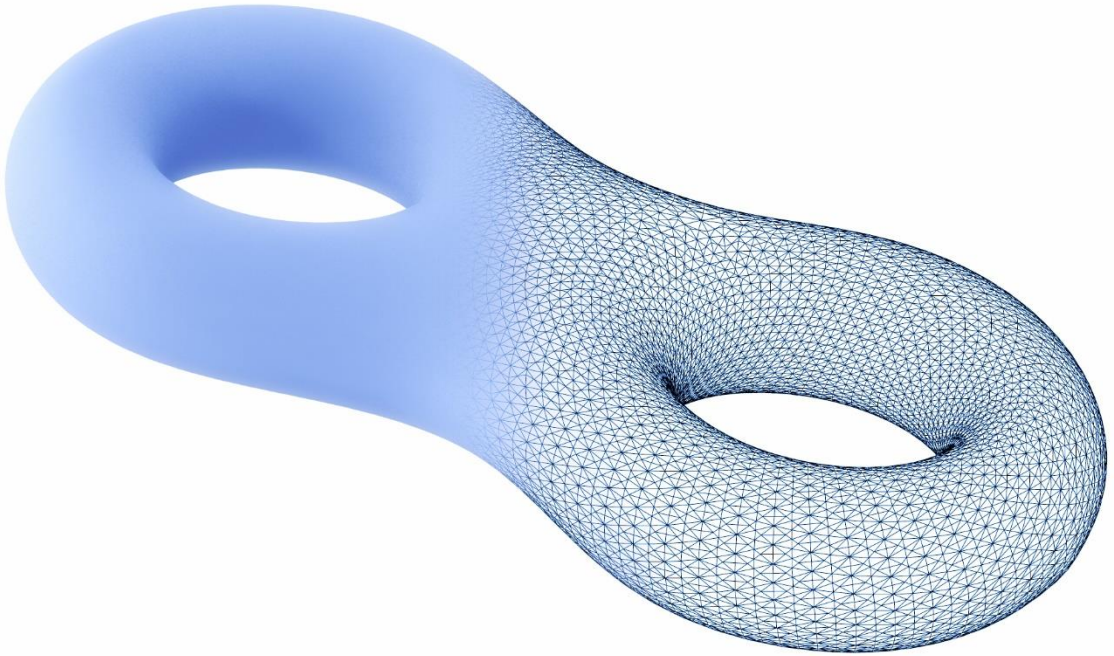
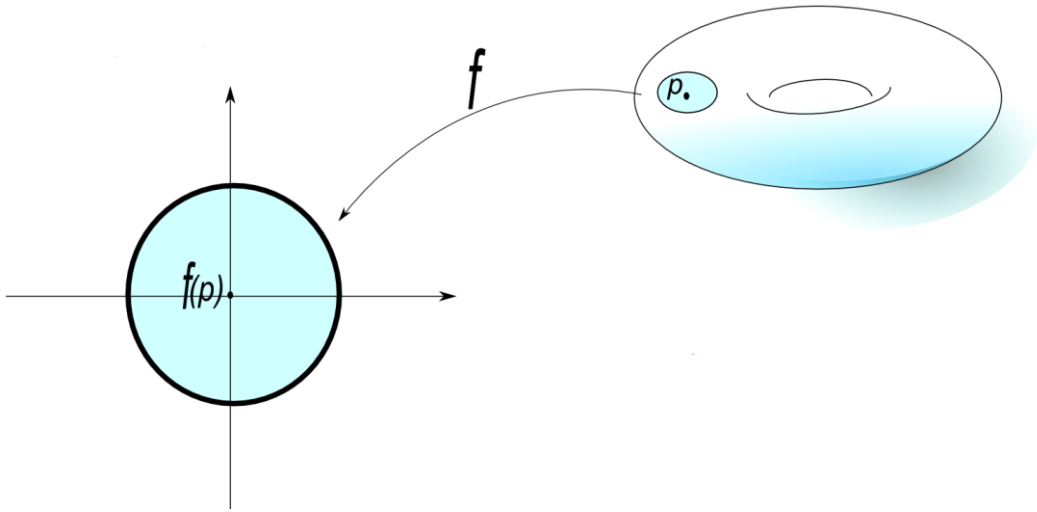
# Surfaces and Their Representations





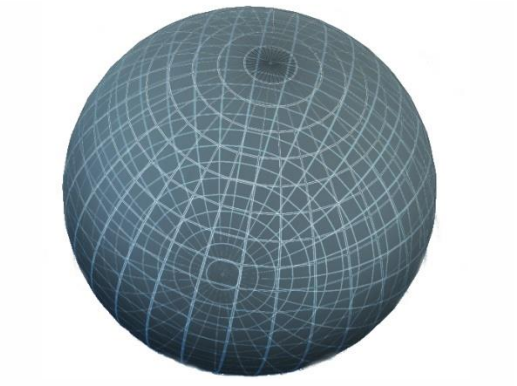

Mustafa Hajij

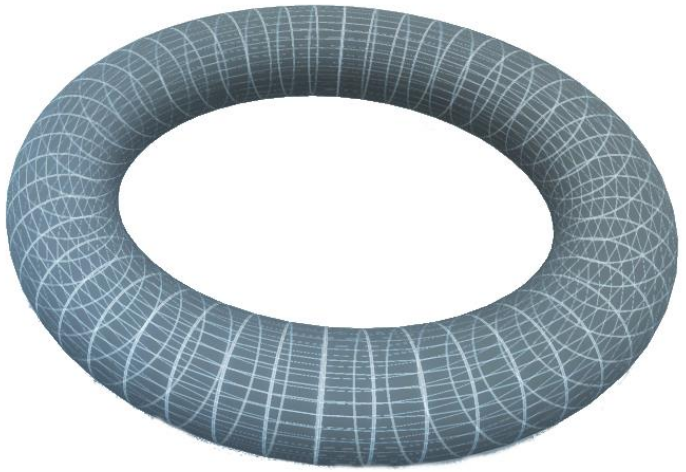
# Outline

- The concept of a surface
- Surfaces with boundary
- Topological properties of surfaces
- Triangulated surfaces
- Mesh data structures

# The concept of a surface

- Consider the following shapes :

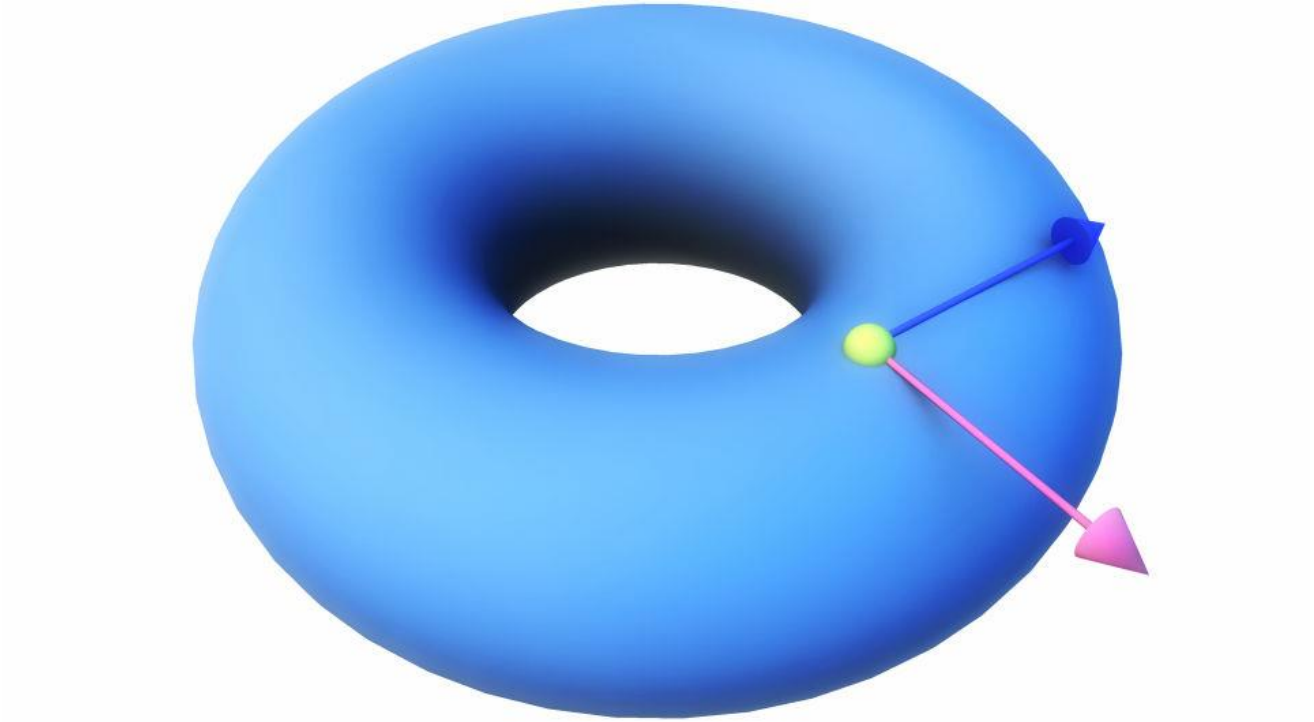




- In all examples that we will consider we will only consider the “surface” of the shape and not what is inside.

# 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.

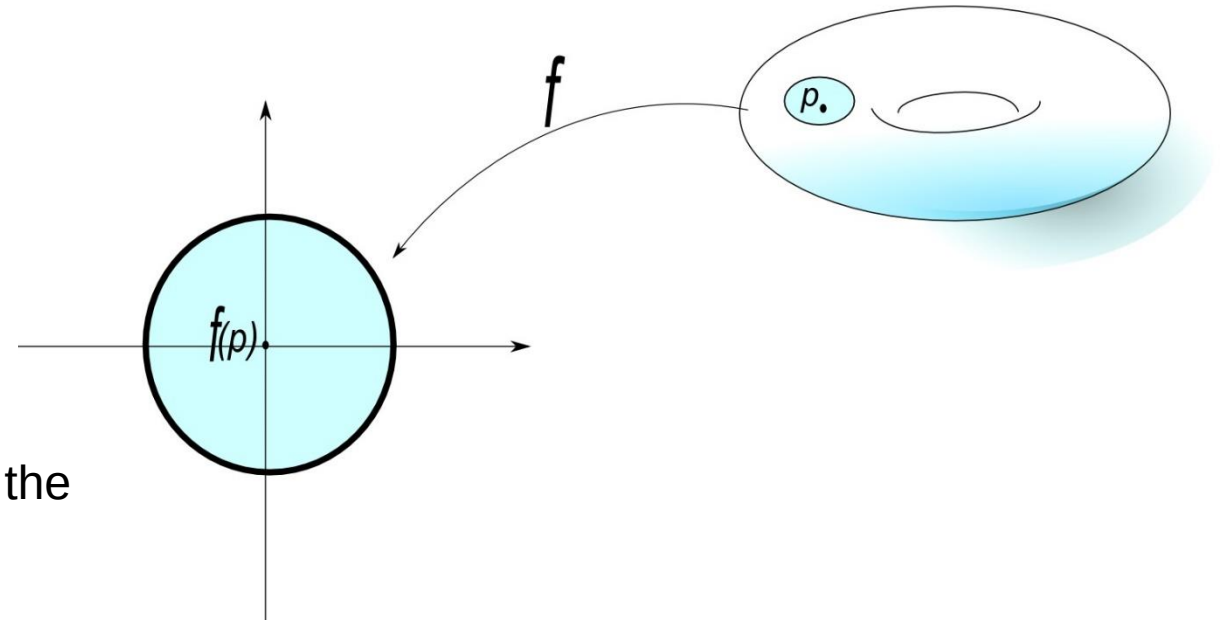
# The concept of a surface

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

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

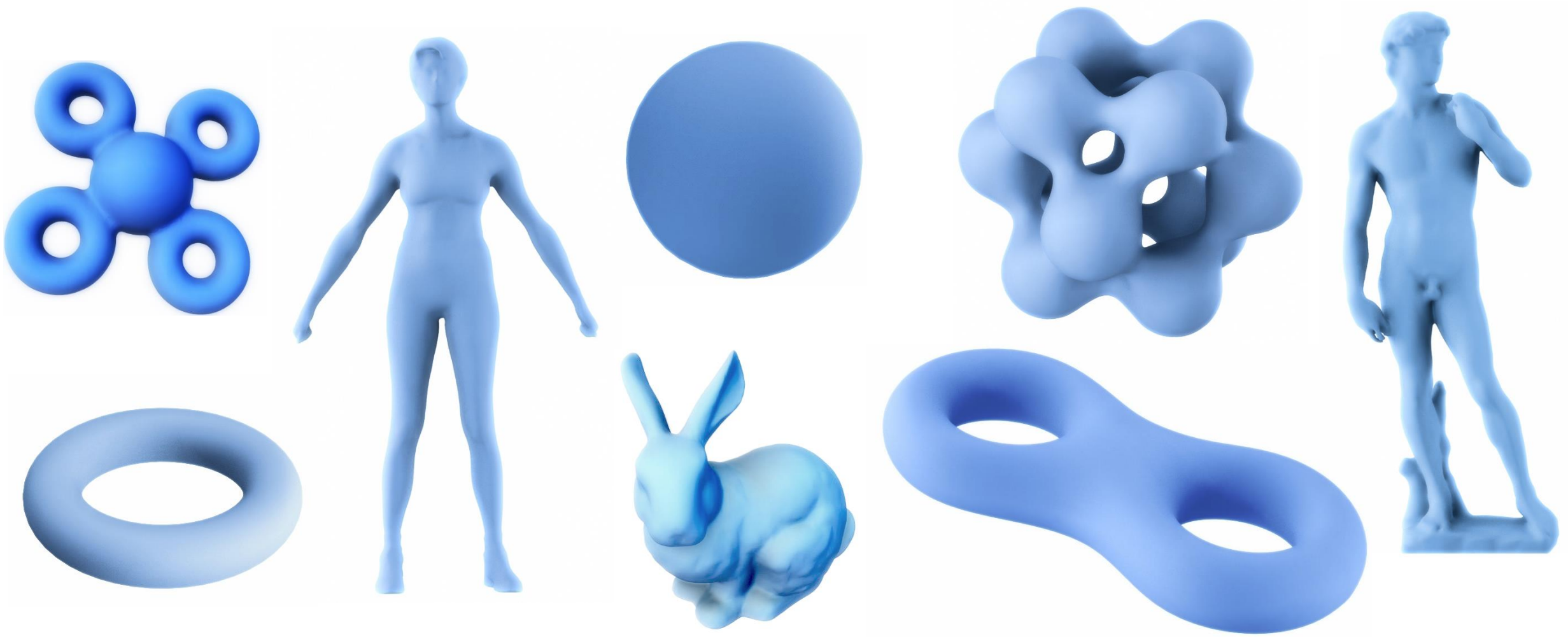
there exists a small disk

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



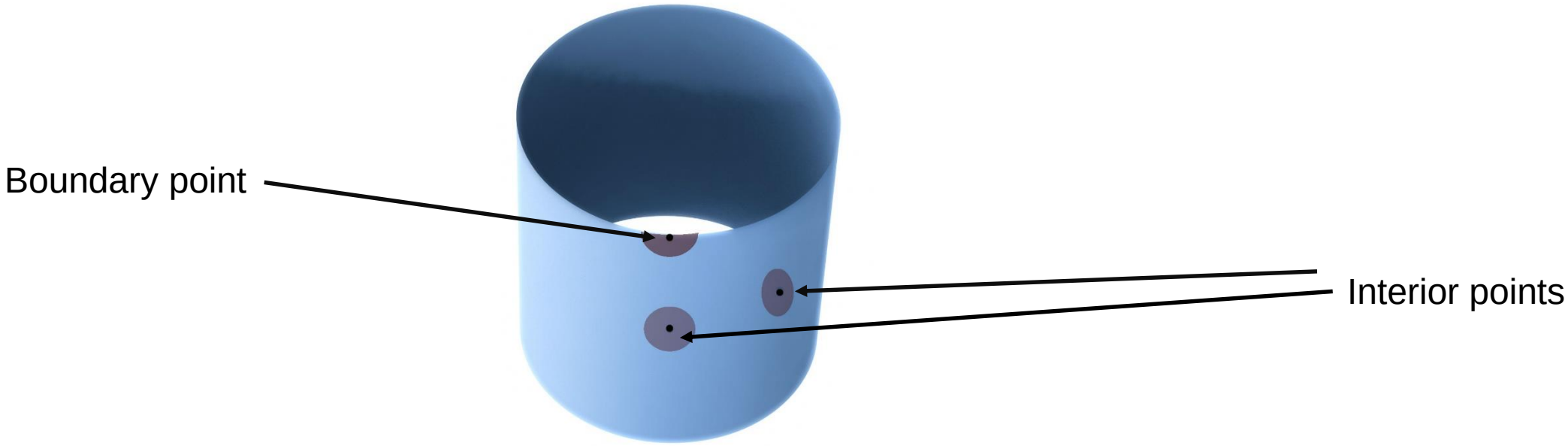
# The concept of a surface

So, the following shapes can be considered as surfaces :



# Surfaces with boundary

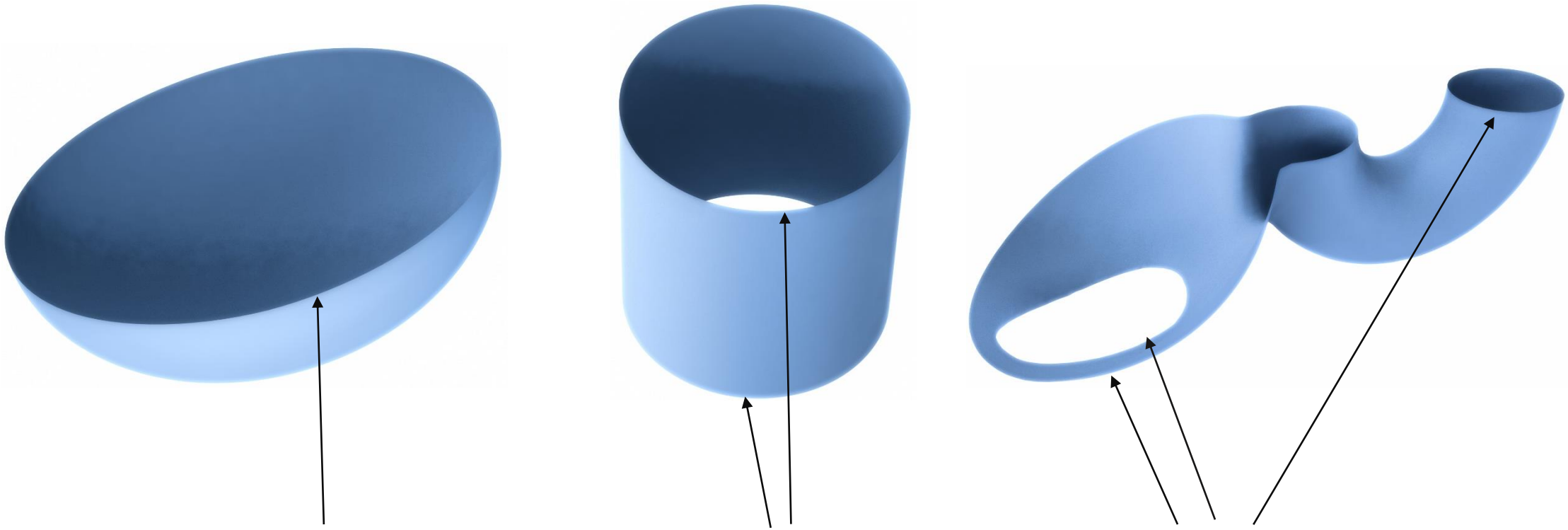
In this context, how do we consider a surface like the one here?




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



One boundary component

2 boundary components

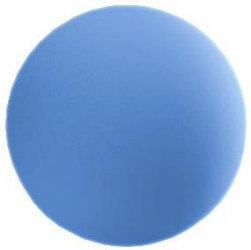
3 boundary components

# Topology of surfaces

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

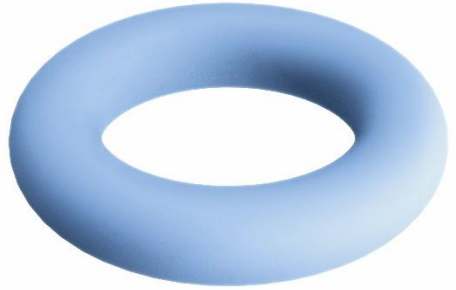
Topology studies the properties of shapes that do not change under continuous deformations.

=



=

=/=

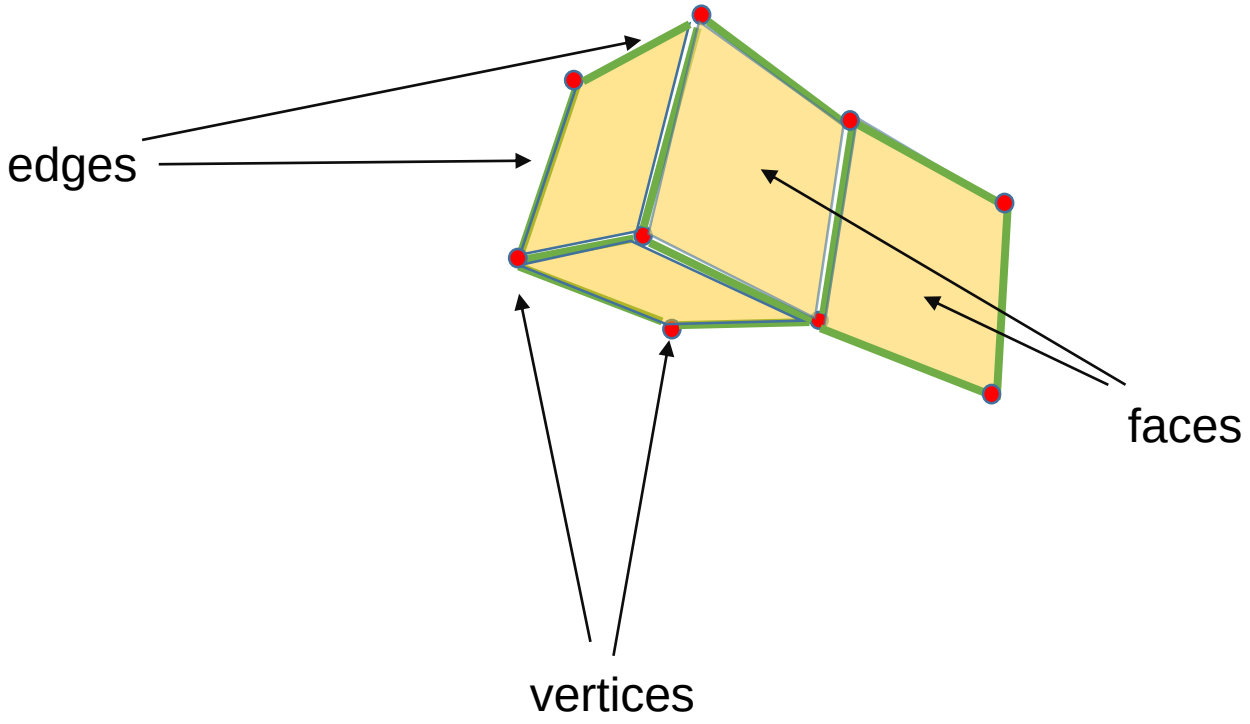


So topologically, the following objects are equivalent because we can deform each one of them contentiously without tearing into the other.

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.

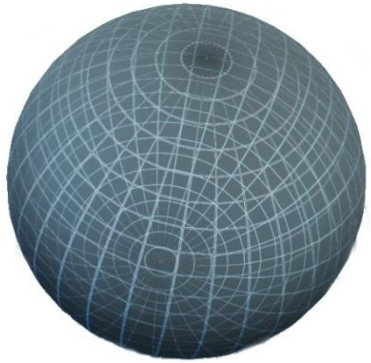


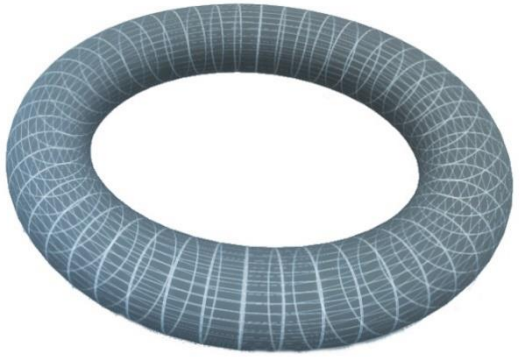
# Polygonal Mesh

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

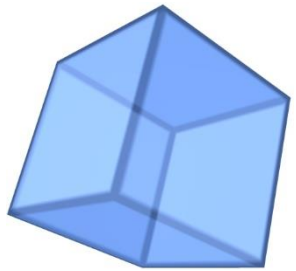




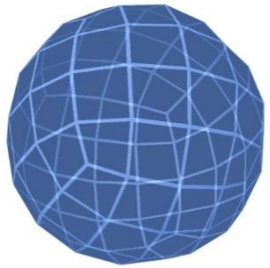
Geometric information of the mesh is given by specifying coordinates in the *3* dimensions to each vertex in the mesh.

# Topological properties of surfaces

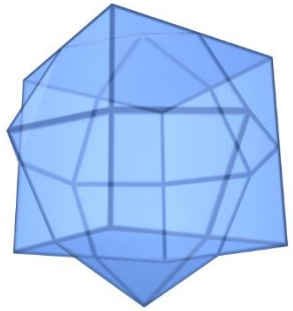
To understand what we mean by a topological we consider the following example.



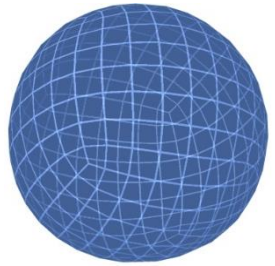
V-E+F=8-12+6=2



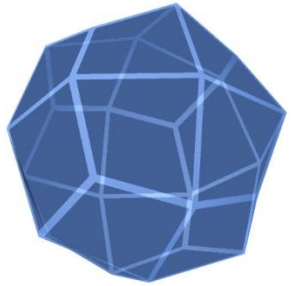
V-E+F=2



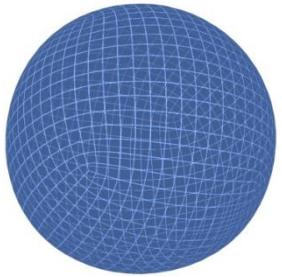
V-E+F=26-48+24=2



V-E+F=2



V-E+F=2

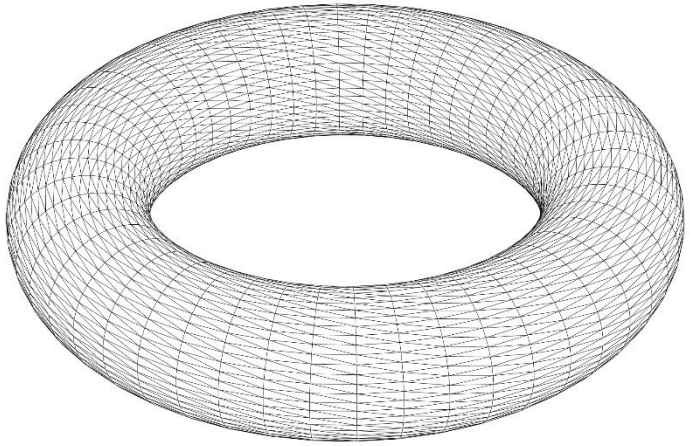


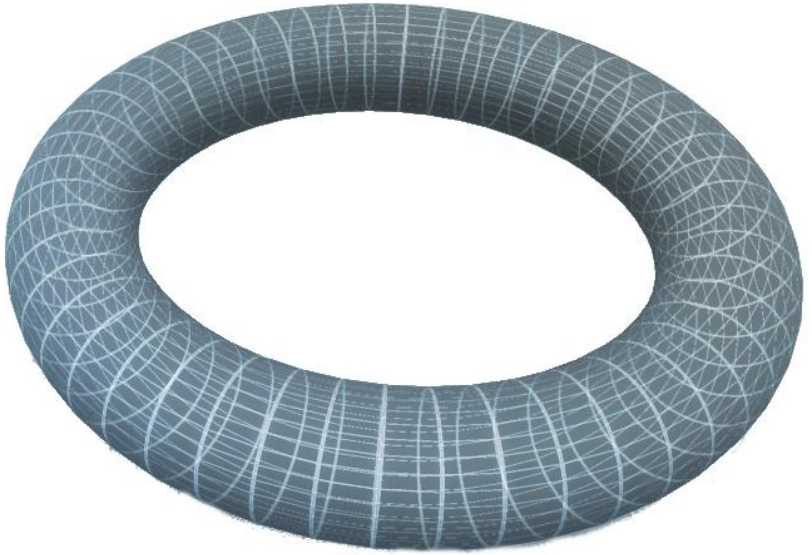
V-E+F=2

# 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.

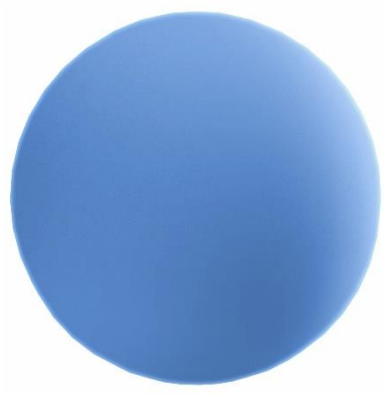
- On the other hand if we try to compute the same quantity for the torus we will get V-E+F=0. Regardless of the number of faces, edges and vertices we choose to build up the torus from.



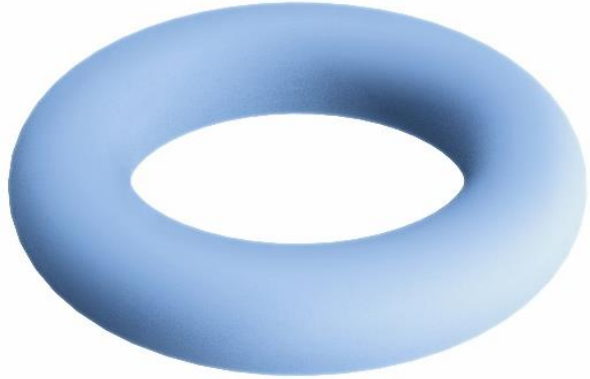


# Topological properties of surfaces, genus of a surface

- Informally, the genus of a connected and orientable surface is the number of “handles” in the surface.

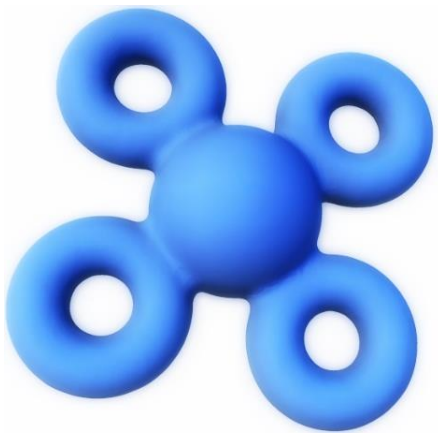


Genus zero surface



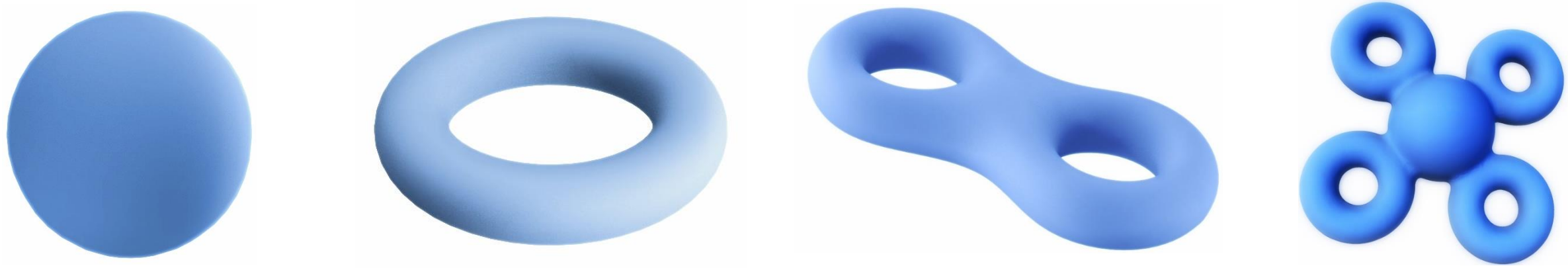
Genus one surface

Genus two surface



Genus four surface

# Topological properties of surfaces, genus of a surface



- If the surface is connected, oriented, and without boundary, then its Euler characteristic is related to the genus via the following formula :

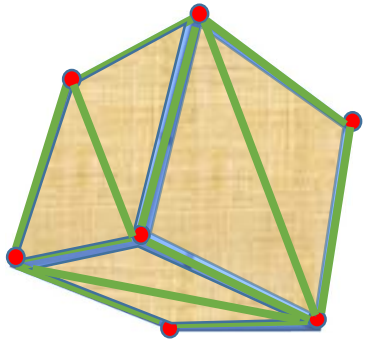
$$Euler\_Char = V - E + F = 2 - 2\,(genus)$$

- For a surface with *b* boundary component the formula becomes :

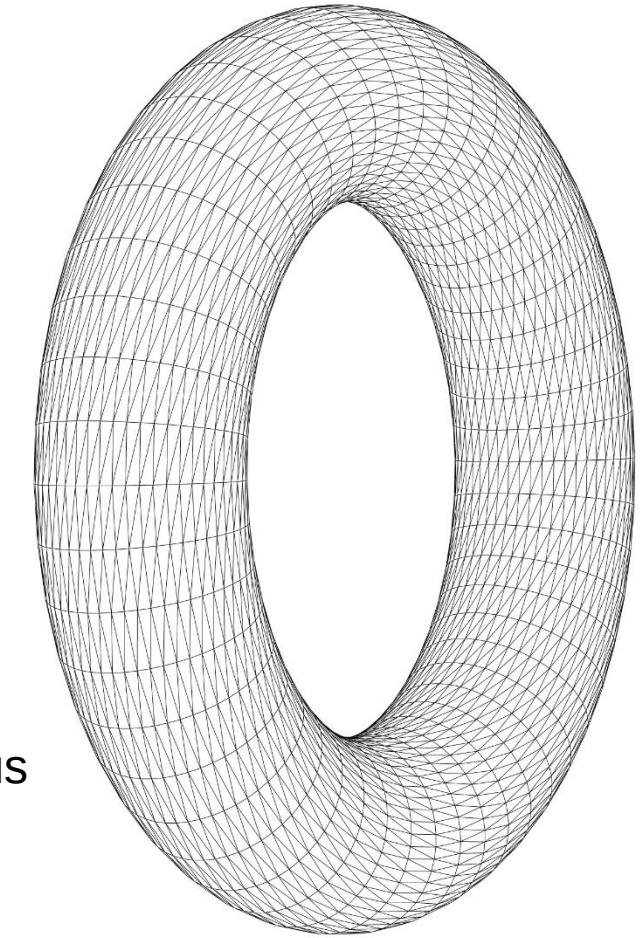
$$Euler\_Char = V - E + F = 2 - 2\,(genus) - b$$

# Triangulated Mesh

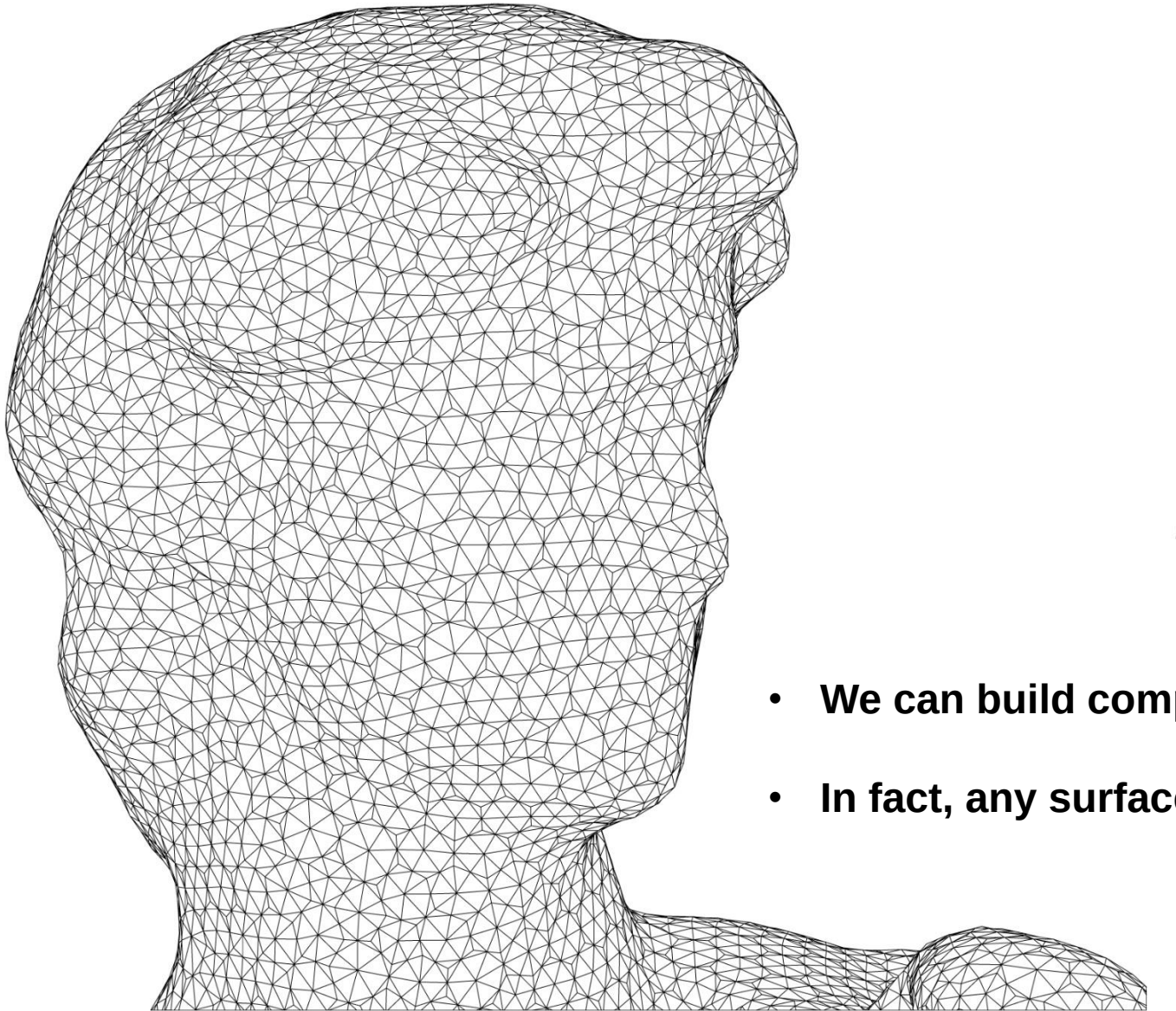
A mesh whose all its faces are triangles is called a triangulated mesh.

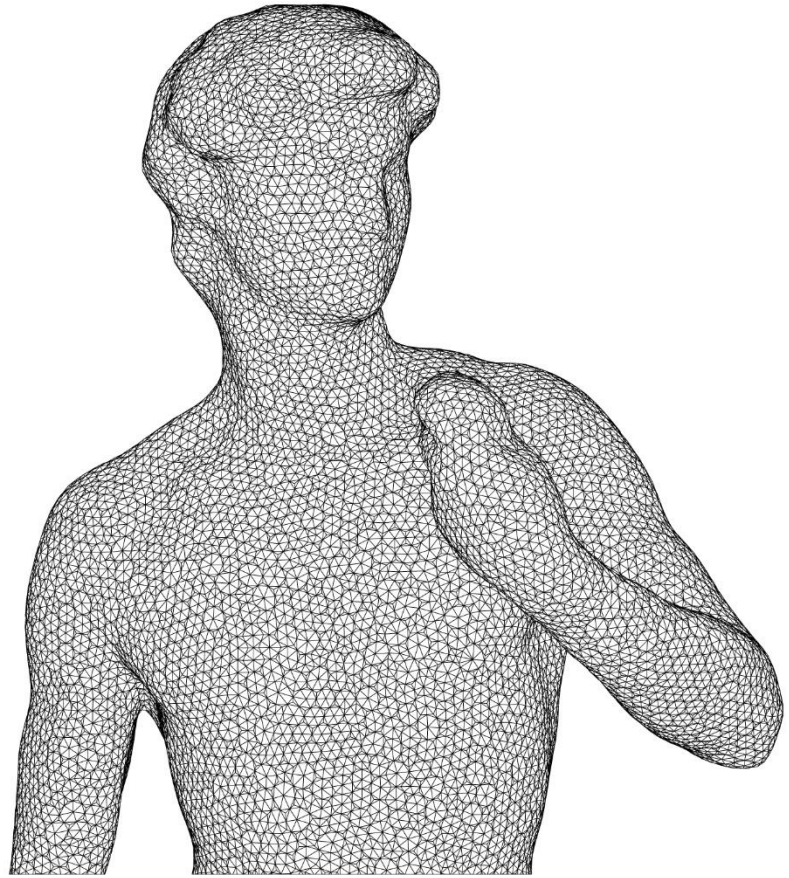


An example of a triangulated torus



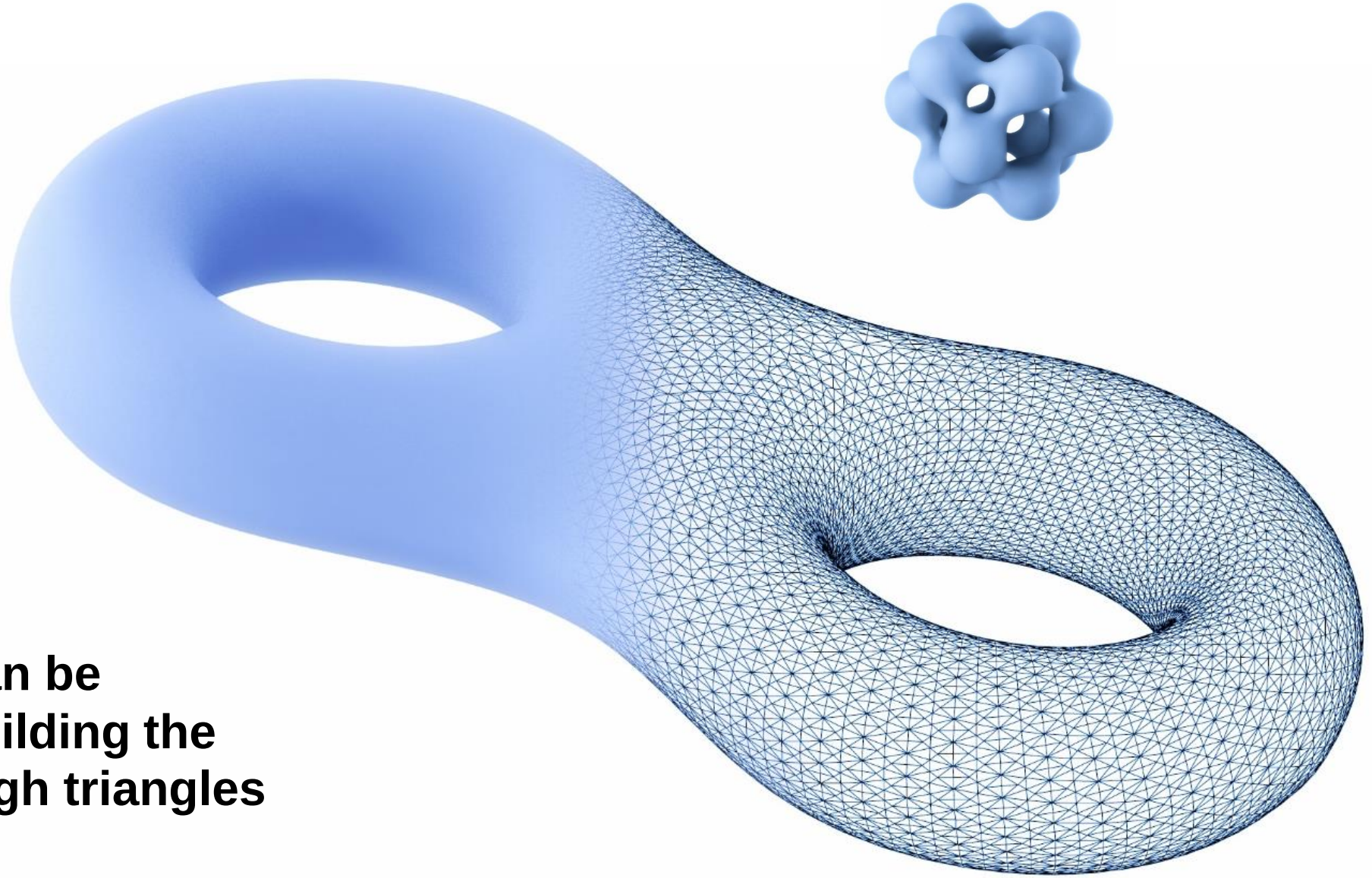
# Triangulated Mesh





- **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

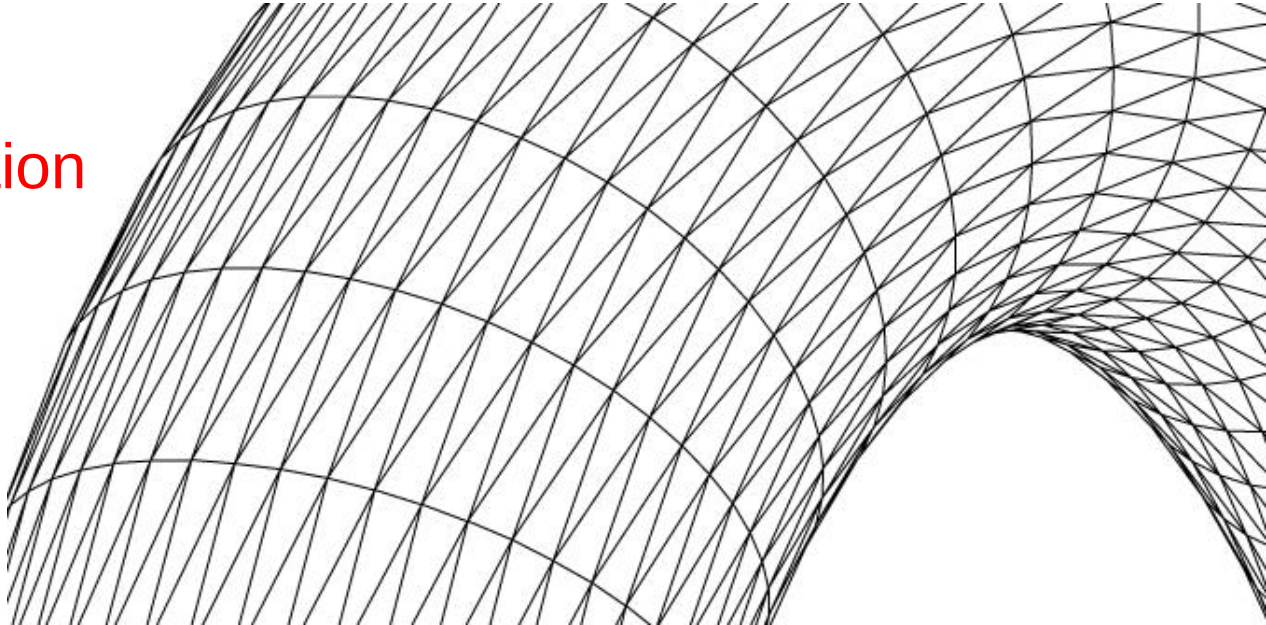
- Formally, a triangulated mesh is specified by the following data

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

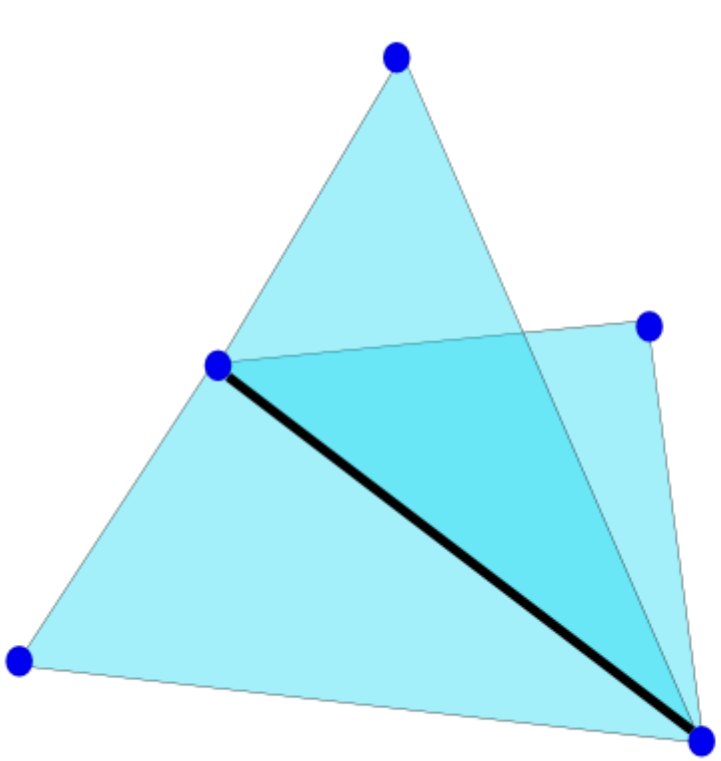
Set of edges $E = \{e_1, ..., e_k\}, \quad E \sqsubseteq V \times V$

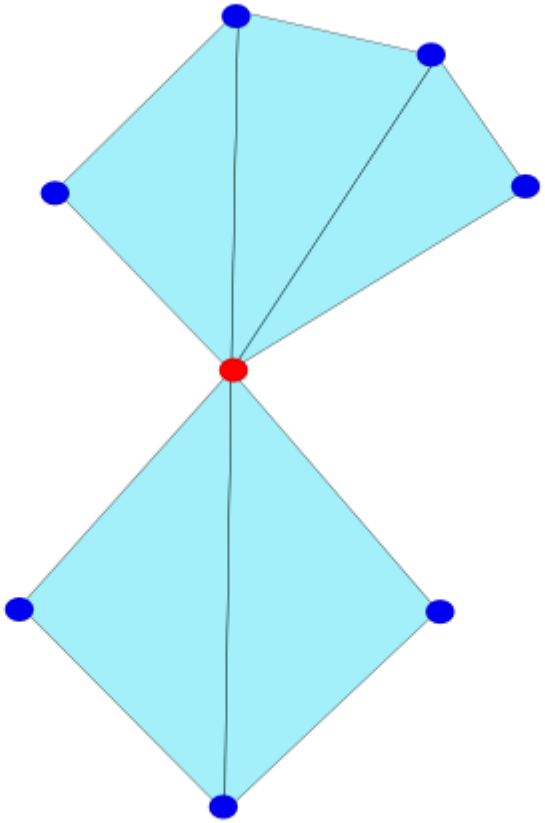
Set of faces $F = \{f_1, ..., f_k\}, \quad F \sqsubseteq V \times V \times V$

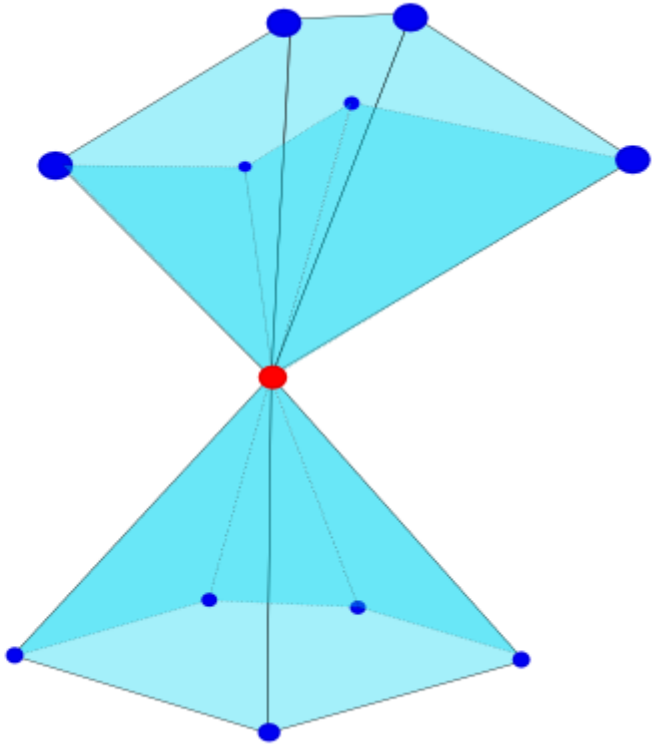
- Moreover, one usually specifies a position $\boldsymbol{p_i}$ **in** $\boldsymbol{R^3}$ for every vertex $\boldsymbol{v_i}$ .



# Triangulated 2-Manifold Mesh



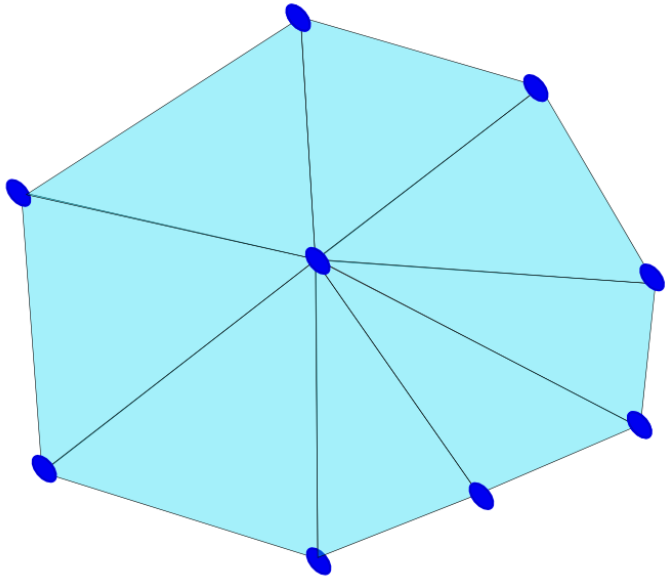


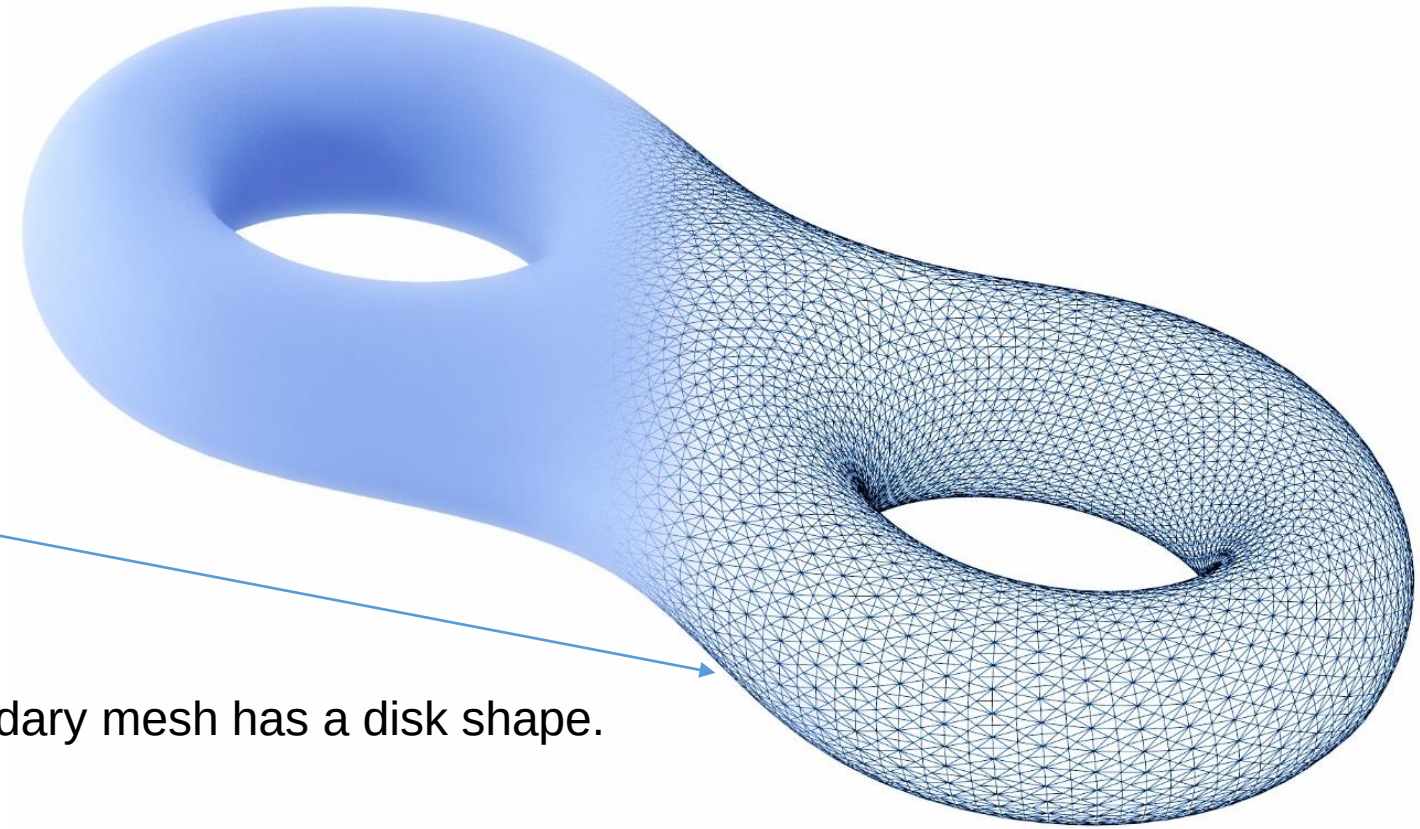


- These are examples of non-manifold meshes. On the left, non-manifold edge, and on the right we have non-manifold vertex

# 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.



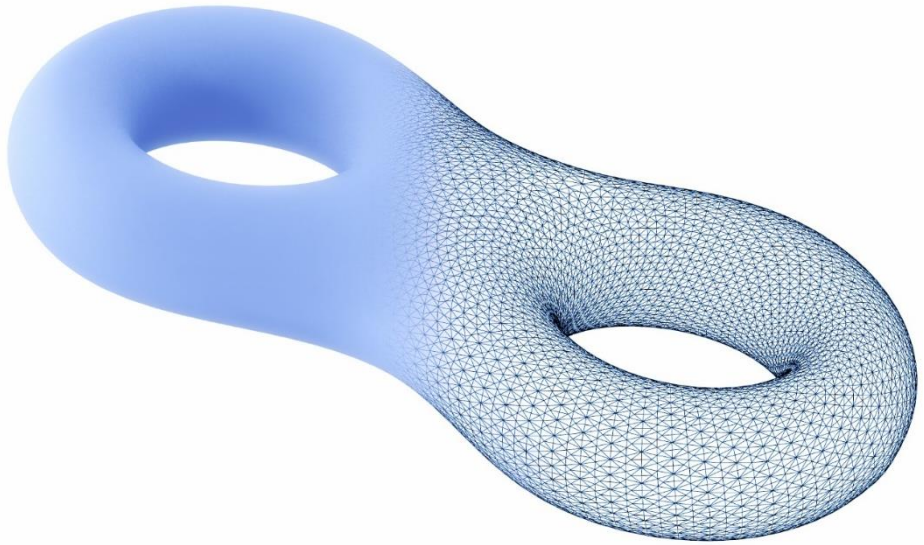


Locally, a triangulated *2*-manifold with no boundary mesh has a disk shape.

# Mesh Data Structure

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

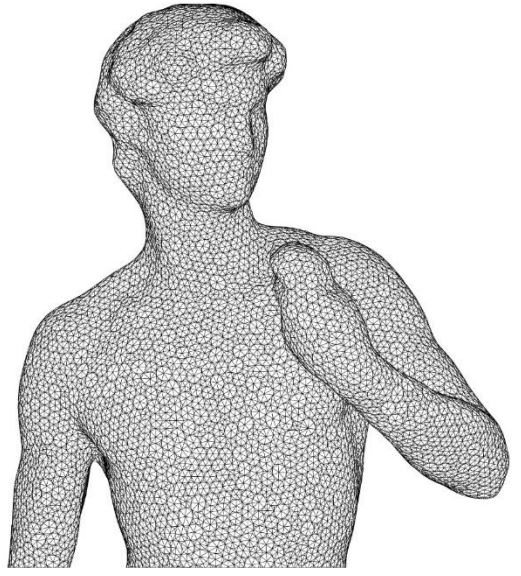
1. Topology : Adjacency information, For example, given a vertex v what are the vertices that v?
2. Geometry : 3d coordinates for each vertex
3. Attributes : Face or vertex normal, texture coordinates, etc.



# Mesh Data Structure

- What do we consider when we try to build a mesh data structure?

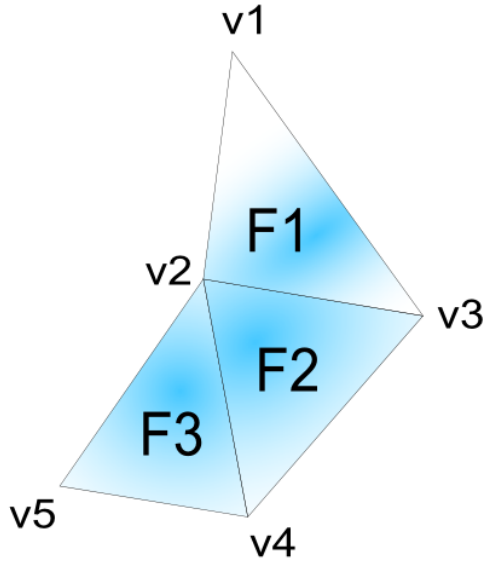
1. Topological considerations : manifold or non-manifold meshes, triangulated meshes or arbitrary polygon meshes, etc.
2. Algorithmic considerations : what kind of operations do we need to do on the mesh? What kind of additional structure we need to associate to the mesh data? Any memory constraints ?



# 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.

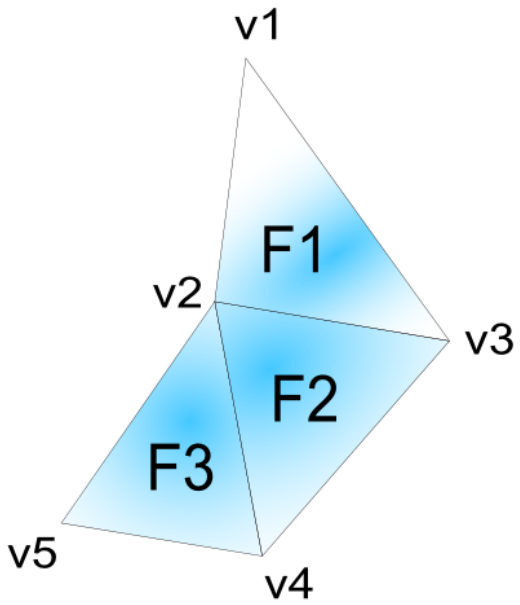



| Faces | | |
|---|---|---|
| x11 y11 z11 | x12 y12 z12 | x13 y13 z13 |
| x21 y21 z21 | x22 y22 z22 | x23 y23 z23 |
| x31 y31 z31 | x32 y32 z32 | x33 y33 z33 |

- 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 :

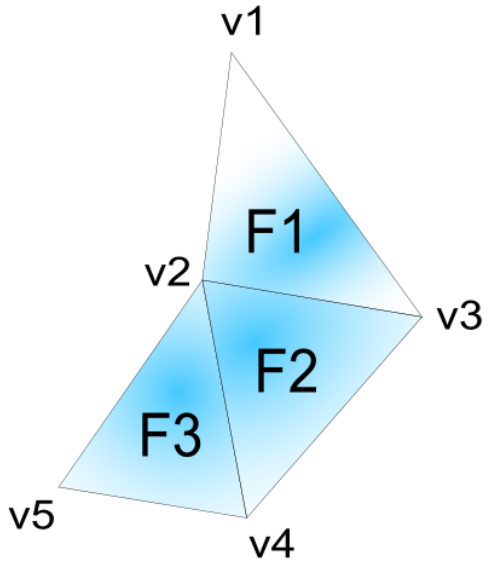



| Faces | | |
|---|---|---|
| x11 y11 z11 | x12 y12 z12 | x13 y13 z13 |
| x21 y21 z21 | x22 y22 z22 | x23 y23 z23 |
| x31 y31 z31 | x32 y32 z32 | x33 y33 z33 |

- 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 |

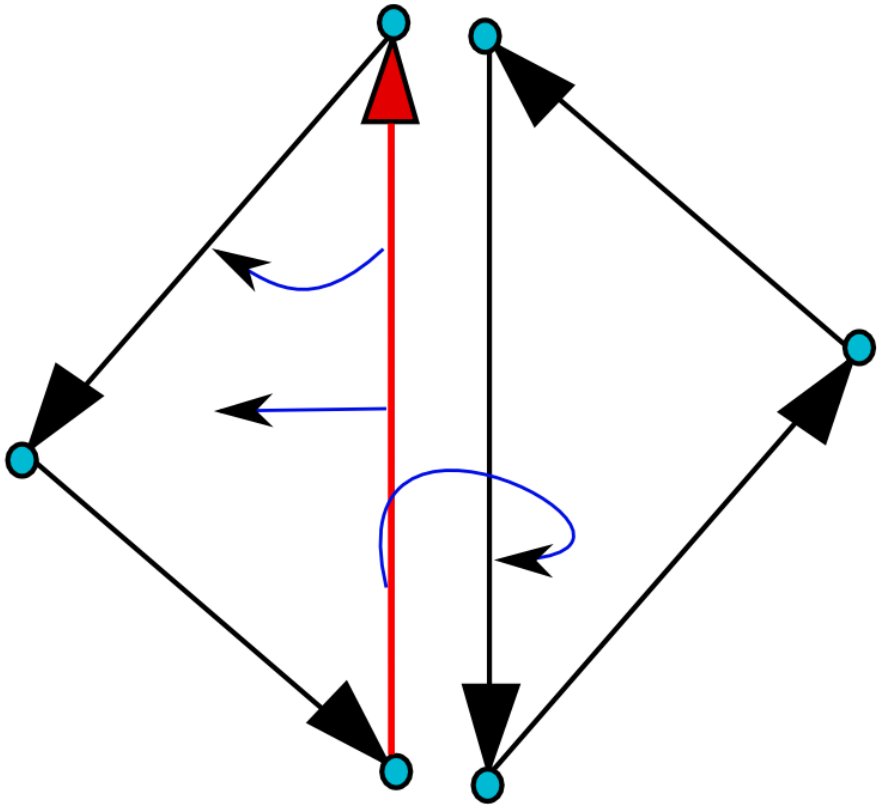
| Faces | | | |
|---|---|---|---|
| F1 | 1 | 2 | 3 |
| F2 | 2 | 4 | 3 |
| F3 | 2 | 5 | 4 |

- 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 Data Structure

## Half-edge data structure:

- The half-edge data structure is a mesh data structure that is used to store the topological adjacency information on the mesh.
- The idea is the following : if two faces are sharing an edge, then we split this edge into a pair “half-edges”. Clearly, this data structure can only be used to represent 2-manifolds mesh.

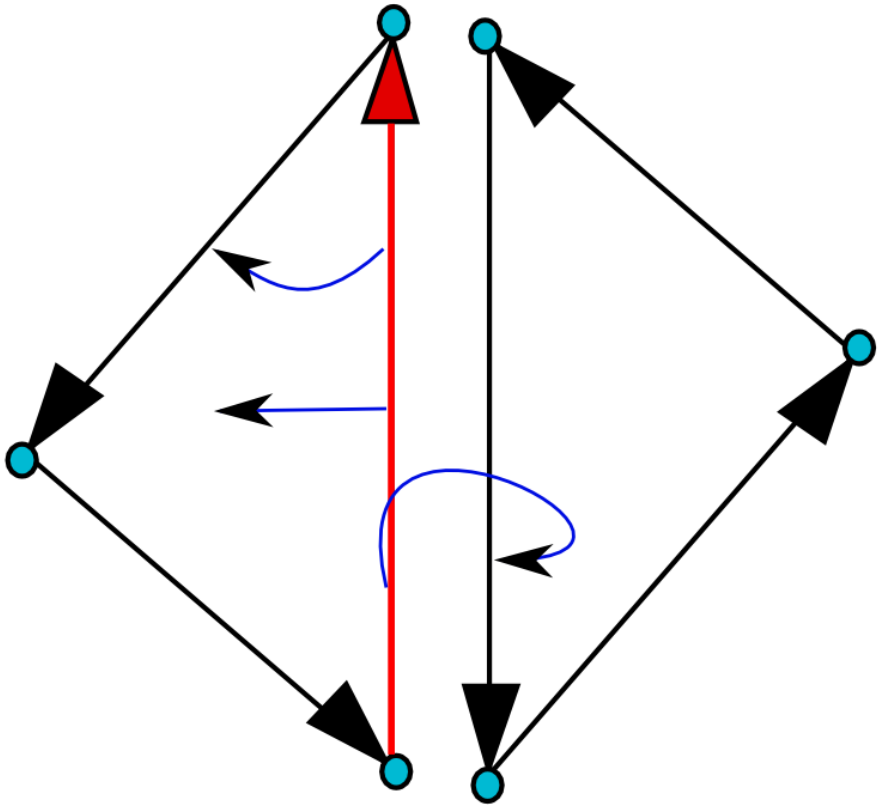


# Mesh Data Structure

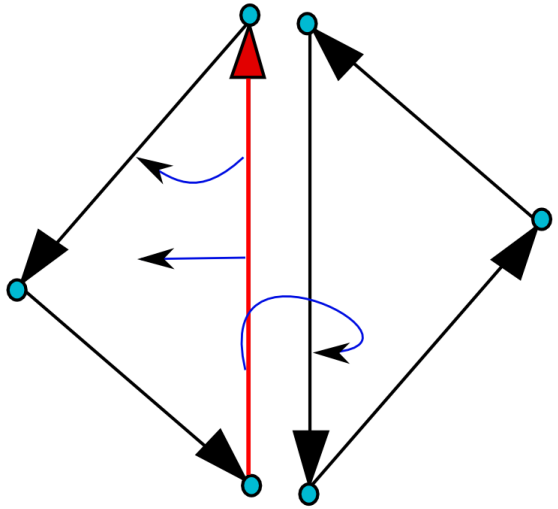
## Half-edge data structure:

- Half-edge class stores:

1. The vertex it points to
2. The adjacent face (the face it belongs to)
3. The opposite half-edge
4. The next half-edge (in counter clock-wise order)
5. The previous half-edge.
6. The edge it is attached to.

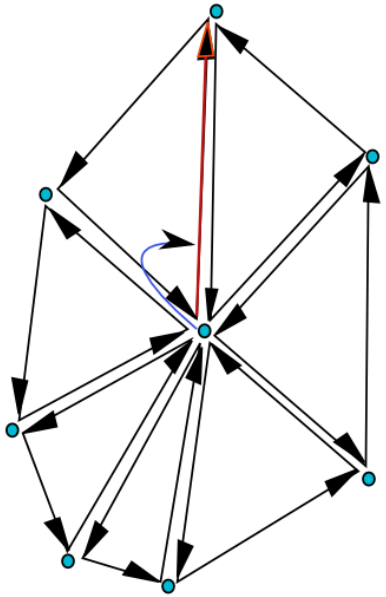


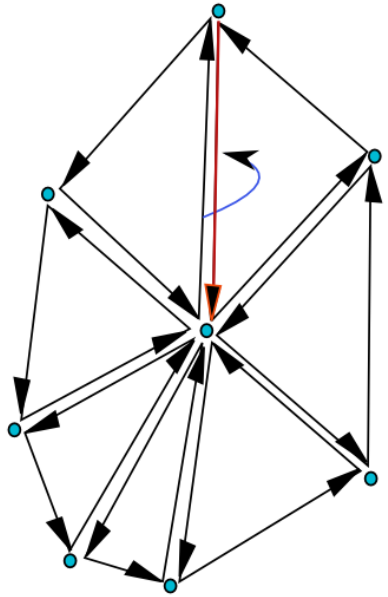
# Mesh Data Structure

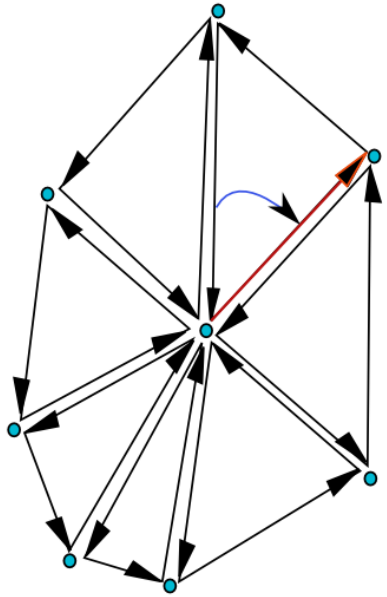


## Half-edge data structure:

- Given a vertex v, we can traverse all the vertices that are connected via an edge to v as follows:







# Mesh Data Structure

## Half-edge data structure:

- Given a vertex v, we can traverse all the edges that are connected in counter-clock-wise order
- Given a vertex v, we can traverse all faces connecting to v in a CCW order
- Given a vertex v, we can traverse all half-edges pointing to that vertex in a CCW order
- Given a vertex v, we can traverse all half-edges pointing away from v in a CCW order