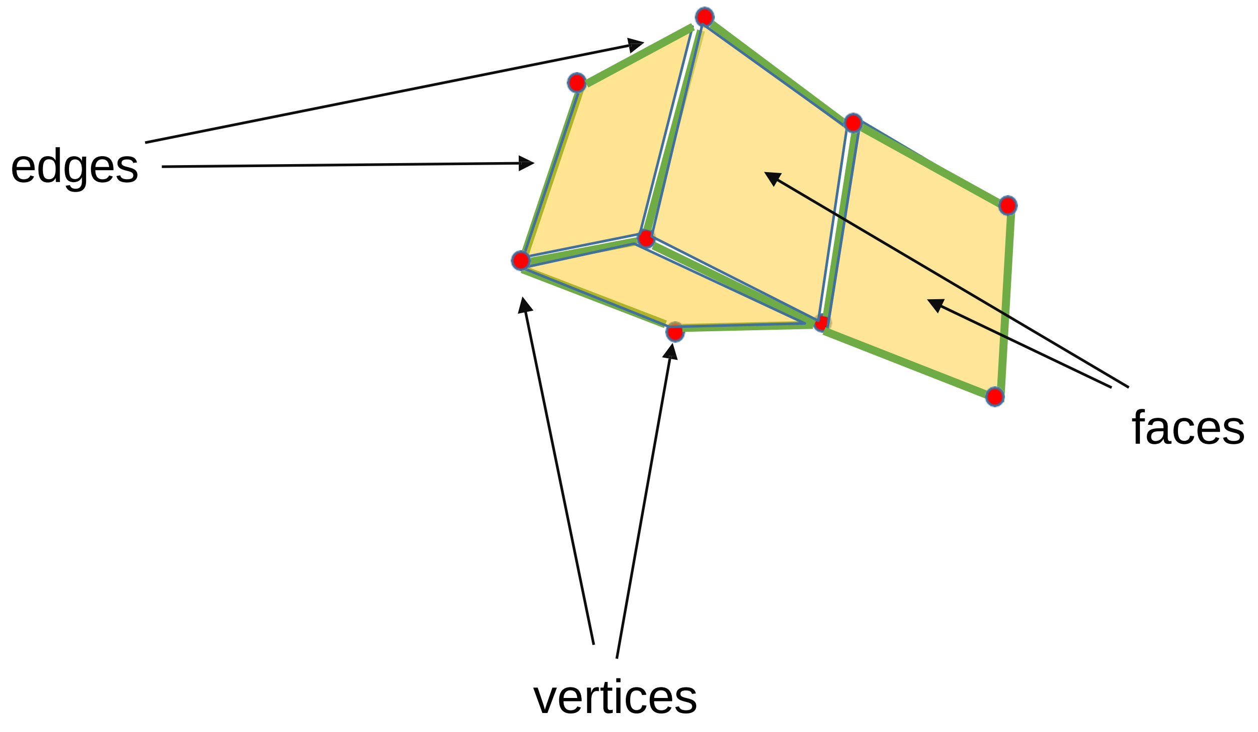


So topologically, the following objects are equivalent because we can deform each one of them contentiously without tearing into the other.

However, the sphere cannot be continuously deformed into the torus. Hence the sphere and the torus are topologically district.

Polygonal Mesh

Roughly speaking, a polygonal mesh is a set of vertices **V**, a set of edges **E** and a set of faces **F** that are coherently glued together.

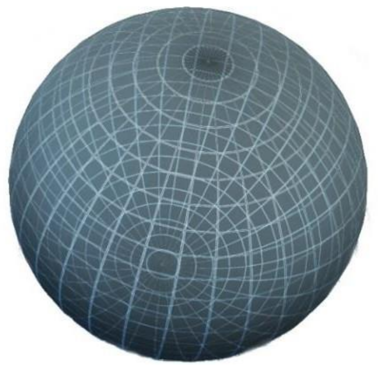


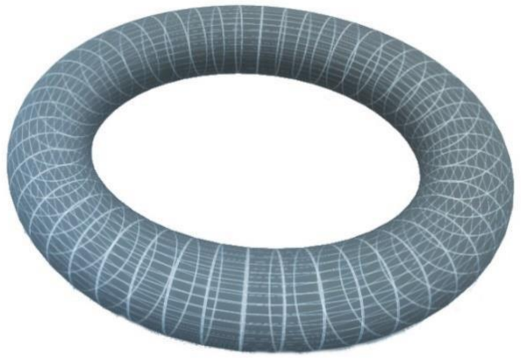
Polygonal Mesh

The information encoded in a mesh can be either topological information or geometric information.

Roughly speaking, the topology of the mesh is the way various elements of the mesh is glued and connected together.

For instance, the topology of the sphere is different from the topology of the torus

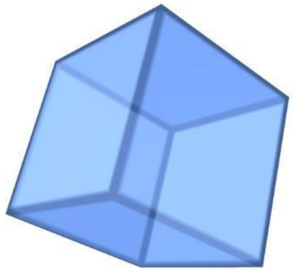




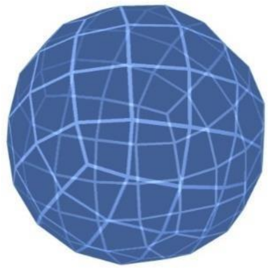
Geometric information of the mesh is given by specifying coordinates in the *3* dimensions to each vertex in the mesh.

Topological properties of surfaces

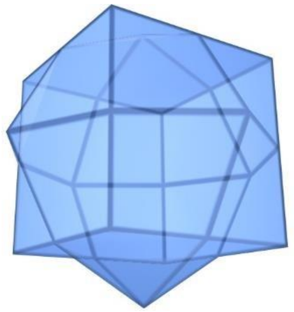
To understand what we mean by a topological we consider the following example.



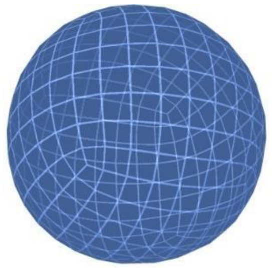
V-E+F=8-12+6=2



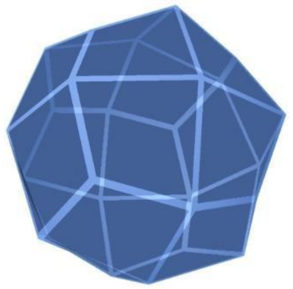
V-E+F=2



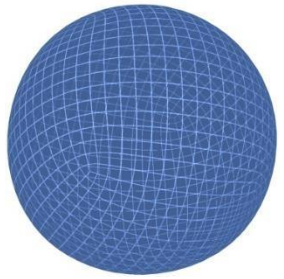
V-E+F=26-48+24=2



V-E+F=2



V-E+F=2

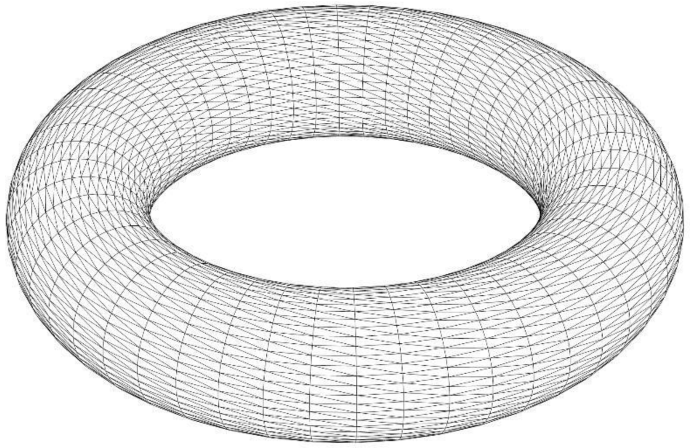


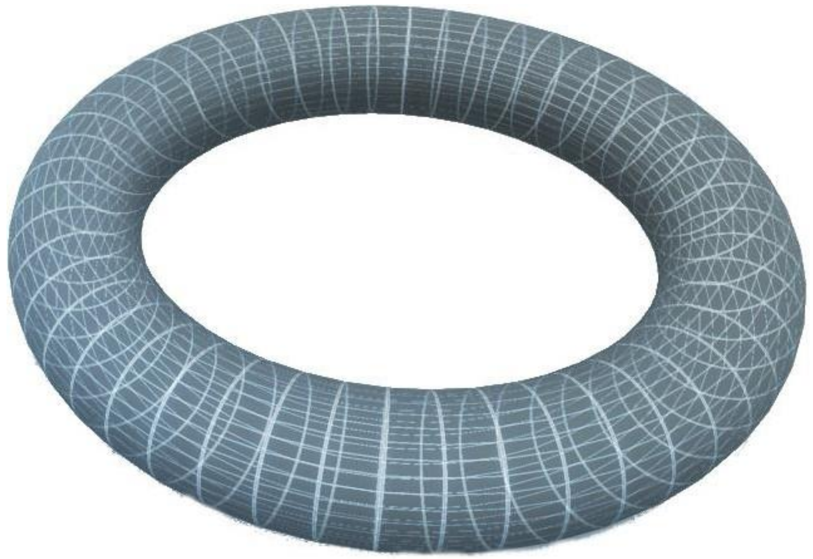
V-E+F=2

Topological properties of surfaces

- So for all these spherical shapes the following quantity V-E+F=2. This is true regardless of the number of *faces, edges and vertices* we choose to make the spherical shape.

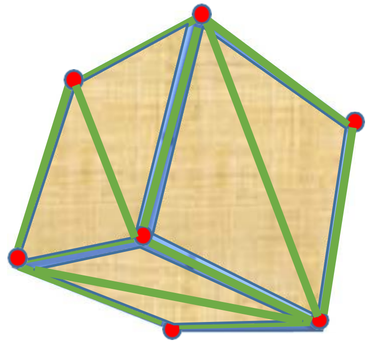
- On the other hand if we try to compute the same quantity for the torus we will get V-E+F=0. Regardless of the number of faces, edges and vertices we choose to build up the torus from.

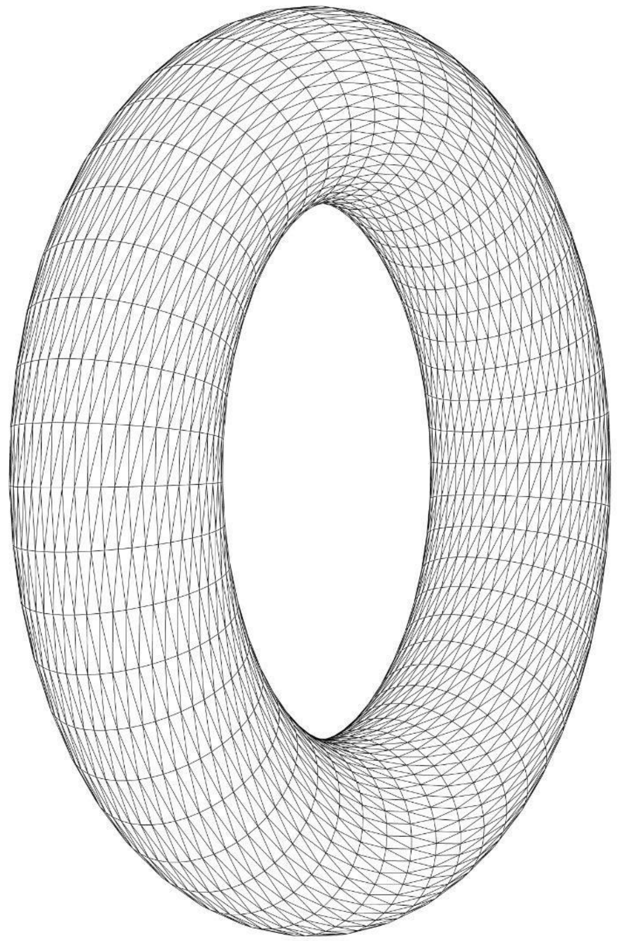




Triangulated Mesh

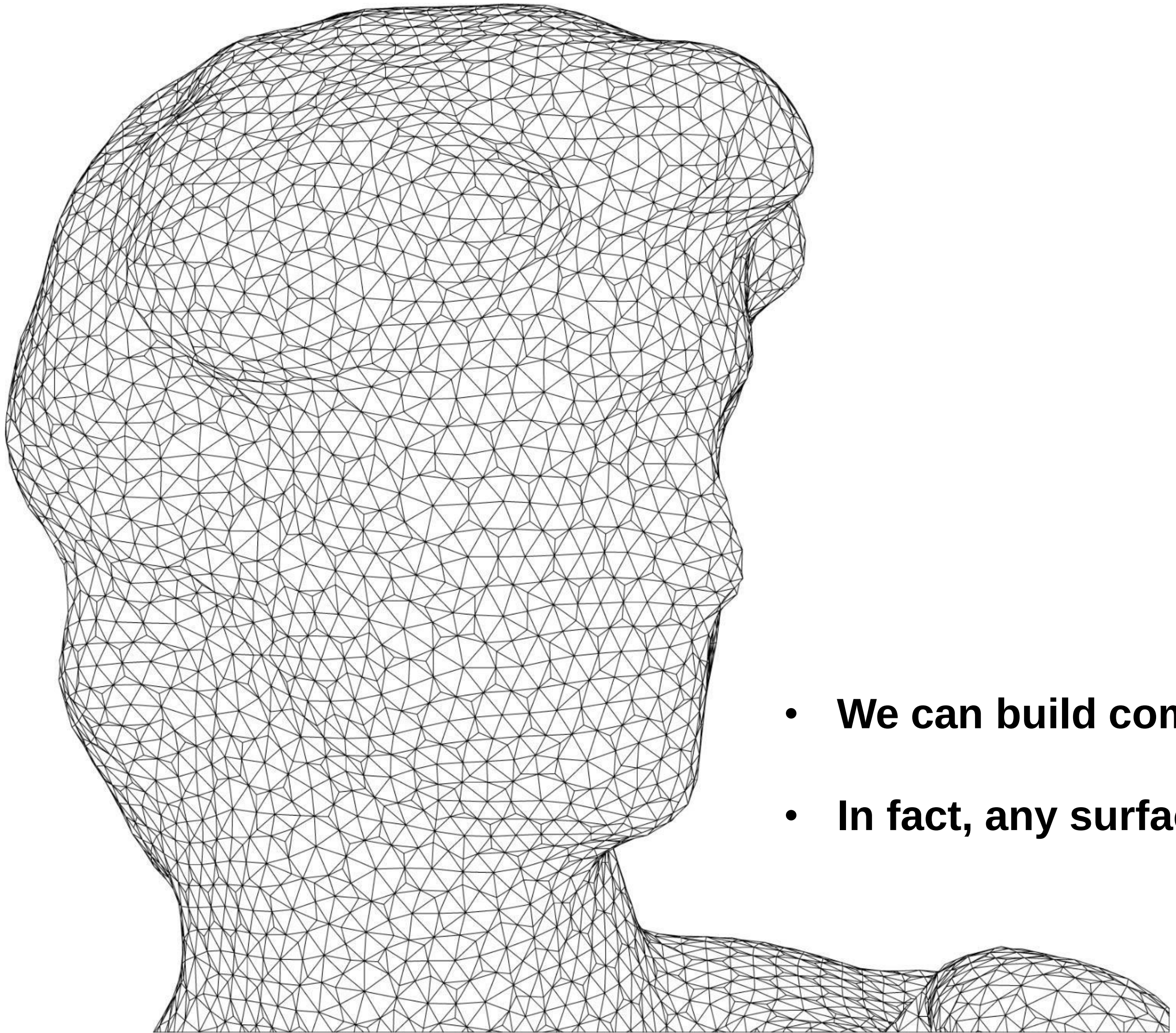
A mesh whose all its faces are triangles is called a triangulated mesh.

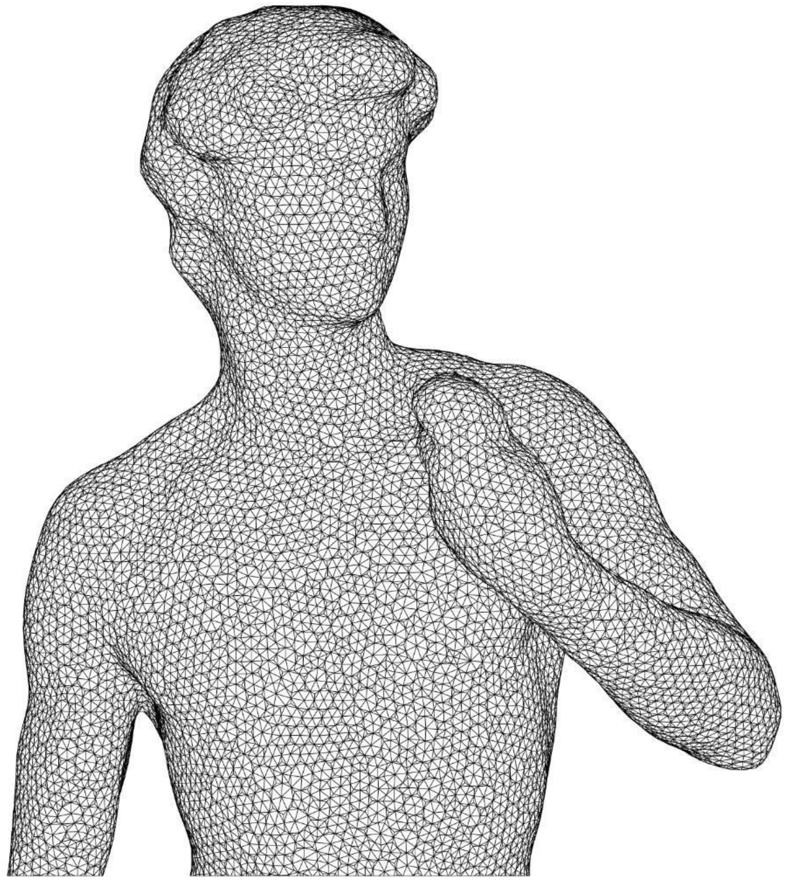




An example of a triangulated torus

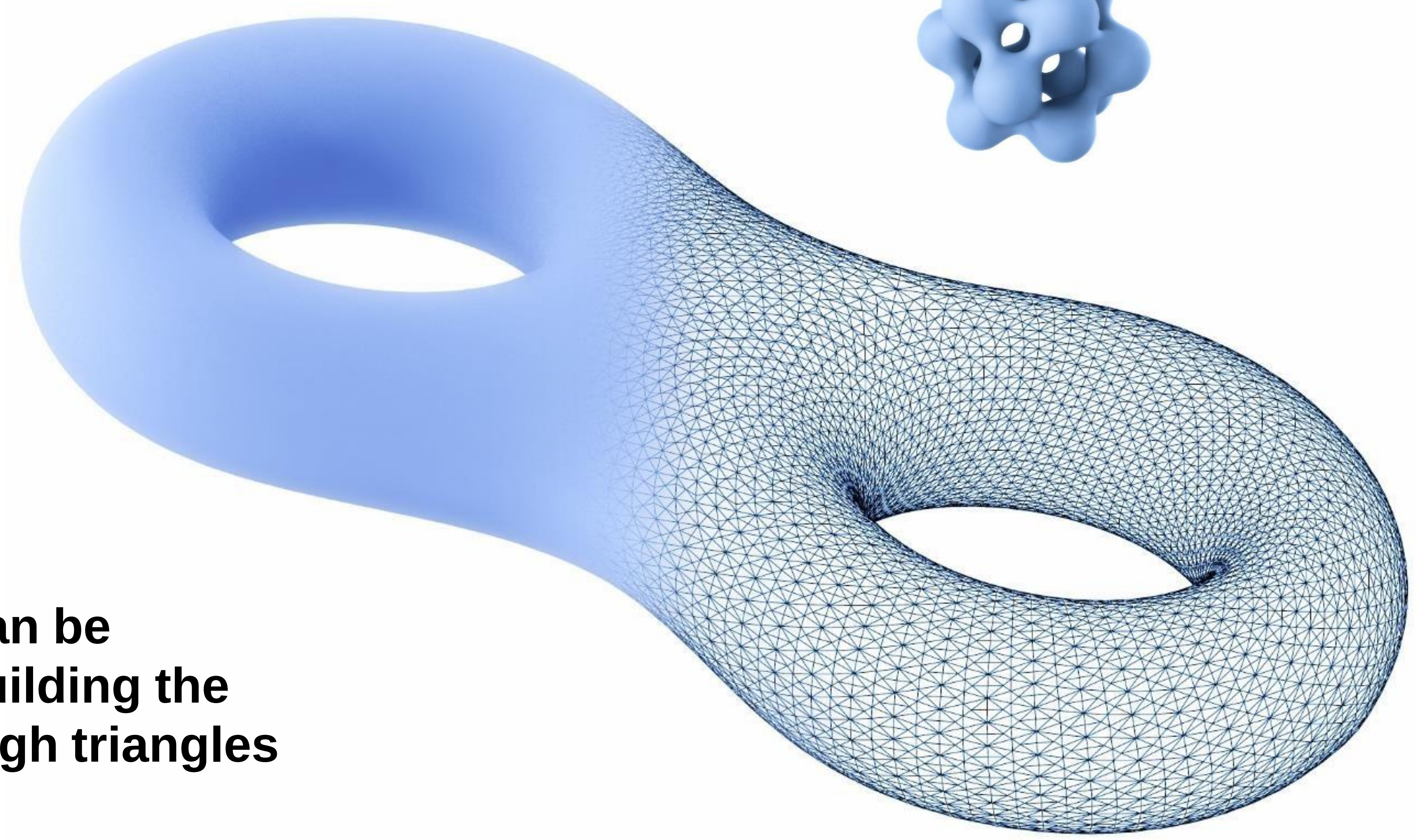
Triangulated Mesh





- **We can build complicated surfaces by gluing triangles.**
- **In fact, any surface can be built from triangles.**

Triangulated Mesh



- **Smooth surfaces can be approximated by building the surfaces from enough triangles**

Triangulated Mesh

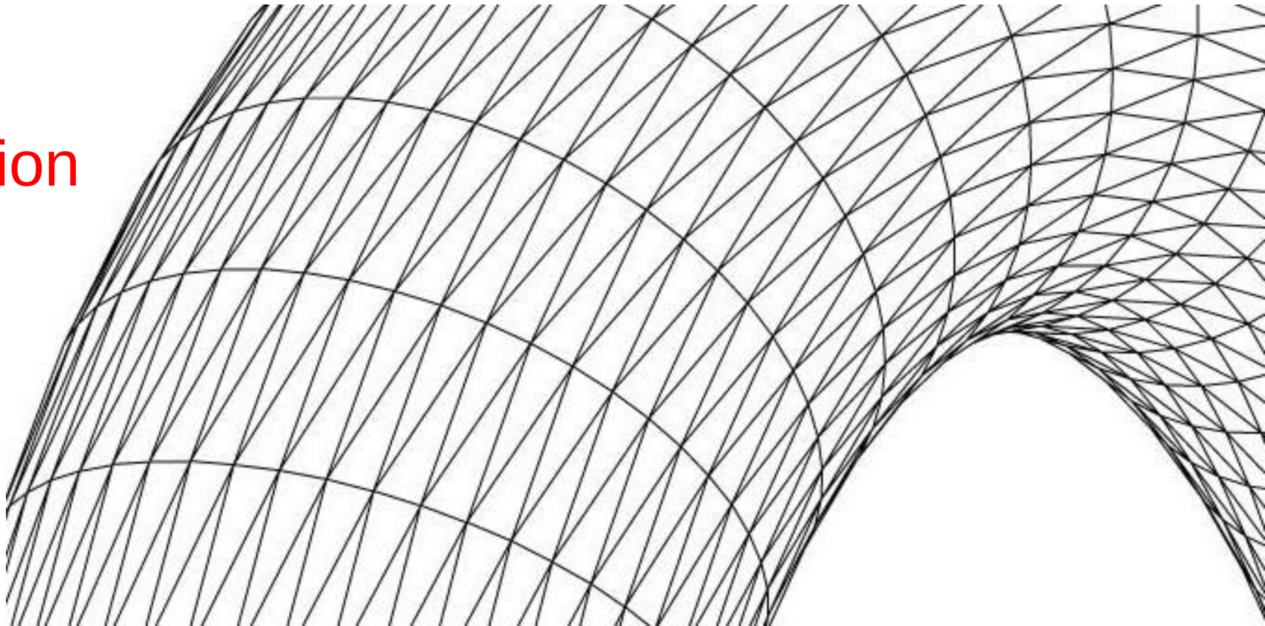
- Formally, a triangulated mesh is specified by the following data

Set of vertices $V = \{v_1, ..., v_n\}$

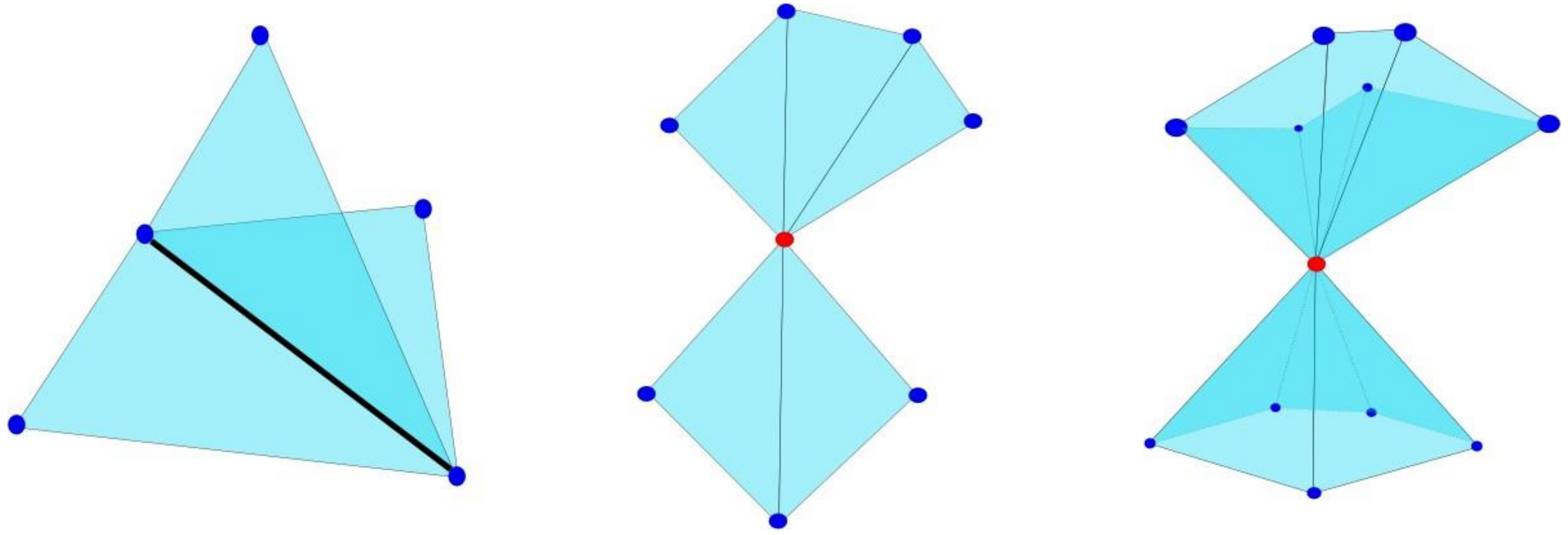
Set of edges $E = \{e_1, ..., e_k\}, \;\; E \sqsubseteq V \times V$

Set of faces $F = \{f_1, ..., f_k\}, \;\; F \sqsubseteq V \times V \times V$

- Moreover, one usually specifies a position ***p_i* in *R^3*** for every vertex ***v_i*** .



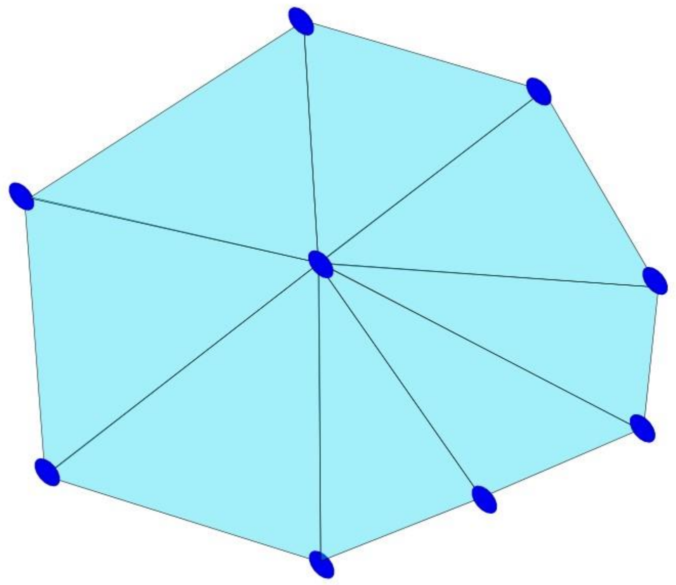
Triangulated 2-Manifold Mesh

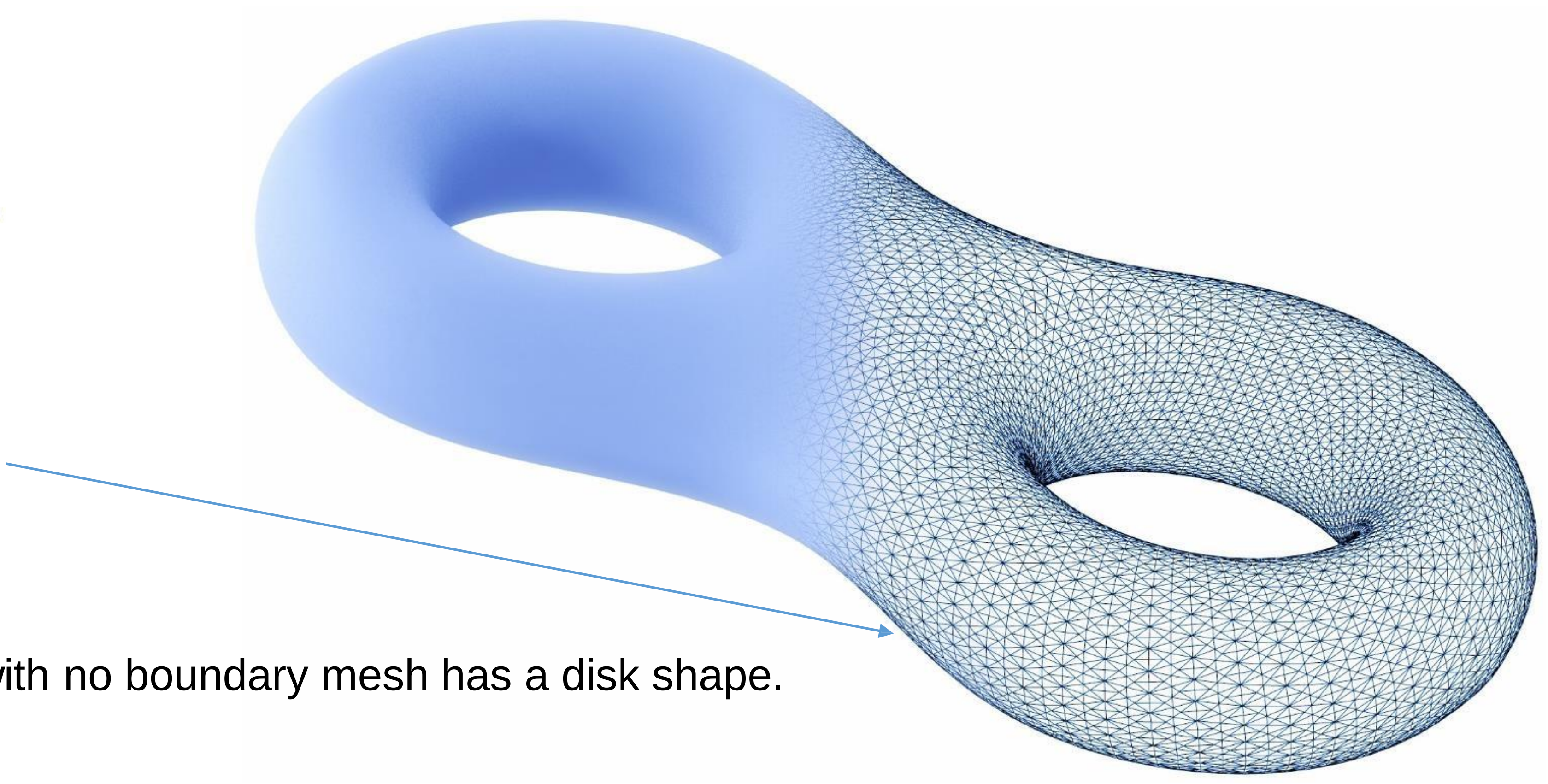


- These are examples of non-manifold meshes. On the left, non-manifold edge, and on the right we have non-manifold vertex

Triangulated 2-Manifold Mesh

- A *2*-manifold triangulated mesh, is a triangulated mesh that does not contain non-manifold edges nor non-manifold vertices nor self-intersections.



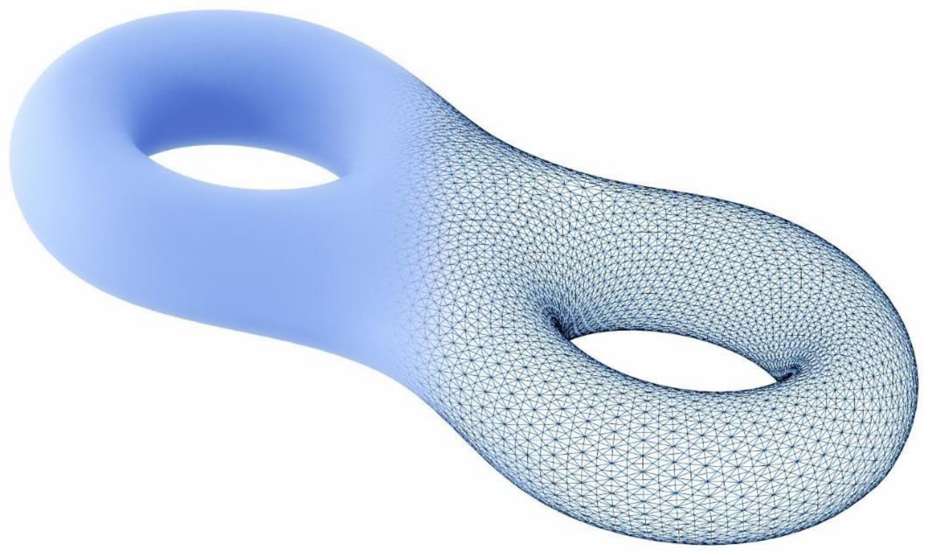


Locally, a triangulated *2*-manifold with no boundary mesh has a disk shape.

Mesh Data Structure

- What information do we need to store about a polygonal mesh?

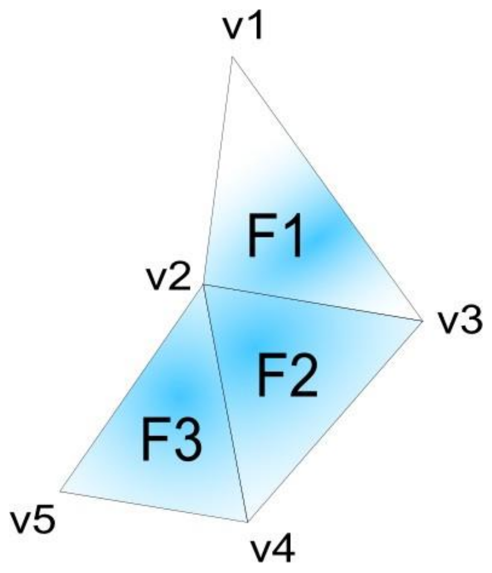
1. Topology : Adjacency information, For example, given a vertex v what are the vertices that v?
2. Geometry : 3d coordinates for each vertex
3. Attributes : Face or vertex normal, texture coordinates, etc.



Mesh Data Structure

Face-set data structure :

- In this data structure we list the set of polygons in the mesh and for each polygon and we represent each polygon by the coordinates of its vertices.

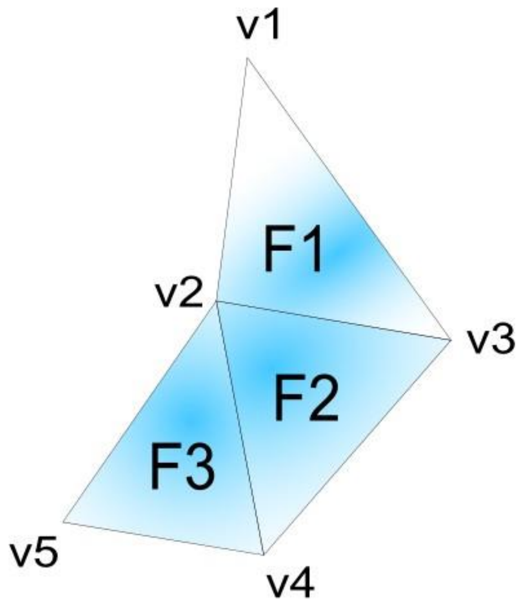


Faces		
x11 y11 z11	x12 y12 z12	x13 y13 z13
x21 y21 z21	x22 y22 z22	x23 y23 z23
x31 y31 z31	x32 y32 z32	x33 y33 z33

- Problems with this data structure : topology information are not directly stored and vertices position are stored more than once. For instance, position of v2 is stored 3 times in the previous table.

Mesh Data Structure

Face-set data structure :

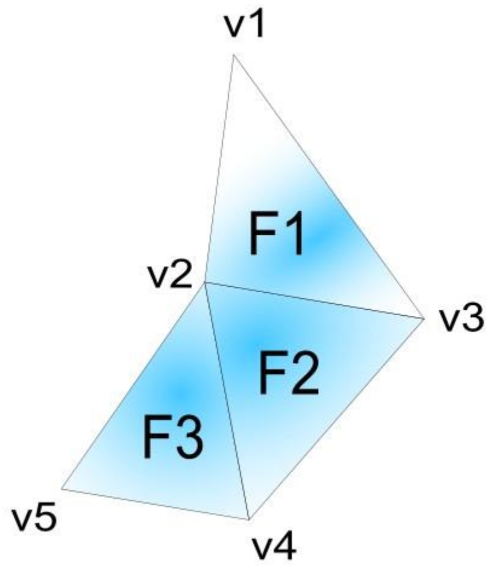


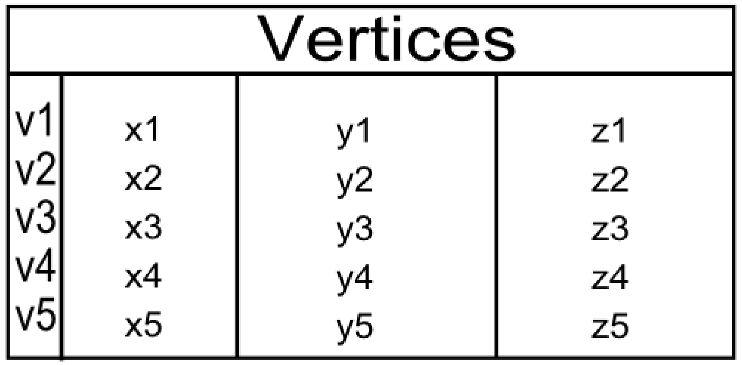
Faces		
x11 y11 z11	x12 y12 z12	x13 y13 z13
x21 y21 z21	x22 y22 z22	x23 y23 z23
x31 y31 z31	x32 y32 z32	x33 y33 z33

- If we represent each coordinate position with a 32-bit single precision number (4 bytes), then each triangle face needs 4 times 3 times 3 =36 bytes.

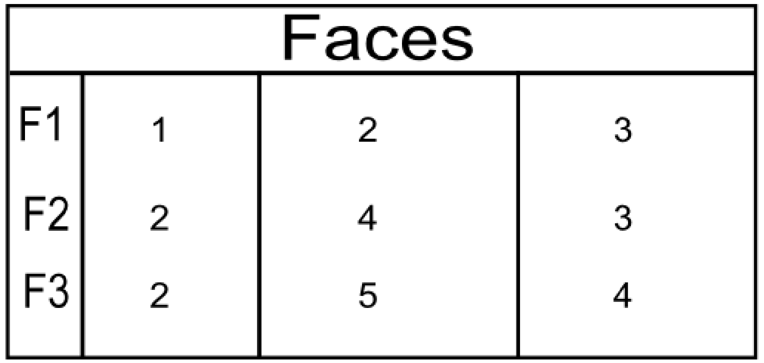
Mesh Data Structure

Indexed face-set data structure :





Vertices			
v1	x1	y1	z1
v2	x2	y2	z2
v3	x3	y3	z3
v4	x4	y4	z4
v5	x5	y5	z5



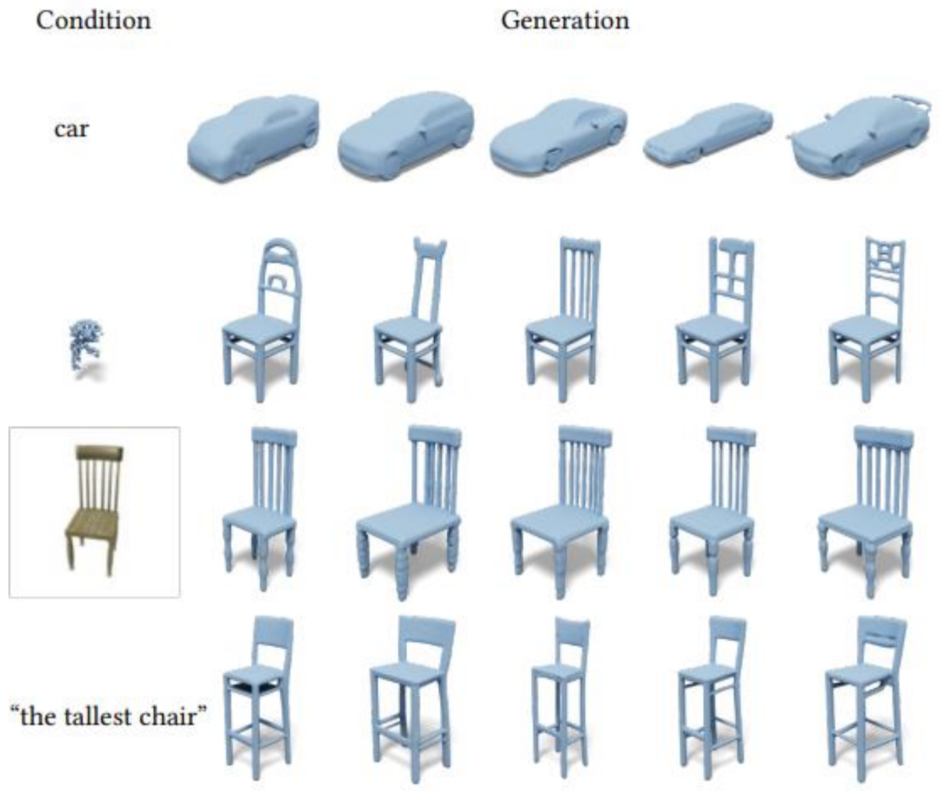
Faces			
F1	1	2	3
F2	2	4	3
F3	2	5	4

- In this data structure we list the set of vertices positions, and we list the set of indices of the faces.
- The file format of OFF, OBJ uses this indexed face-set data structure.

Mesh Generative Models

Task : we like to generate mesh similar way we did for images : find generative models that can generate novel meshes, condition these models on text, image etc.

Applications :computer graphics, gaming, virtual reality, etc.



Ref : https://arxiv.org/pdf/2301.11445

Recall self-attention

An Attention Layer:

- In an attention layer (Vaswani et al., 2017), there are three types of inputs: queries, keys, and values.
- Queries $Q = [q_1, q_2, \ldots, q_{N_q}] \in \mathbb{R}^{d \times N_q}$ and keys $K = [k_1, k_2, \ldots, k_{N_k}] \in \mathbb{R}^{d \times N_k}$ are initially compared to produce coefficients.

The attention coefficients are defined via:

$$A_{i,j} = \frac{\mathbf{q}_j^\top \mathbf{k}_i / \sqrt{d}}{\sum_{i=1}^{N_k} \exp\left(\mathbf{q}_j^\top \mathbf{k}_i / \sqrt{d}.\right)}$$

- Coefficients $A_{i,j}$ are obtained by normalizing the dot products of queries and keys with the softmax function.
- These coefficients are then used to linearly combine values $V = [v_1, v_2, \ldots, v_{N_v}] \in \mathbb{R}^{d_v \times N_v}$.

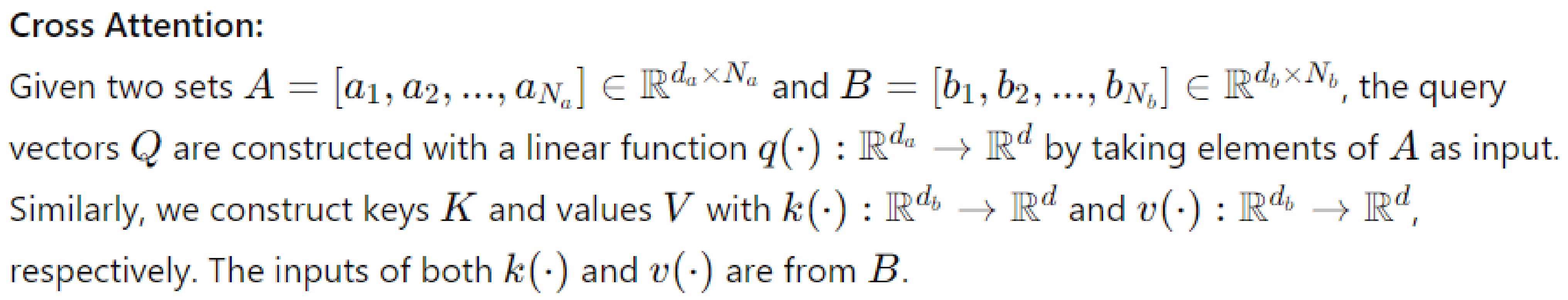
Recall self-attention

The output of the attention layer is given by :

$$
\begin{aligned}
&\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \\
&= \begin{bmatrix} \mathbf{o}_1 & \mathbf{o}_2 & \cdots & \mathbf{o}_{N_q} \end{bmatrix} \in \mathbb{R}^{d_v \times N_q} \\
&= \begin{bmatrix} \sum_{i=1}^{N_k} A_{i,1} \mathbf{v}_i & \sum_{i=1}^{N_k} A_{i,2} \mathbf{v}_i & \cdots & \sum_{i=1}^{N_k} A_{i,N_q} \mathbf{v}_i \end{bmatrix}
\end{aligned}
$$

Observe that the input has length Nq and the output has the same length.

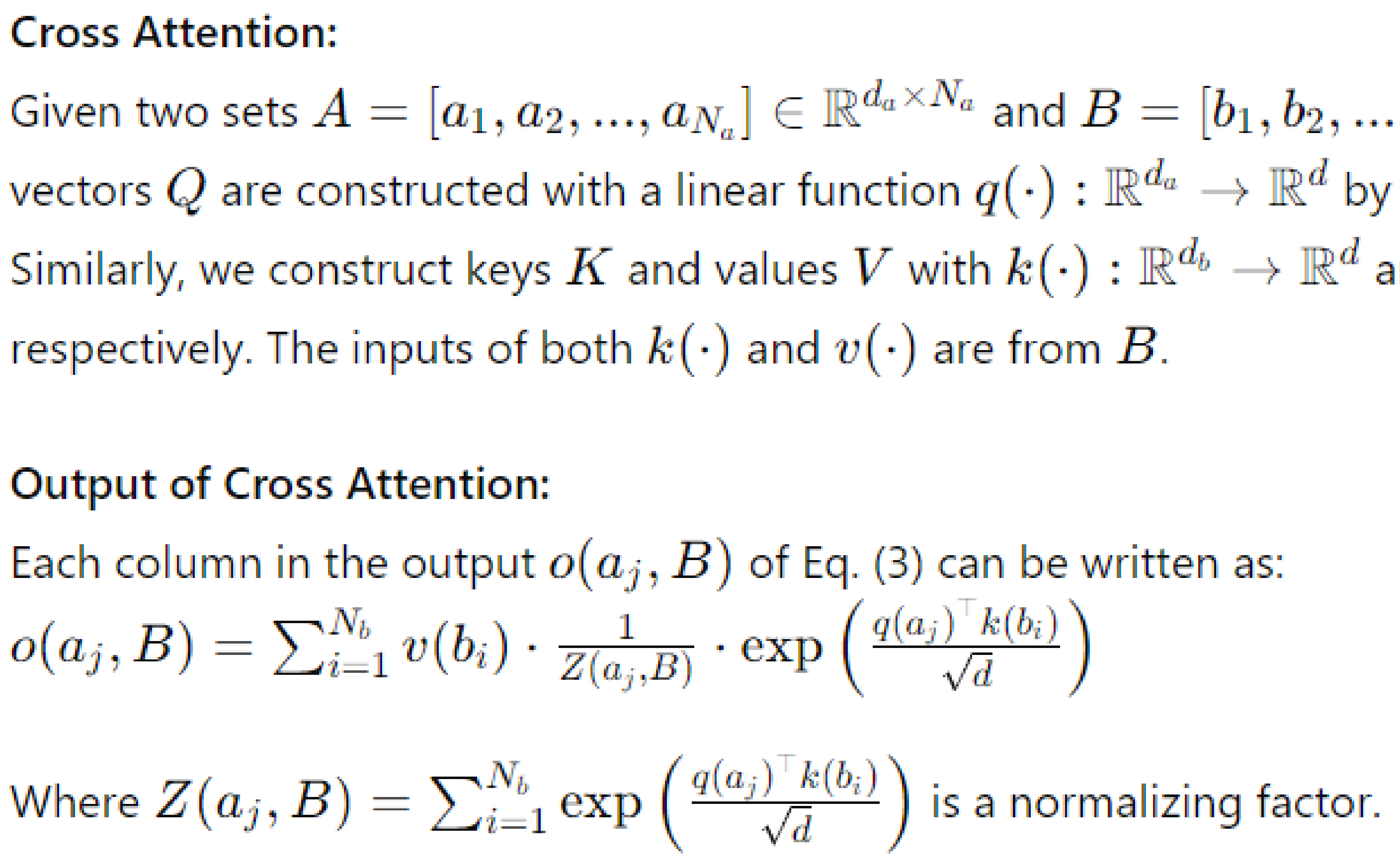
Recall cross-attention



Cross Attention:

Given two sets $A = [a_1, a_2, ..., a_{N_a}] \in \mathbb{R}^{d_a \times N_a}$ and $B = [b_1, b_2, ..., b_{N_b}] \in \mathbb{R}^{d_b \times N_b}$, the query vectors Q are constructed with a linear function $q(\cdot) : \mathbb{R}^{d_a} \rightarrow \mathbb{R}^d$ by taking elements of A as input. Similarly, we construct keys K and values V with $k(\cdot) : \mathbb{R}^{d_b} \rightarrow \mathbb{R}^d$ and $v(\cdot) : \mathbb{R}^{d_b} \rightarrow \mathbb{R}^d$, respectively. The inputs of both $k(\cdot)$ and $v(\cdot)$ are from B.

Recall cross-attention



Cross Attention:

Given two sets $A = [a_1, a_2, ..., a_{N_a}] \in \mathbb{R}^{d_a \times N_a}$ and $B = [b_1, b_2, ..., b_{N_b}] \in \mathbb{R}^{d_b \times N_b}$, the query vectors Q are constructed with a linear function $q(\cdot) : \mathbb{R}^{d_a} \to \mathbb{R}^{d}$ by taking elements of A as input. Similarly, we construct keys K and values V with $k(\cdot) : \mathbb{R}^{d_b} \to \mathbb{R}^{d}$ and $v(\cdot) : \mathbb{R}^{d_b} \to \mathbb{R}^{d}$, respectively. The inputs of both $k(\cdot)$ and $v(\cdot)$ are from B.

Output of Cross Attention:

Each column in the output $o(a_j, B)$ of Eq. (3) can be written as:

$$o(a_j, B) = \sum_{i=1}^{N_b} v(b_i) \cdot \frac{1}{Z(a_j, B)} \cdot \exp\left(\frac{q(a_j)^\top k(b_i)}{\sqrt{d}}\right)$$

Where $Z(a_j, B) = \sum_{i=1}^{N_b} \exp\left(\frac{q(a_j)^\top k(b_i)}{\sqrt{d}}\right)$ is a normalizing factor.

Recall cross-attention

Cross Attention:

Given two sets $A = [a_1, a_2, \ldots, a_{N_a}] \in \mathbb{R}^{d_a \times N_a}$ and $B = [b_1, b_2, \ldots, b_{N_b}] \in \mathbb{R}^{d_b \times N_b}$, the query vectors Q are constructed with a linear function $q(\cdot) : \mathbb{R}^{d_a} \rightarrow \mathbb{R}^{d}$ by taking elements of A as input. Similarly, we construct keys K and values V with $k(\cdot) : \mathbb{R}^{d_b} \rightarrow \mathbb{R}^{d}$ and $v(\cdot) : \mathbb{R}^{d_b} \rightarrow \mathbb{R}^{d}$, respectively. The inputs of both $k(\cdot)$ and $v(\cdot)$ are from B.

Output of Cross Attention:

Each column in the output $o(a_j, B)$ of Eq. (3) can be written as:

$$o(a_j, B) = \sum_{i=1}^{N_b} v(b_i) \cdot \frac{1}{Z(a_j, B)} \cdot \exp\left(\frac{q(a_j)^\top k(b_i)}{\sqrt{d}}\right)$$

Where $Z(a_j, B) = \sum_{i=1}^{N_b} \exp\left(\frac{q(a_j)^\top k(b_i)}{\sqrt{d}}\right)$ is a normalizing factor.

Cross Attention Operator:

The cross attention operator between two sets is denoted as:

$$CrossAttn(A, B) = [o(a_1, B) \quad o(a_2, B) \quad \ldots \quad o(a_{N_a}, B)] \in \mathbb{R}^{d \times N_a}$$

Recall relation between self and cross-attention

$$\mathrm{SelfAttn}(\mathbf{A}) = \mathrm{CrossAttn}(\mathbf{A}, \mathbf{A}).$$

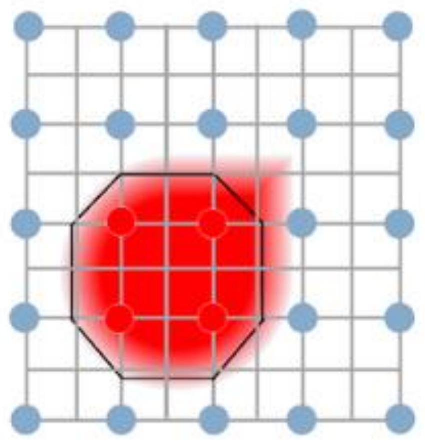
In other words, cross attention of A with itself gives back self-attention.

Occupancy field

An occupancy field is a volumetric representation that assigns a binary value (occupied or unoccupied) to each point in a 3D space, effectively capturing the spatial occupancy of objects within that volume.

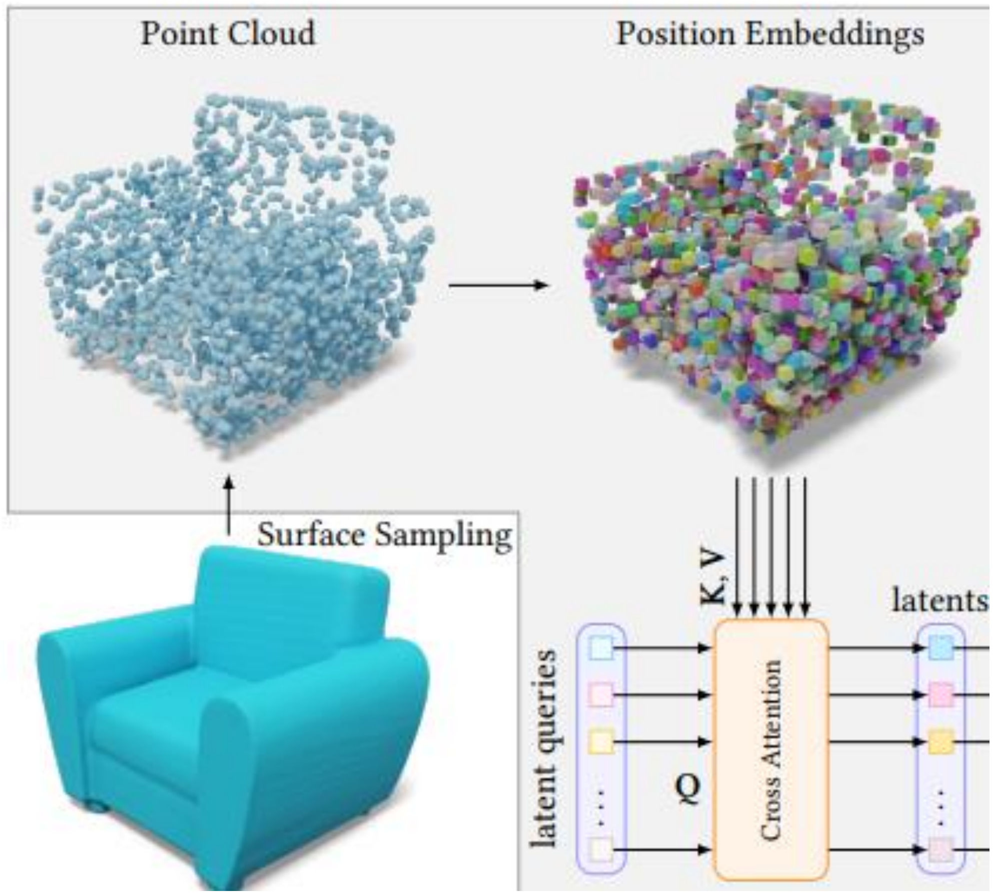
It serves as a compact and efficient way to encode the geometry and spatial layout of a scene or object.

Example



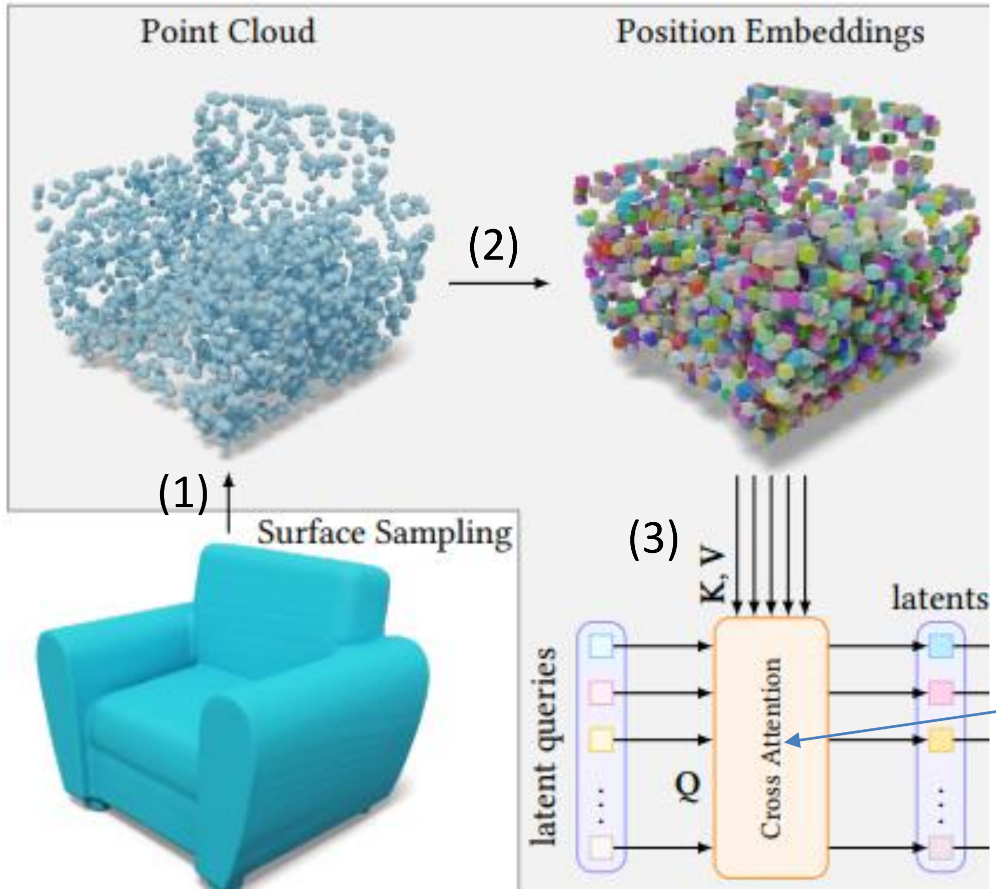
Mesh AutoEncoder : The encoder

We first sample points from the surface.

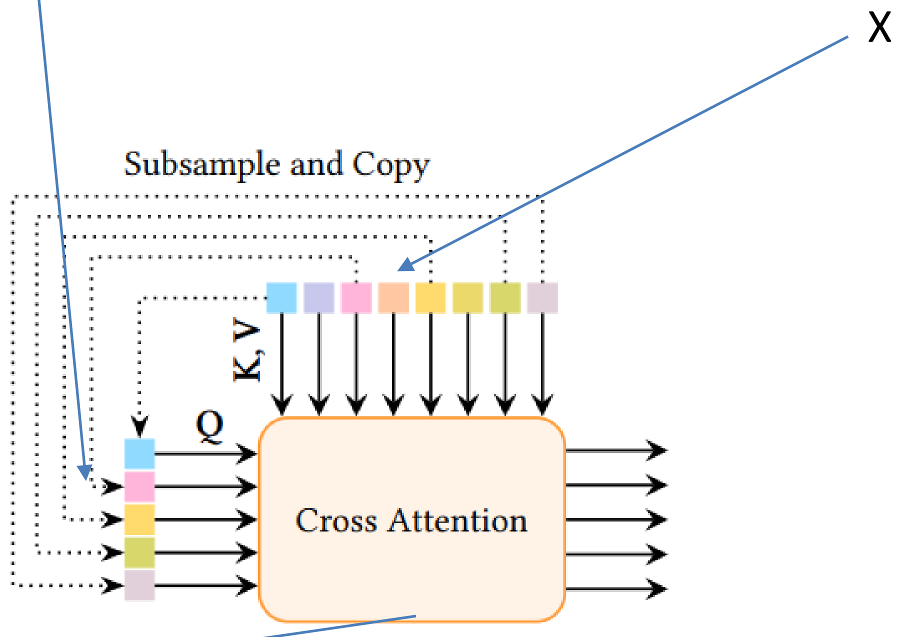


Mesh AutoEncoder : The encoder

We first sample points from the surface.



$\mathbf{X}_0 \doteq \mathrm{FPS}(\mathbf{X}) \in \mathbb{R}^{3\times M}$



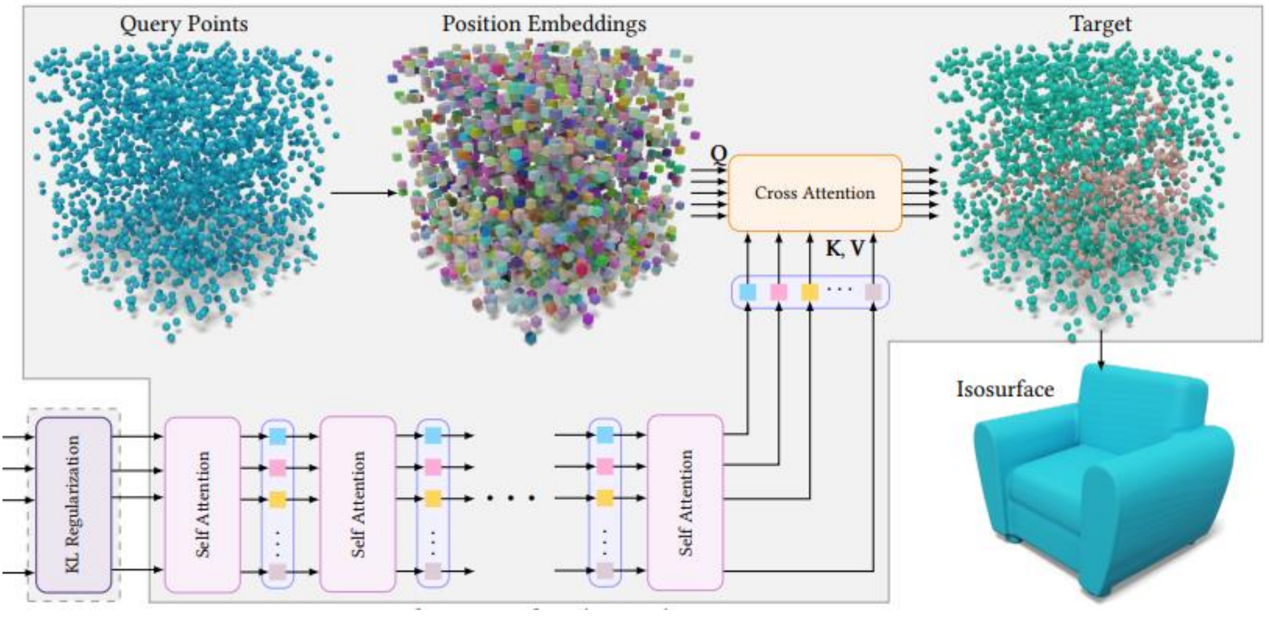
(b) Point Queries

$$\mathrm{Enc}_{\mathrm{points}}(\mathbf{X}) = \mathrm{CrossAttn}(\mathrm{PosEmb}(\mathbf{X}_0), \mathrm{PosEmb}(\mathbf{X}))$$

The cross attention is applied to X_0 and X can also be seen as a "partial" self attention.

Mesh AutoEncoder : The encoder

We first sample points from the surface.



(1) We sample query points, these points will be mappedTo inside and outside via the NN.
(2) Compute the positional embedding of each query point.

(3) Cross attention of these points with the encoder's output.
(4) The output of the cross attention is mapped to [0,1]
And then a binary classification loss is used to determine
If the output of each query point is inside or outside.
(5) An algorithm to compute the surface mesh from the occupancy is then utilized to reconstruct the output final mesh