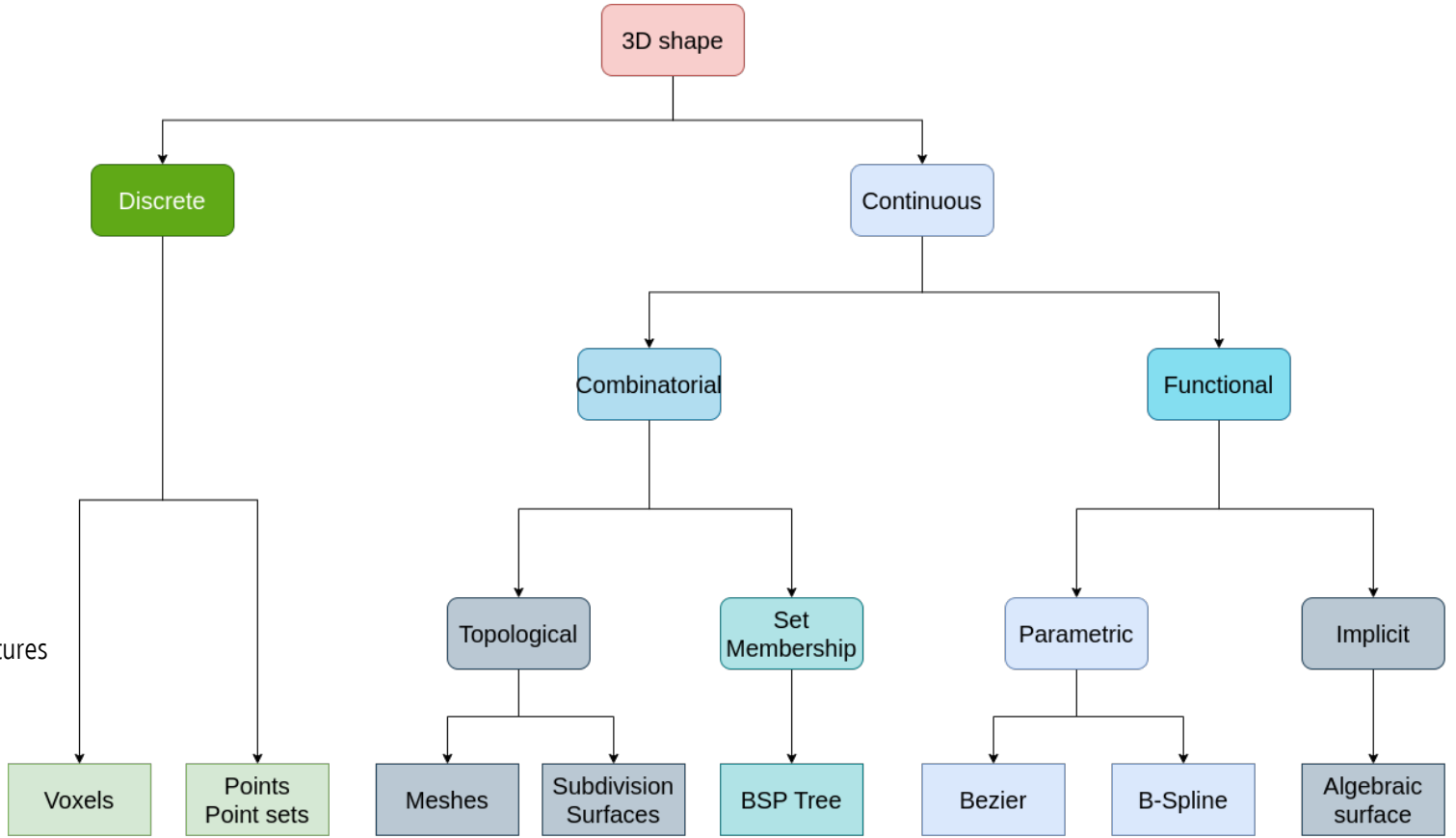
3D Models

Shape representations

Recap: shape representations

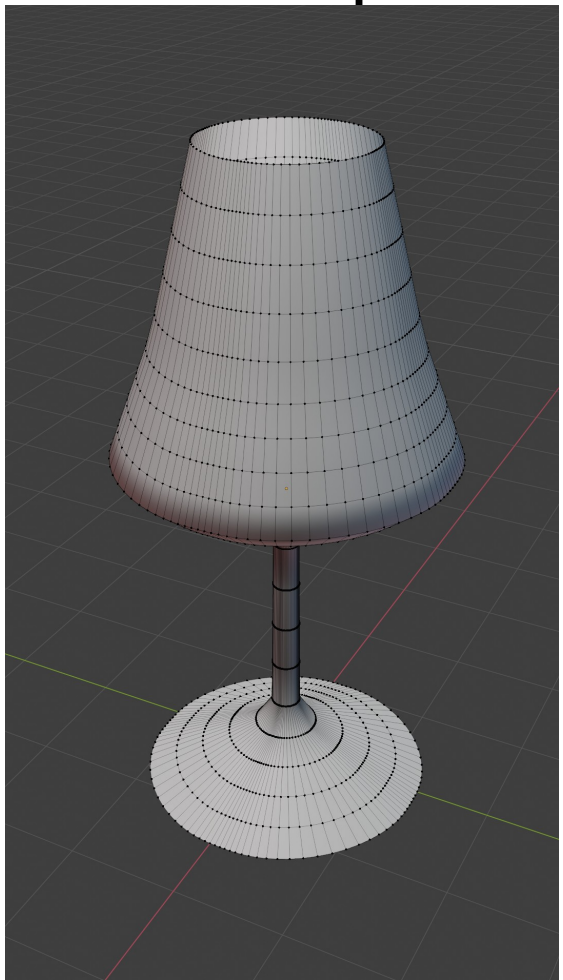
- Points
  - Point clouds
  - Particles and Particle systems
- **Surfaces:**
  - **Polygonal mesh**
  - Subdivision surfaces
  - **Parametric surfaces**
  - Implicit surfaces
- Volumetric objects/solids
  - Voxels
  - Space partitioning data-structures
- High-level structures
  - Scene graph

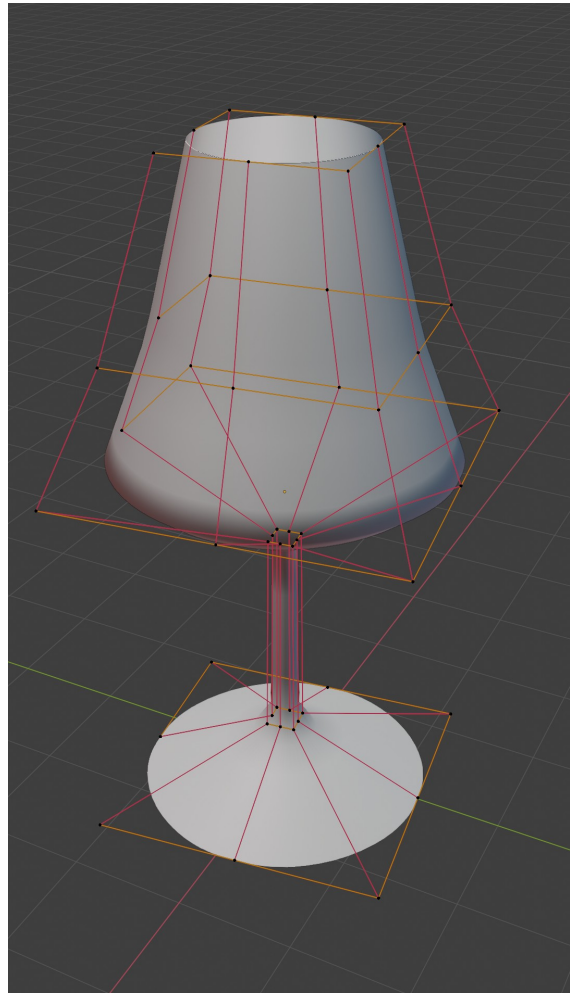


Foundations of 3D surface representation

Foundational **surface shape representations** found in geometrical modeling are*:

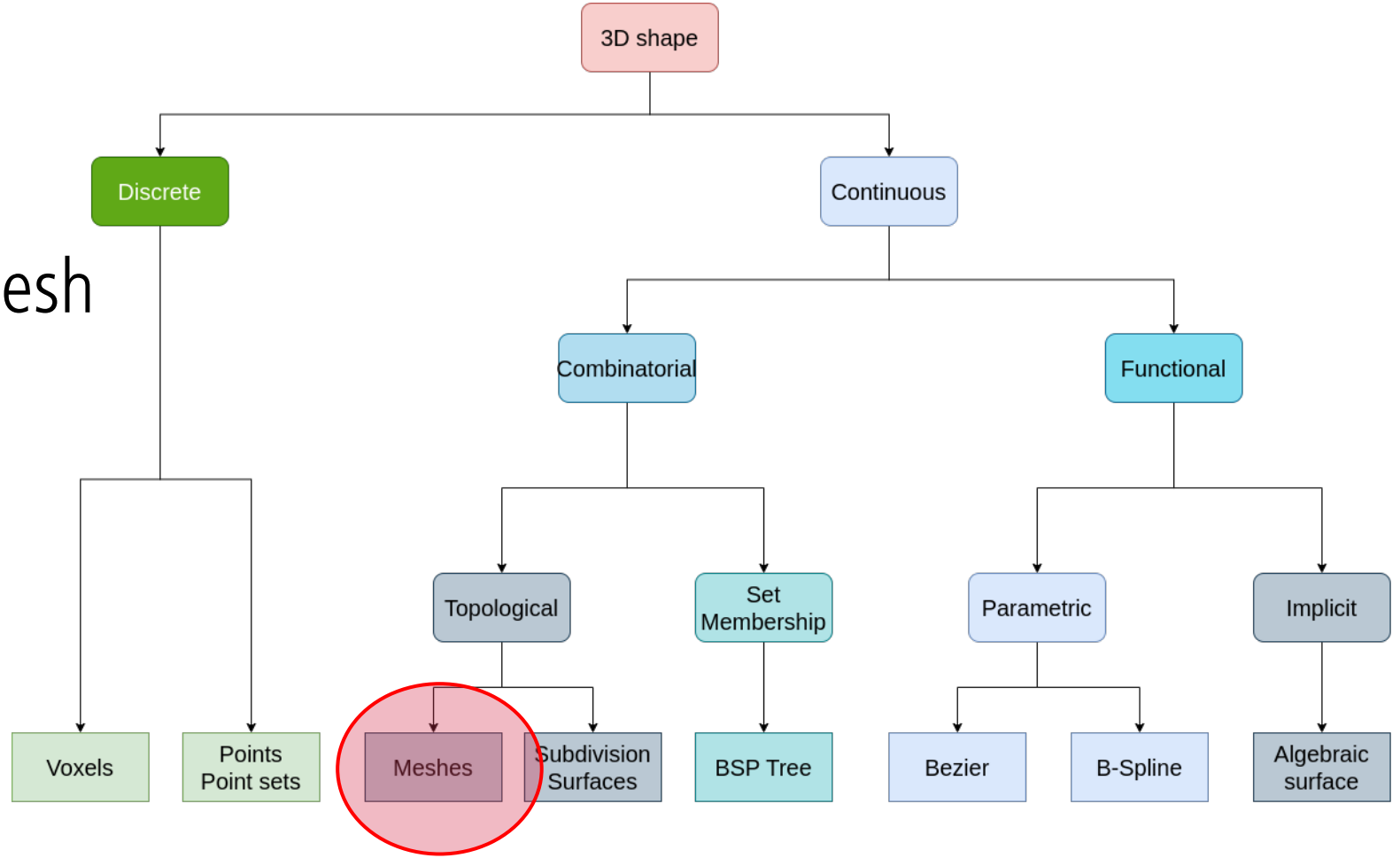
- **Polygon meshes**
- **Parametric surfaces**





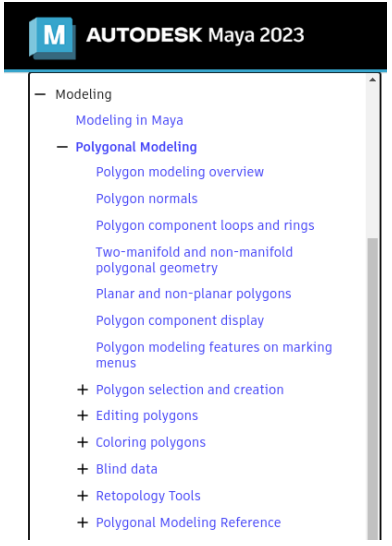
*Note that these representations are used to describe surface of the shape (a manifold – 2D surface in 3D world). Later, we will discuss how to describe interior of object (its volume). Interior of object can be described purely with spatially varying material enclosed in described surface. Also, advanced shape representations (e.g., voxels) can be used to efficiently describe the mesh. Since the topic of volumetric representation requires more knowledge about material and/or advanced shape representations, it will be covered later.

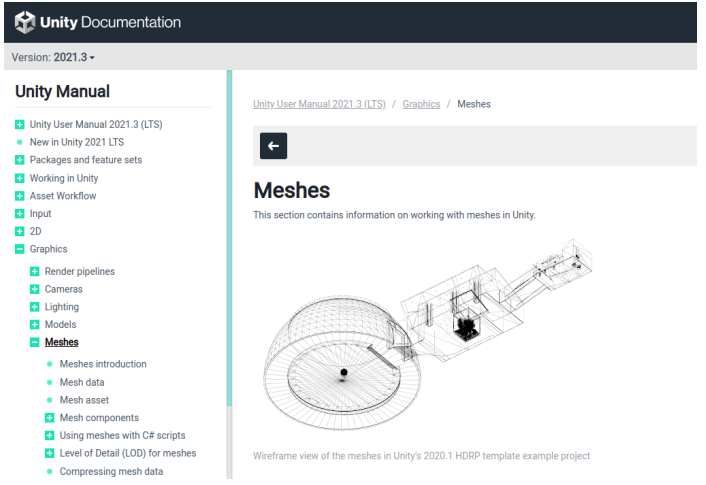
Polygonal mesh

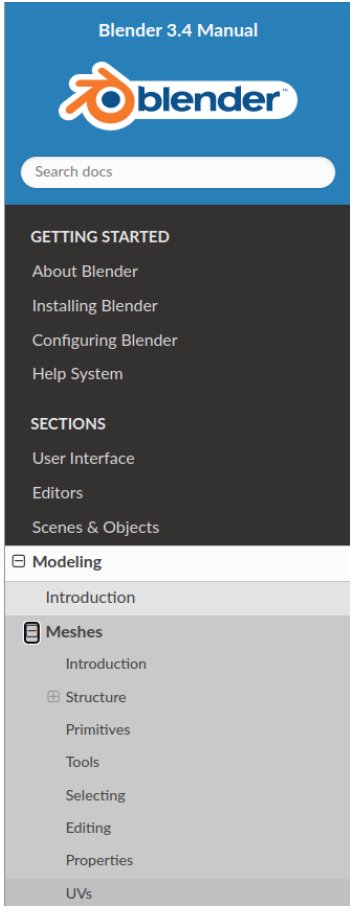


Polygon meshes

- **Polygon mesh** (shortly mesh) representation is one of the most oldest, popular and widespread geometry representation used in computer graphics
  - Very often, in professional DCC tools or game engines* we can find mesh representation that is used either for modeling or for rendering







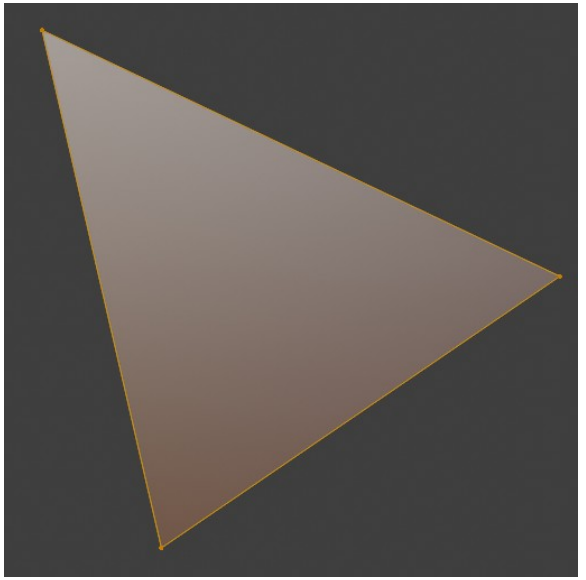
Blender: https://docs.blender.org/manual/en/latest/modeling/meshes/
Maya: https://help.autodesk.com/view/MAYAUL/2023/ENU/?guid=GI
Houdini: https://www.sidefx.com/docs/houdini/nodes/lop/mesh.html
Unity: https://docs.unity3d.com/Manual/class-Mesh.html
Unreal: https://docs.unrealengine.com/4.26/en-US/WorkingWithContent/Types/StaticMeshes/

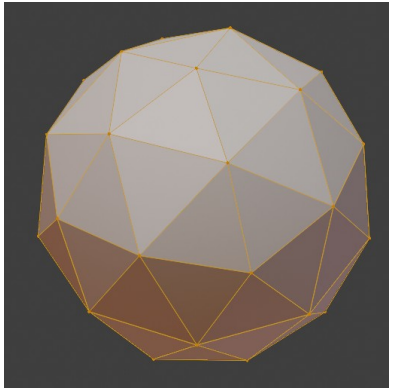
* Very often, mesh is commonly used for transporting models and scenes from DCC tools to game engines. DCC tools enable modeling using different shape representations, but in a lot of cases, all shapes are transformed to mesh representation and exported to other programs.

Mesh polygons: representation

Mesh building block: polygon

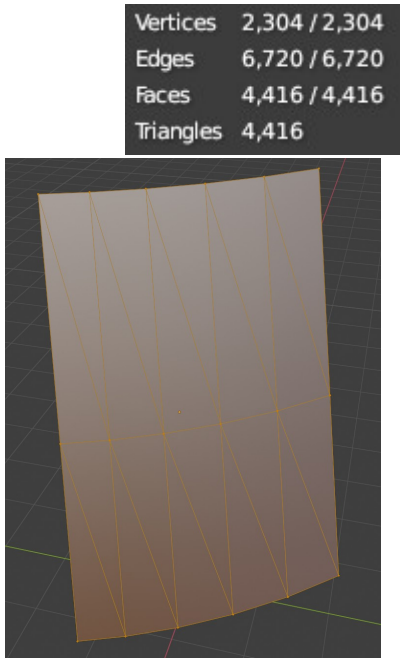
- Polygon is planar shape which is defined by connecting array of points.
- Individual points are called **vertices** (vertex, singular)
  - In 2D they are defined using two coordinates,e.g., (x,y)
  - In 3D they are defined using three coordinates, e.g., (x,y,z)
- Lines connecting two vertices are called **edges**.
- Once edges are presented and connect vertices we can define a **face**
  - Order of connecting vertices matter and it can be clockwise or counterclockwise – **winding direction**
  - Face orientation is defined by **normal** and normal depends on winding direction

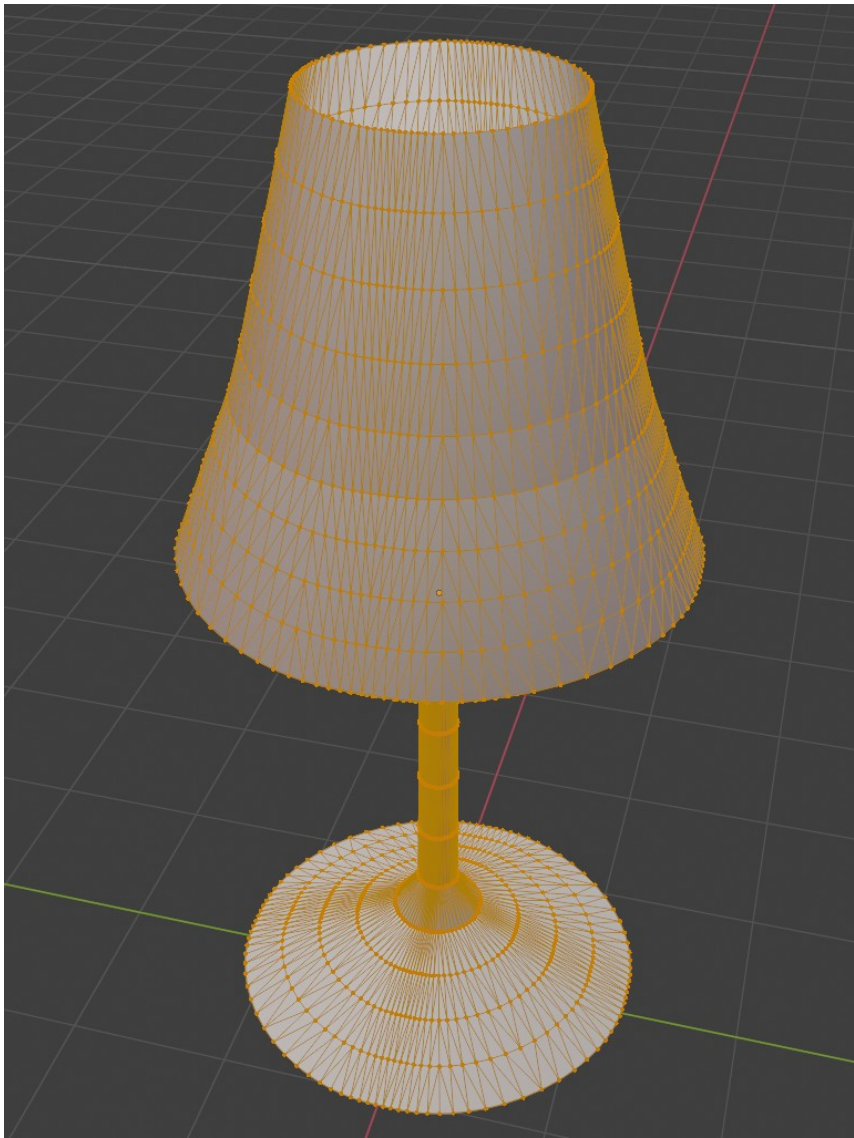




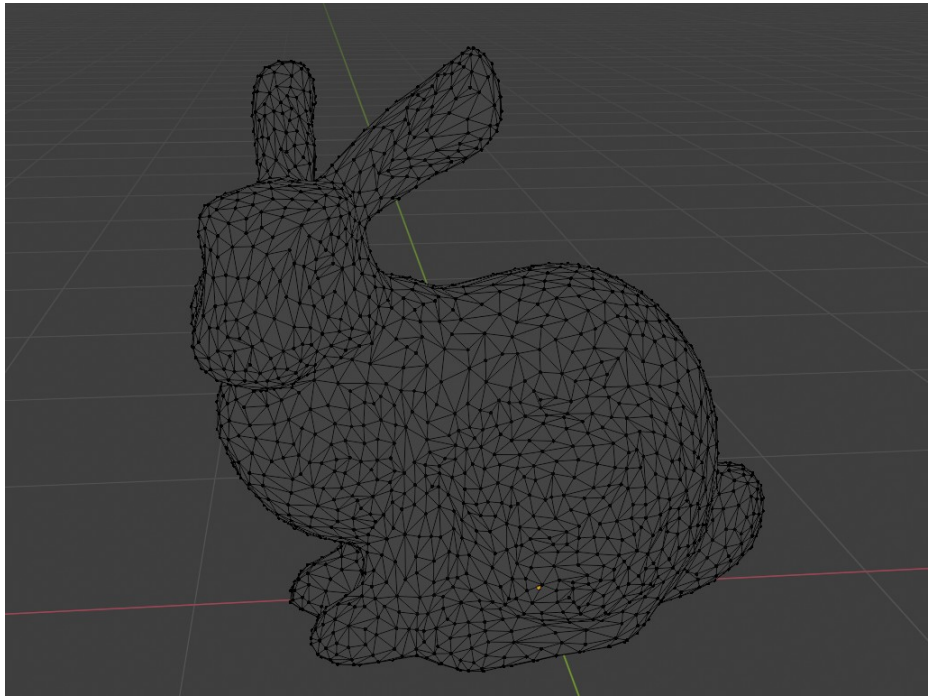
Types of polygons: triangle

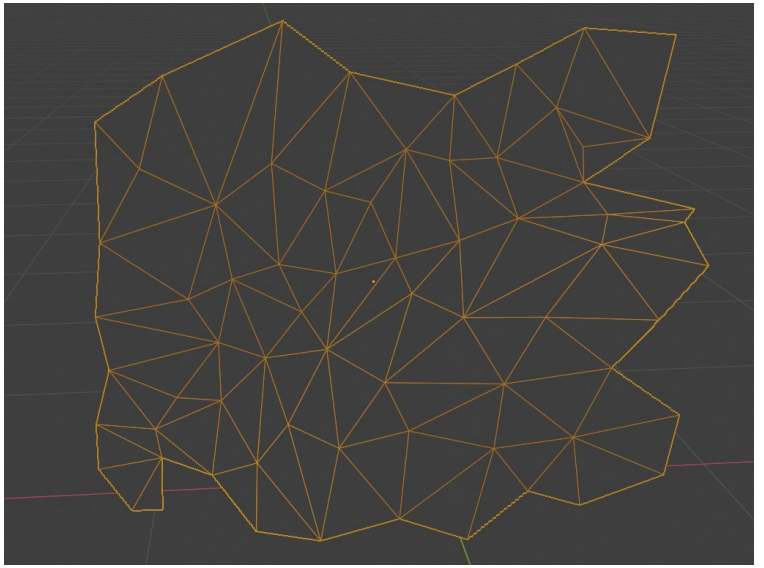
- Polygon with three vertices → **triangle polygon**.
  - Very important type of polygon in computer graphics!





Types of polygons: triangle





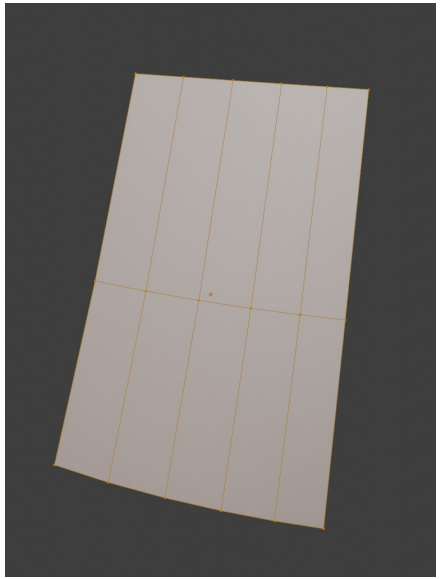
Triangle mesh

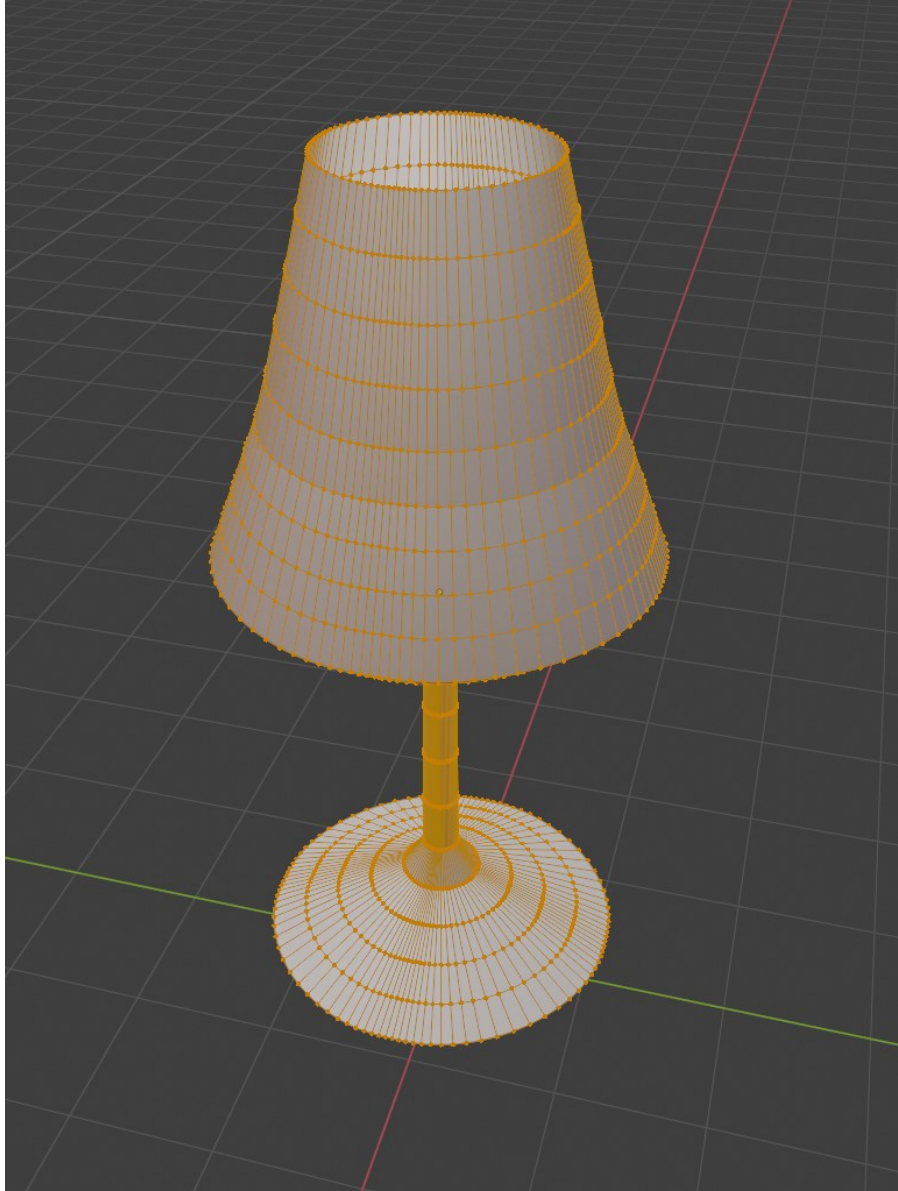
- Triangle mesh is foundational and most widely used data-structure for representation of a shape in graphics
- Triangle mesh consists of many triangles joined along their edges to form a surface
- Triangle is fundamental and simple primitive:
  - All vertices lie in the same plane – **always coplanar**
  - GPU graphics rendering pipeline is optimized for working with triangles
  - Easy to define ray-triangle intersections needed for ray-tracing-based rendering
  - Easy to subdivide in smaller triangles
  - Texture coordinates are easily interpolated across triangle
- Different shape representations used in modeling and acquisition can be transformed to triangle mesh
- Triangle mesh has nice properties:
  - Uniformity: simple operations
    - Subdivision: single triangle is replaced with several smaller triangles. Used for smoothing
    - Simplification: replacing the mesh with the simpler one which has the similar shape (topological or geometrical). Used for level of detail

Types of polygons: quads

- In case of four vertices, the polygon is called **quad polygon** (shortly quad).

Vertices	2,304 / 2,304
Edges	4,512 / 4,512
Faces	2,208 / 2,208





Quad mesh

- Often used as a modeling primitive
- Complexity:
  - Easy to create a quad where not all vertices lie on a plane
- In graphics pipeline it is always transformed to triangle.
  - Optionally, In ray-tracing-based rendering plane-ray intersection may be defined and then triangle representation is not needed*.

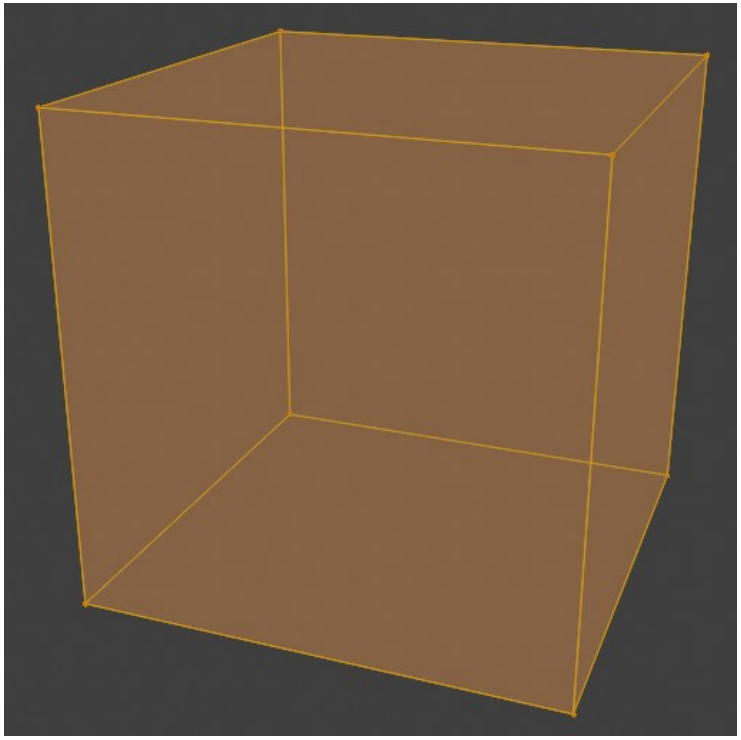
* As we will see, there is always a trade-off between which representation is good for modeling and which representation is good for rendering. Ray-tracing-based rendering can get very flexible with rendering wide representations of shapes but then there is a question if this is feasible to implement and maintain. Often, mapping between different shape representation is researched and used so that on higher level users are provided with intuitive authoring tools and on low level, rendering engine is given efficient representation for rendering process.

Types of polygons: general polygon

- Polygon with more than four vertices is called **general polygon.**
  - Polygons can be convex or concave, and more complex, they may also have holes.
- **Atomic element** – face - of mesh can be any polygon. Common types are: triangle and quad.
- It is a good practice to keep atomic elements as simple as possible (so that computation is easier) and combine those atomic elements into more complex shapes, e.g., convex or concave meshes or meshes with holes.

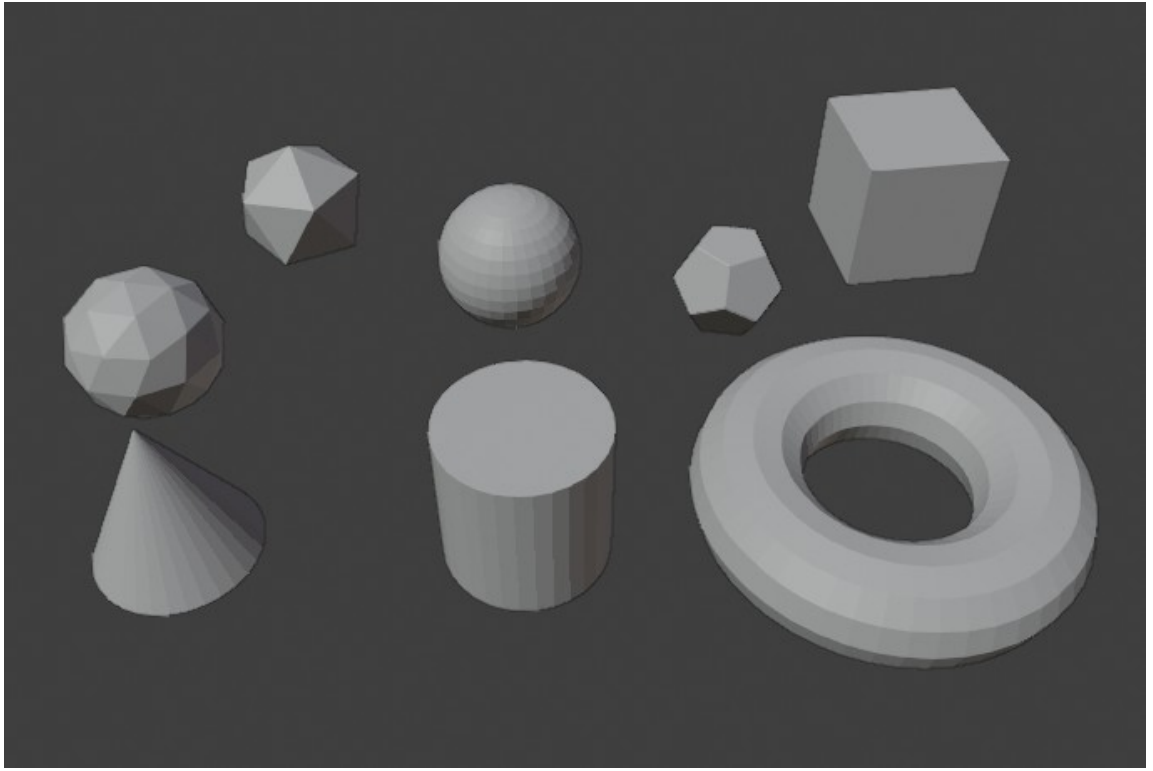
Connectivity

- Complex polygons depend on vertex connectivity information.
- Example: cube
  - Define 8 vertices
  - Define how are those vertices connected to form faces



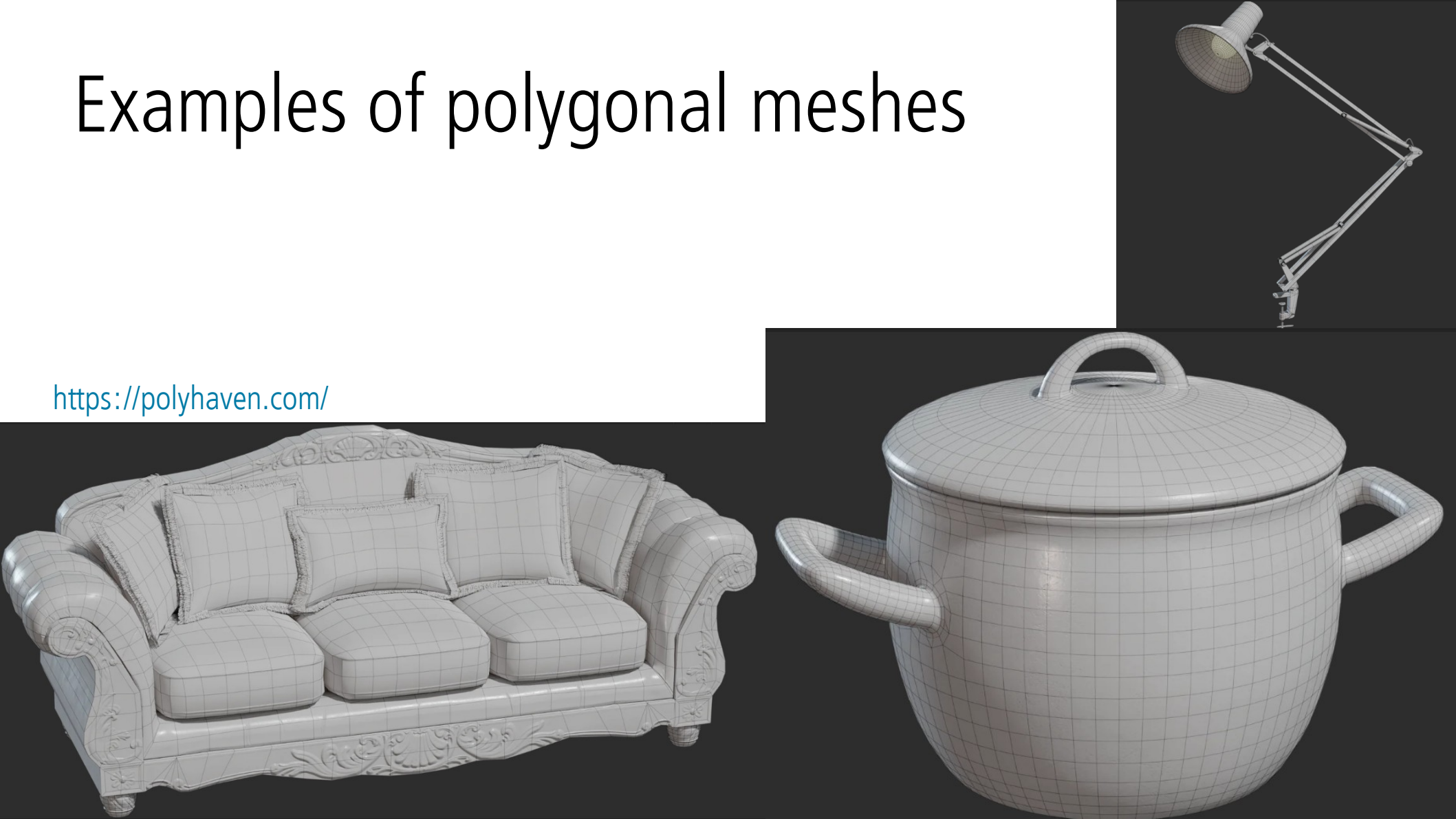
Polygon mesh representation

- Basic information needed for representing and storing polygonal mesh:
  - **Vertex positions** → Geometry
  - **Vertex connectivity** → topology
- 3D polygon mesh: 2D surface embedded in 3D space



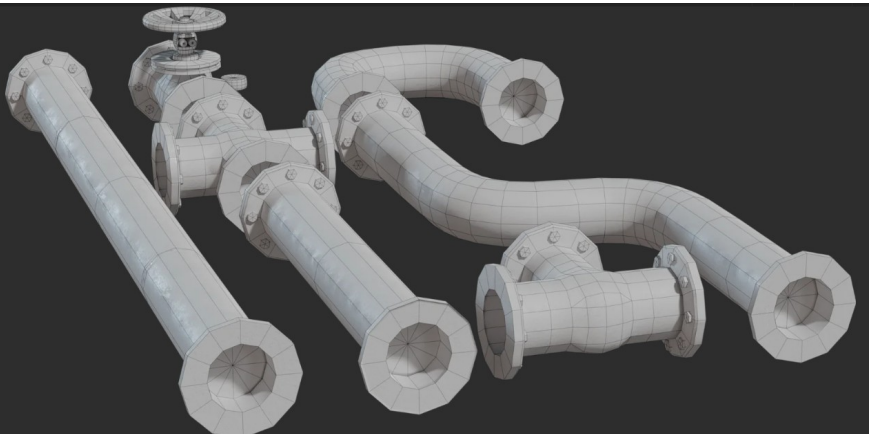
Examples of polygonal meshes

https://polyhaven.com/



Examples of polygonal meshes

https://polyhaven.com/



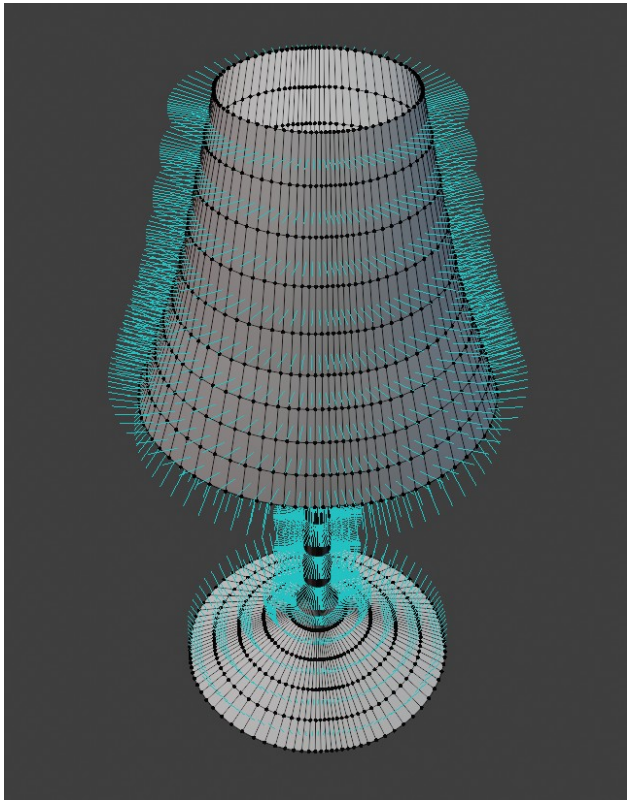
Additional mesh information

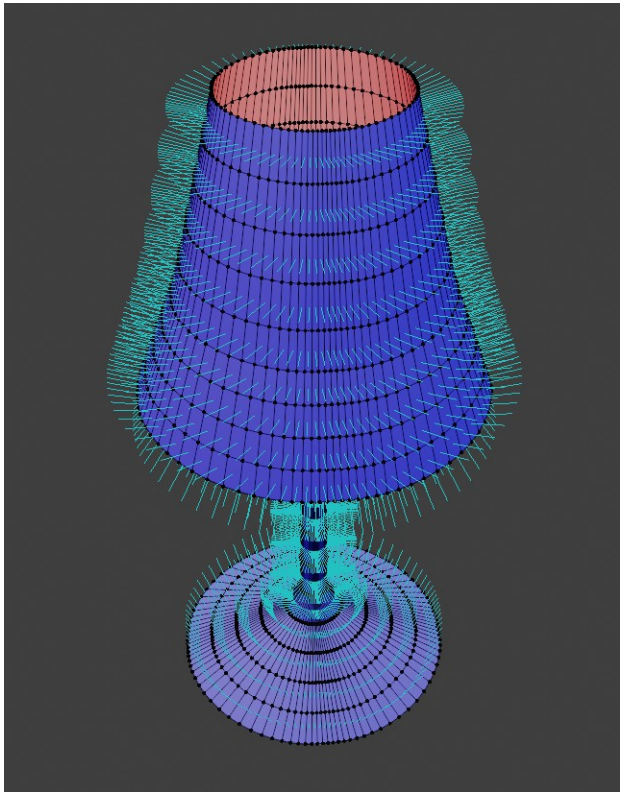
- Besides vertex positions and connectivity information, additional information can be used.
- Those are values that user creates during modeling and which are used for rendering* :
  - Normal*
  - Texture coordinate*
  - Color*
  - Any kind of information that can be encoded and used for rendering: weights, temperature, etc.
- Representation and storage of models is highly developed topic. For now we represented ideas on which any professional solutions build on.

* Those are in general called primitive variables. Primitive is generic term in computer graphics which describes object which is understandable by program.

Additional mesh information: Normals

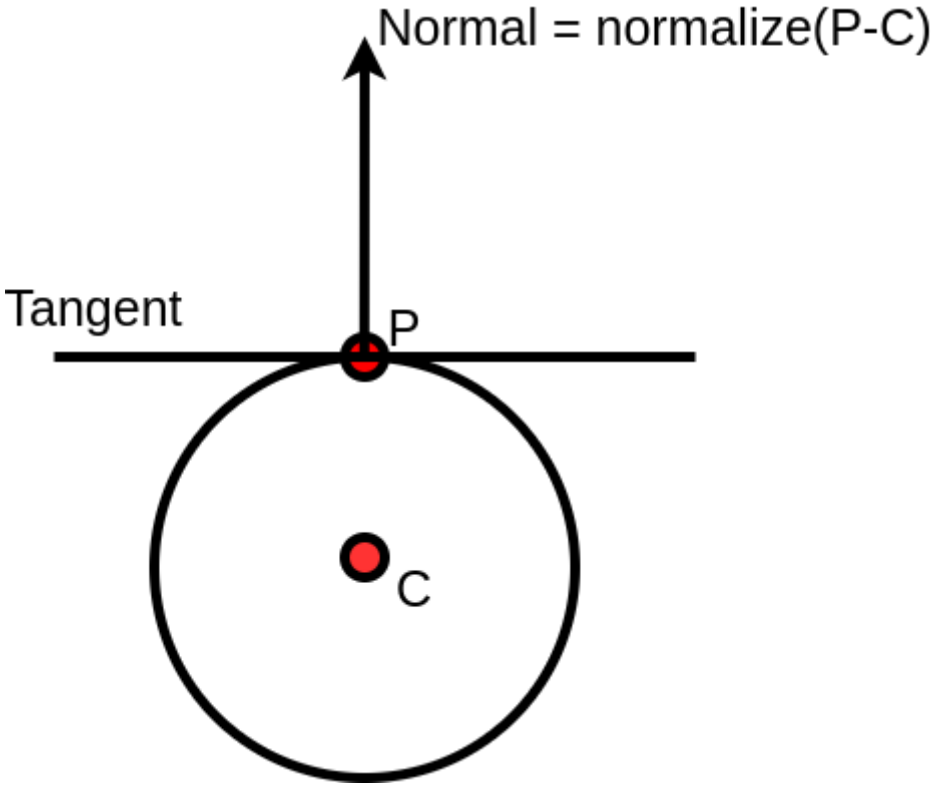
- Orientation of surface in each point is determined by normal vector.
- Normal vector is **core information for rendering (shading) and modeling**.
- Mesh normal can be defined per:
  - face
  - per vertex





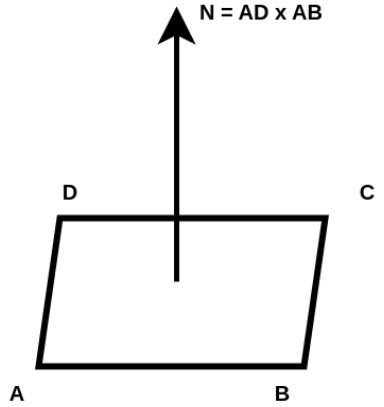
Computing normals

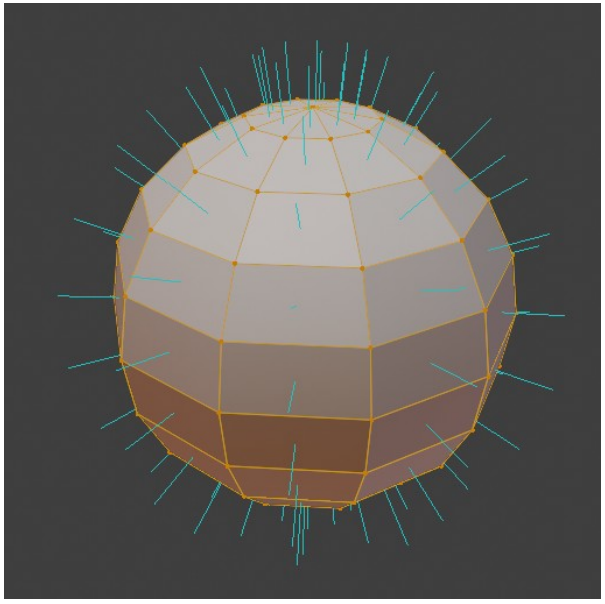
- Normal in surface point is **vector perpendicular to tangent** in that surface point
- Computation of normals depends on shape representation
  - Sphere normal
  - Mesh face normals can be calculated using triangle or quad polygon edges.

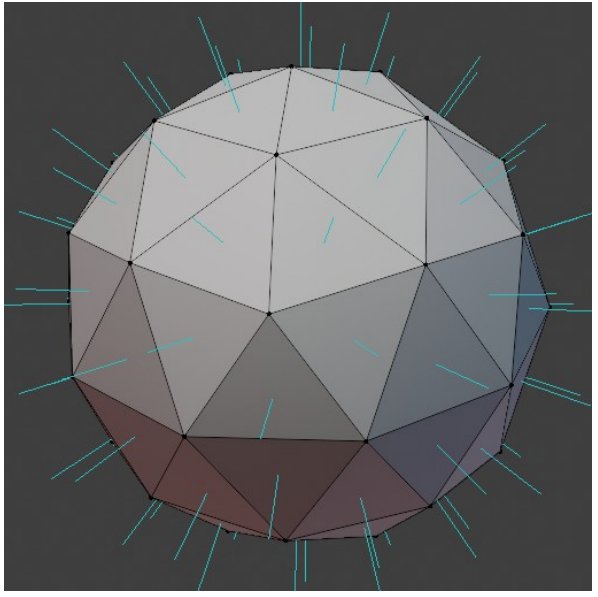


Normals: per face

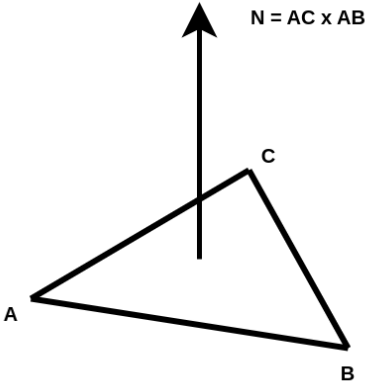
- Cross product between triangle or quad edges
- Winding order of vertices defines orientation of normal
- For right-handed coordinate system, polygons with counter clockwise winding will be front facing





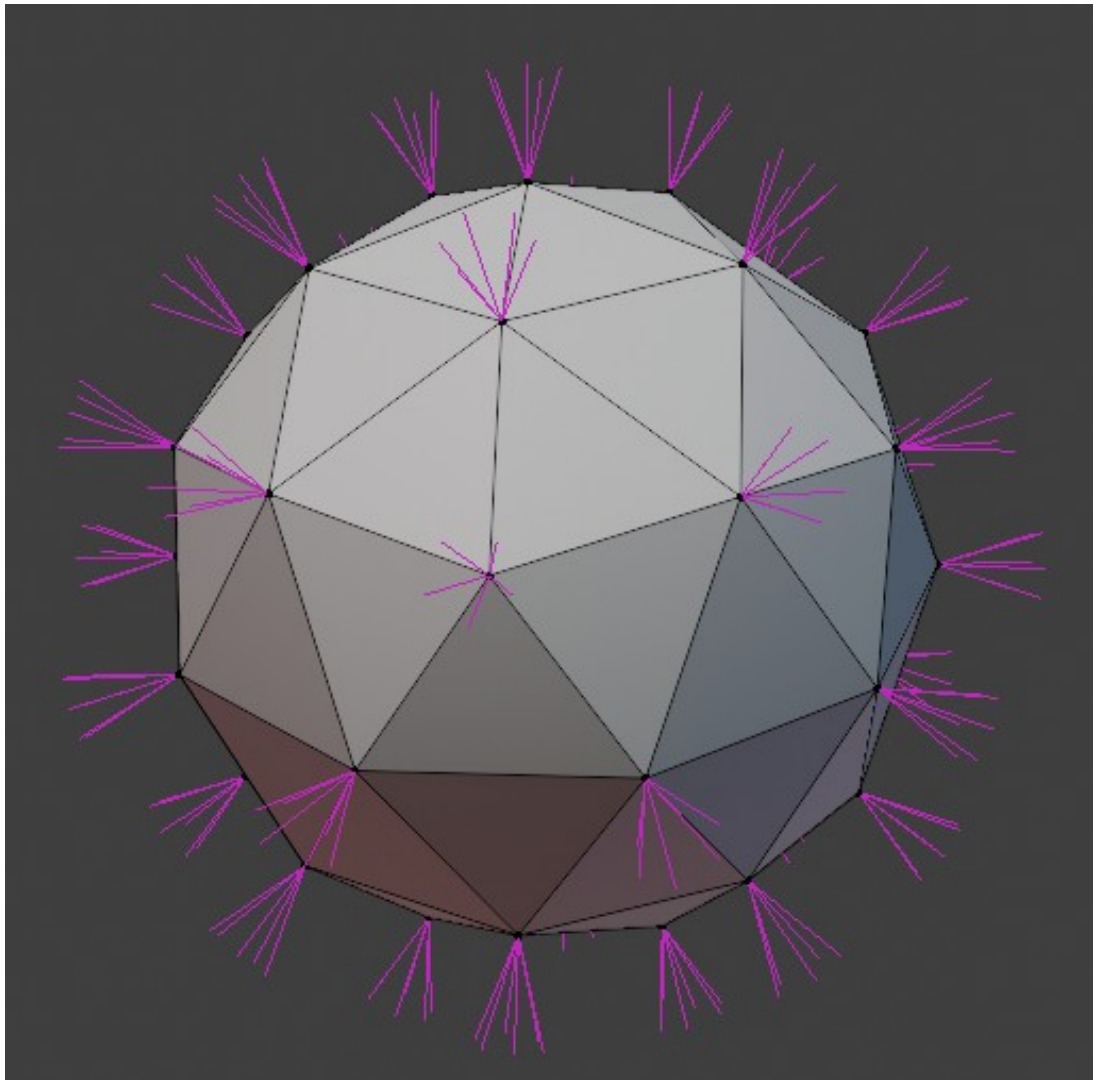


```
N = (B-A).crossProduct(C-A);
```



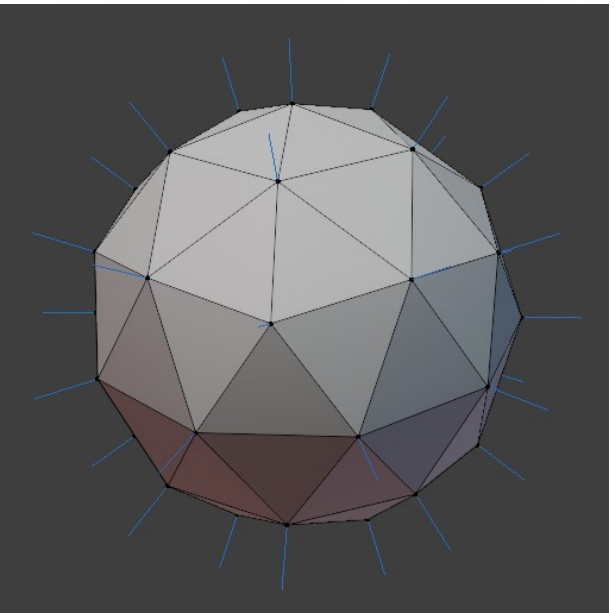
Normals: per vertex

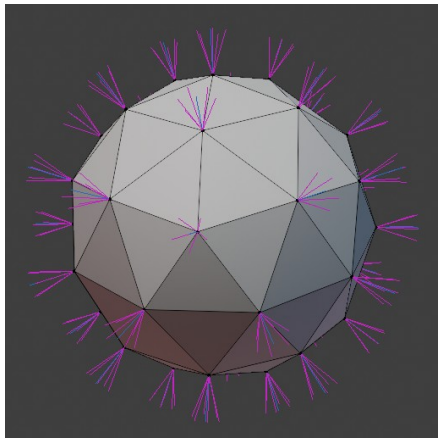
- If normals are defined per vertex, then multiple normals, one for each face, can be stored per vertex.

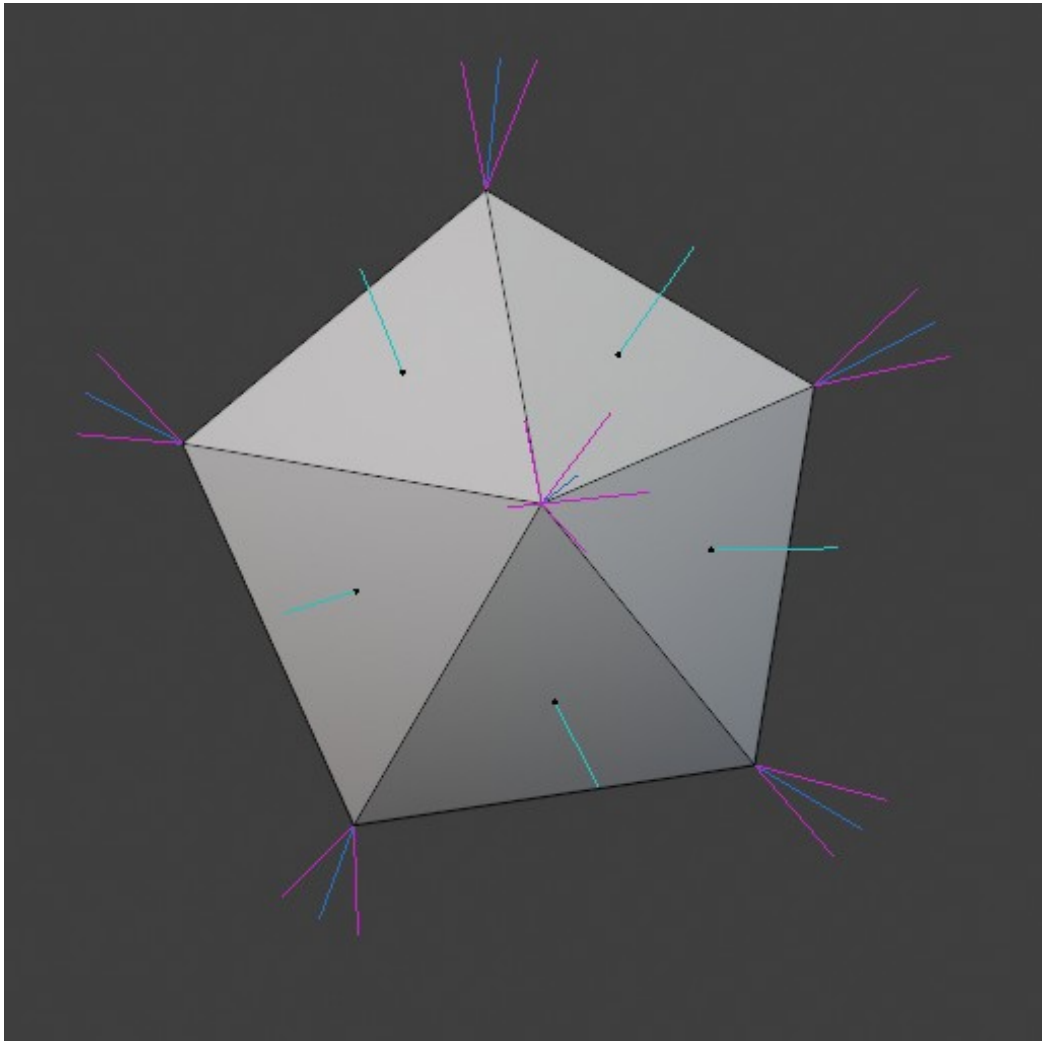


Normals: per vertex

- One normal per vertex is calculated as:
  - For each face:
    - Calculate face normal
    - Add normal to each connected vertex normal
  - For each vertex normal:
    - normalize

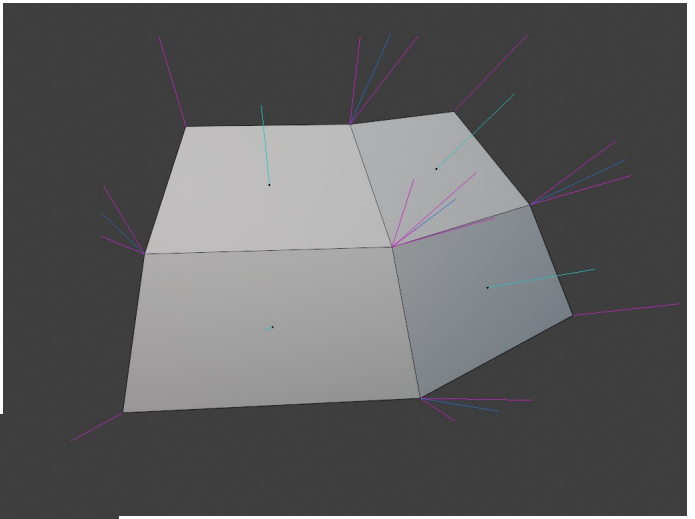


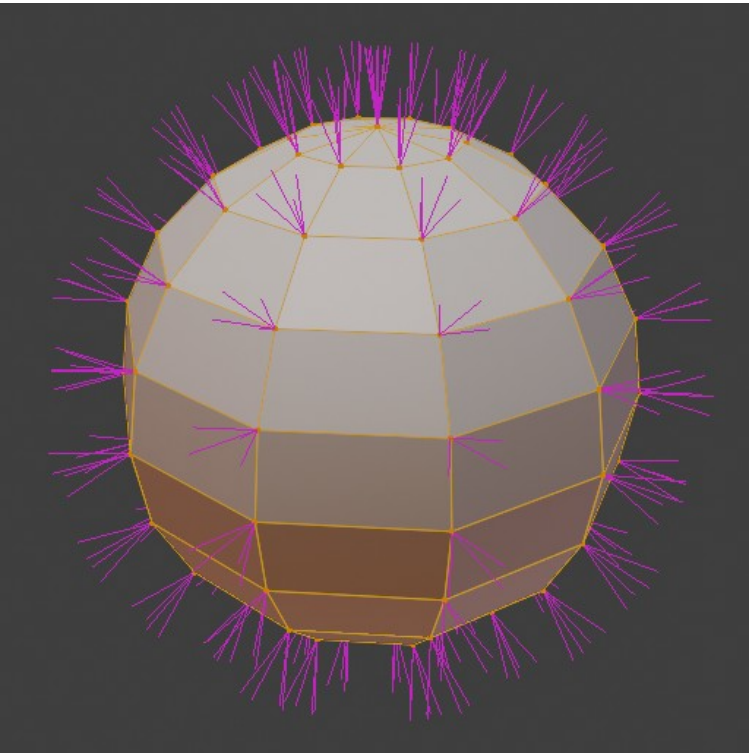


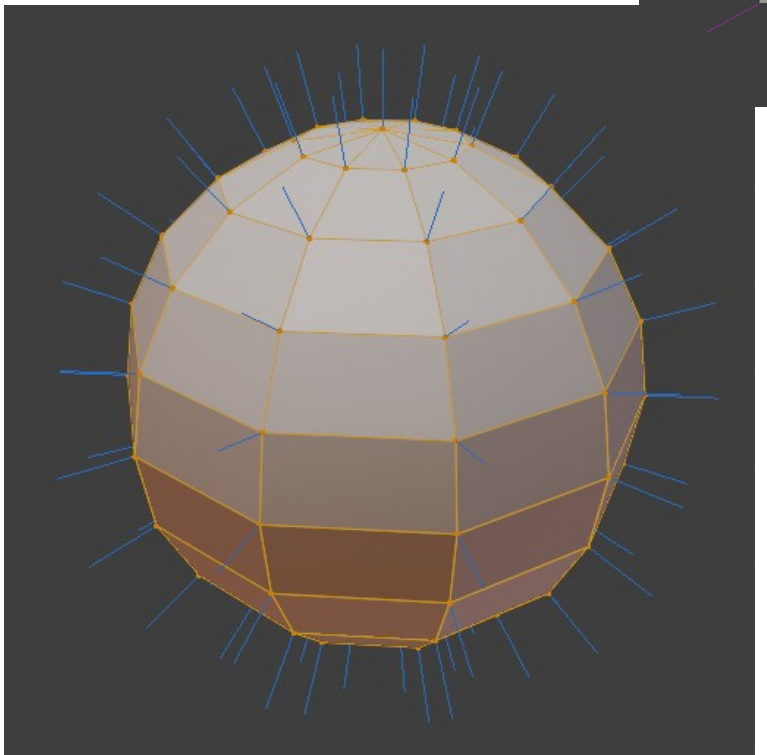


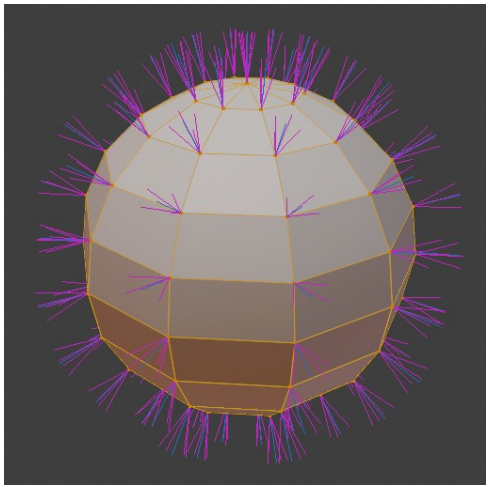
Normals: per vertex

- One normal for each face per vertex
- One normal per vertex



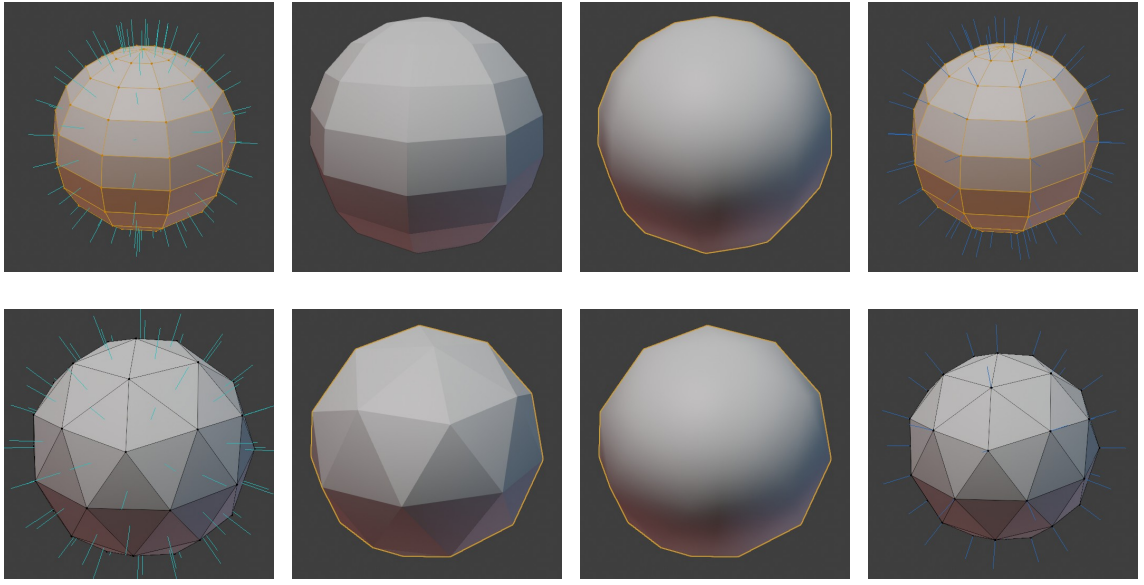






Using normals: shading

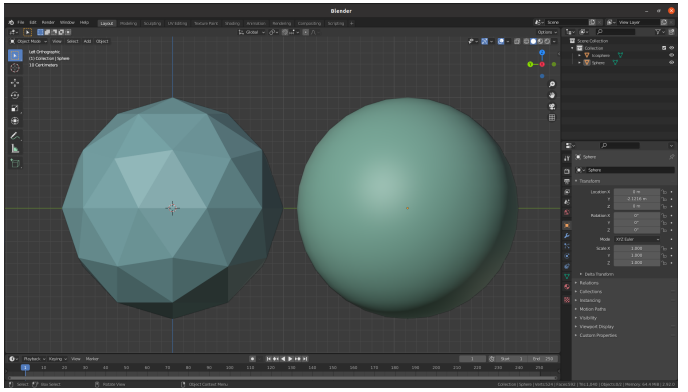
- Appearance of object depends both on shape and material.
  - Normal is main shape information used in shading; appearance calculation
- **Gouraud shading**: normals defined per vertex are linearly interpolated over triangles (using **barycentric coordinates**).
  - Geometry is not changed: note faceted silhouette
  - Illusion of smooth surface is created during shading

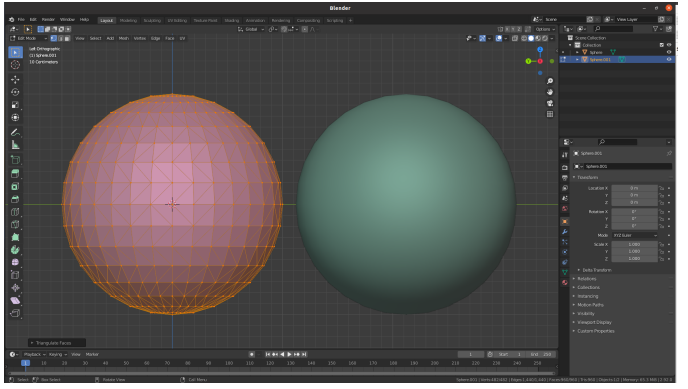


Smooth shading: example

Note: approximating curved surfaces with polygons

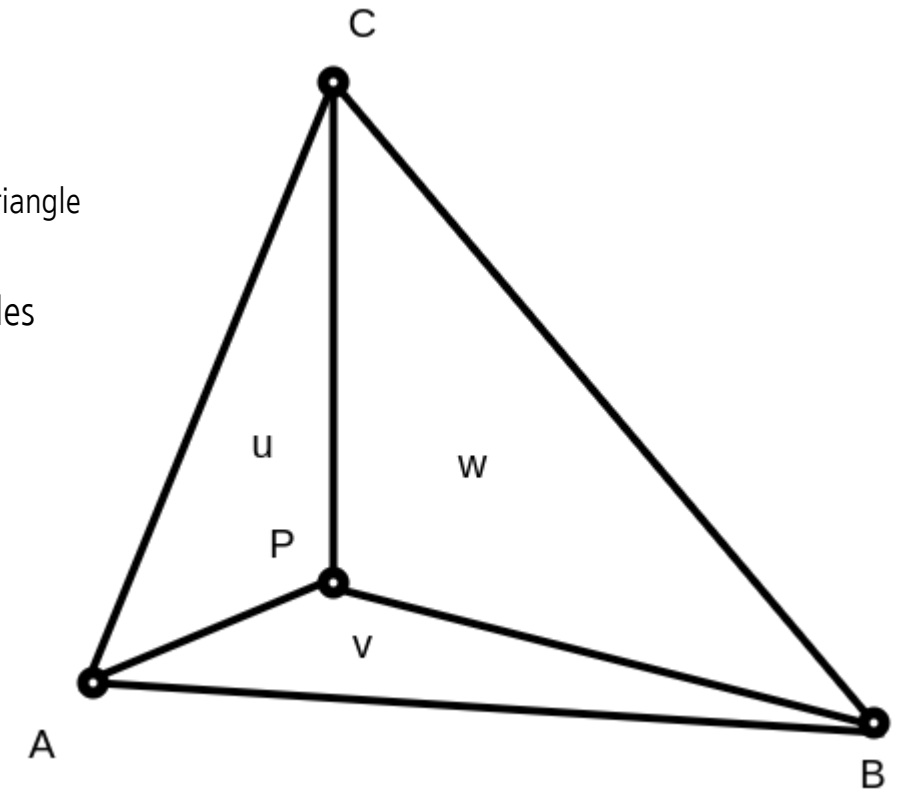
- Example: sphere vs icosahedron
  - Each point on icosahedron is close to point of sphere
  - Each normal vector of icosahedron is close to vector normal of the sphere in the same point. But, function that assigns normals to the sphere is continuous while for icosahedron is piecewise constant → this influences reflection of light!





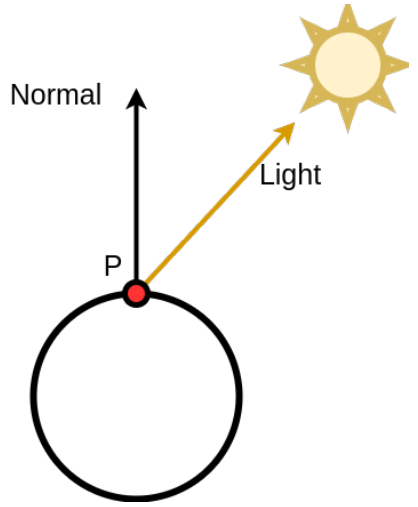
Triangle mesh: barycentric coordinates

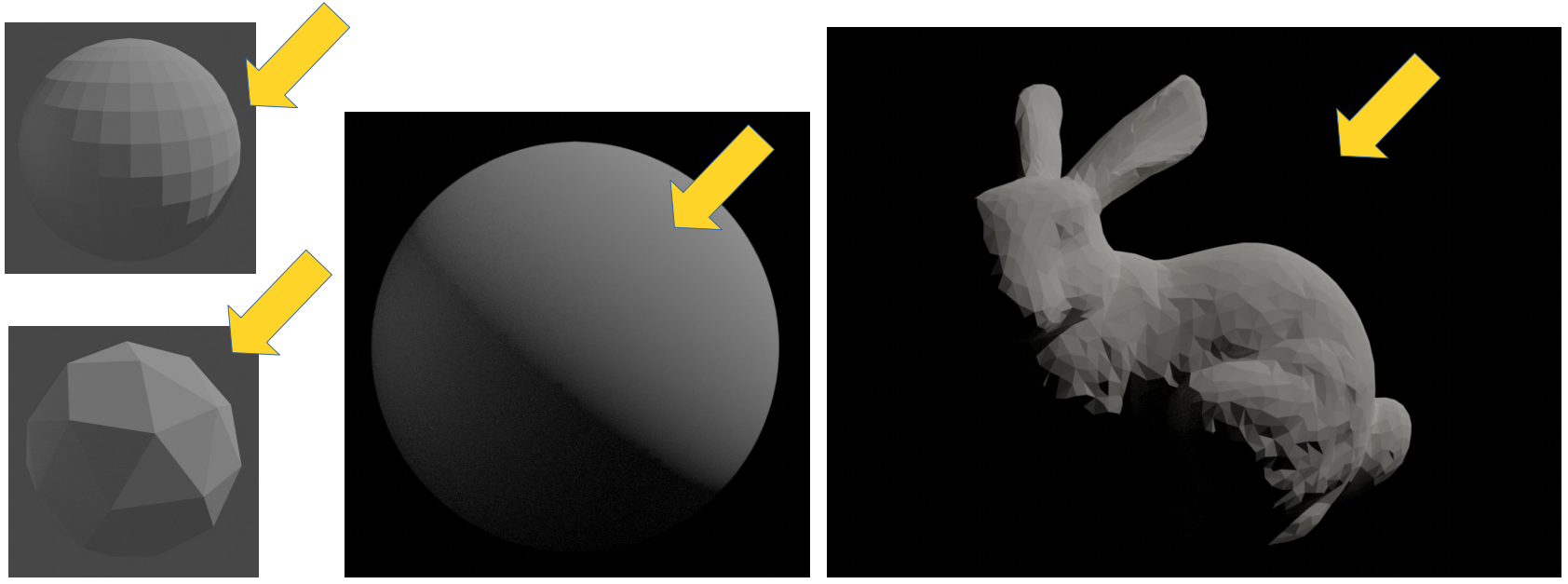
- Express position of any point located on triangle with three scalars.
  - P = uA + vB + wC, where A,B,C are triangle vertices
  - u,v,w, are barycentric coordinates, where u + v + w = 1 and
  - 0 <= u,v,w <=1, otherwise P is outside of triangle. If equal then P is on triangle edge.
- aka areal coordinates: u,v,w are proportional to area of sub-triangles defined by P
  - u = Area(Triangle(CAP)) / Area(Triangle(ABC))
  - v = Area(Triangle(ABP)) / Area(Triangle(ABC))
  - w = Area(Triangle(BCP)) / Area(Triangle(ABC))
  - Area(Triangle(ABC)) = ||(B-A)x(C-A)||/2
- Barycentric coordinates are very useful for interpolating vertex data across triangle surface, e.g., normals, colors, etc.
  - We know P and A,B,C. Thus, calculate, u,v and w and use this factors for interpolating data



Using normals: shading

- Amount of light incident on surface depends on cosine of the angle between light direction and surface normal
  - Lambert's Cosine law
  - cos(θ) = dot(N, L)



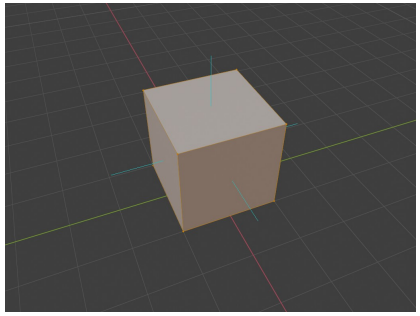
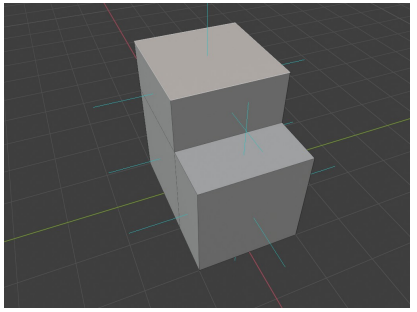
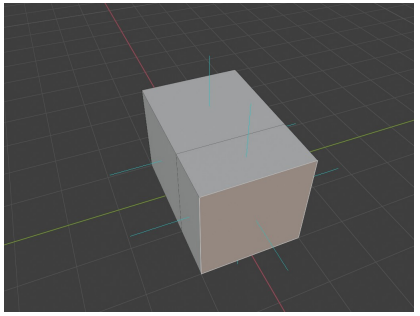
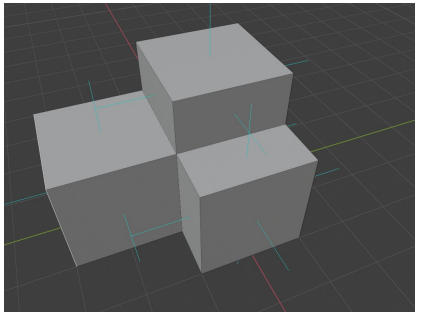
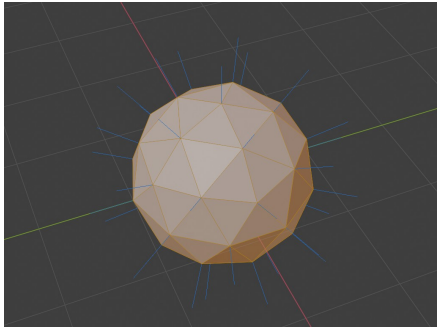
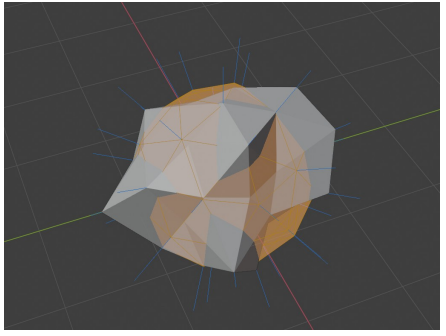
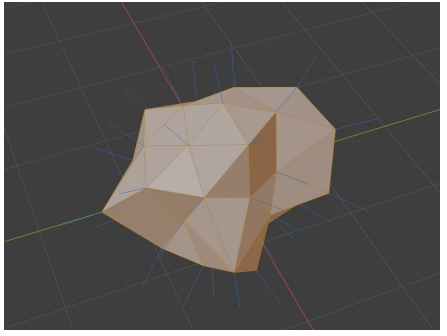


Using normals: shading

- Generally, normal vector is perpendicular to surface, but it can be perturbed so it is not perpendicular.
  - This is trick which is used for shading TODO

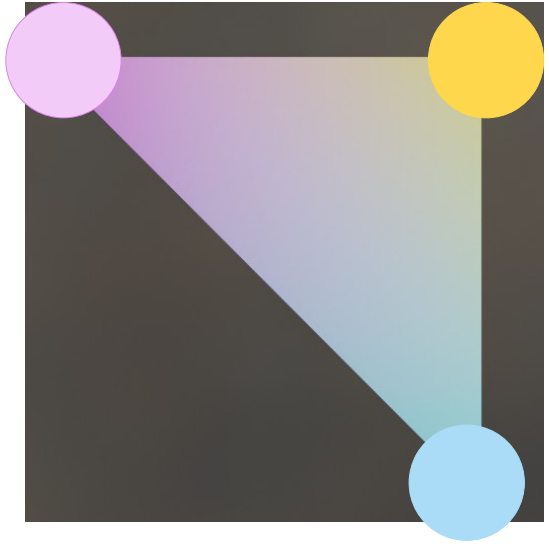
Using normals: geometric manipulation

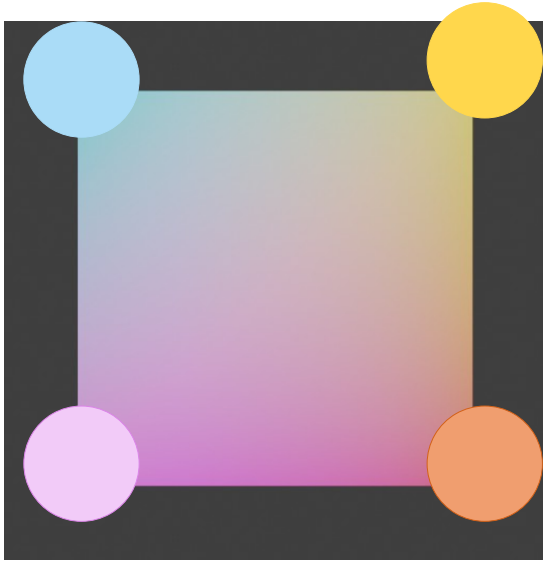
- One of the basic modeling operations is **extrusion**
  - This operation uses normal information for moving the face
- Proceudural modeling utilizes normal direction for **displacing mesh vertices**
- These are only some examples of normal vector usages. The point is to highlight its importance.

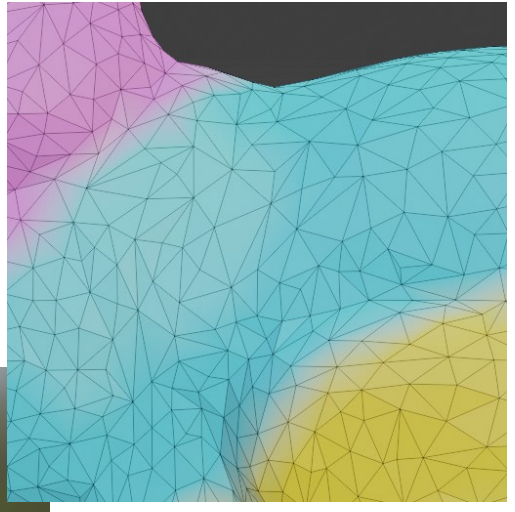








Additional mesh information: vertex colors

- Similarly as normals are defined per vertex and interpolated over triangle for shading purposes, the same can be done with color
- Simplest way to introduce variation over object surface – a form of **texture**
- This method works fine for meshes with finer structure → more colors can be assigned to more vertices.
  - When it is not possible to have fine mesh, then texture is used.







https://docs.blender.org/manual/en/latest/sculpt_paint/vertex_paint/index.html

Additional mesh information: vertex weights

- TODO

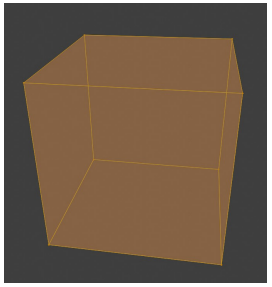
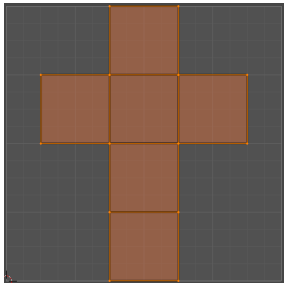
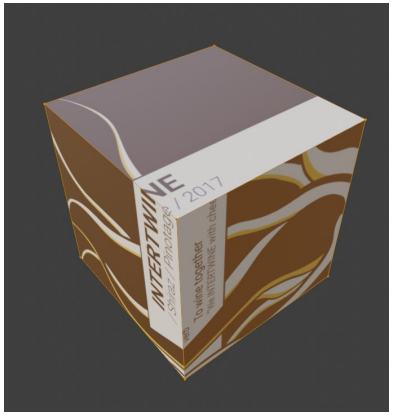
Additional mesh information: texture coordinates

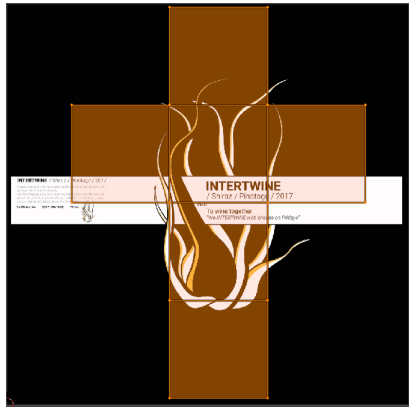
- To add more details on object, often images and procedural patterns are used → **textures**.
- The problem of applying texture image on 3D object is often quite complex than on flat plane.
  - A a way of mapping a 2D image/pattern to 3D shape can be done by "unwrapping" mesh onto 2D plane.

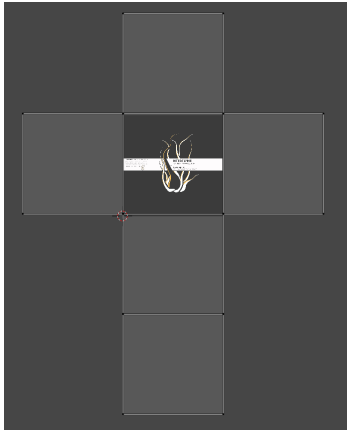
https://polyhaven.com/a/wine_bottles_01

Mesh unwrapping: intuition

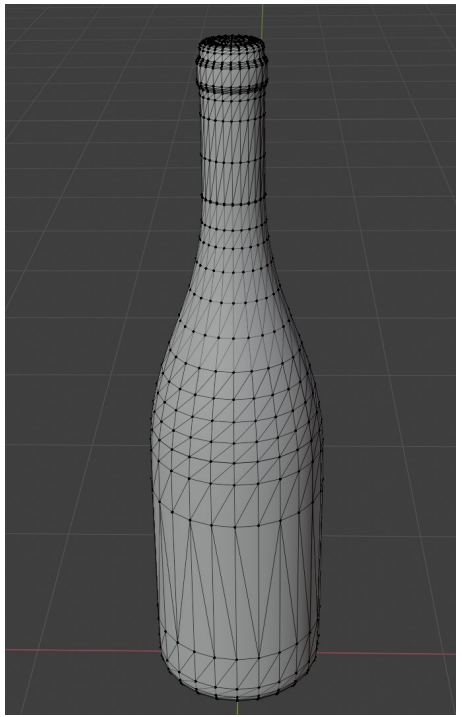
- Mesh is unwrapped into 2D plane we can imagine it lies in 2D coordinate system.
- Vertices of unwrapped mesh now have coordinates in this 2D coordinate system (u,v) → **texture coordinates***
  - (u,v) are in [0,1] range
  - Unwrapped faces can be manipulated, but then texture might get deformed
  - Whole unwrapped mesh can be translated or rotated or scaled to achieve different texture positioning
  - Several faces can overlap

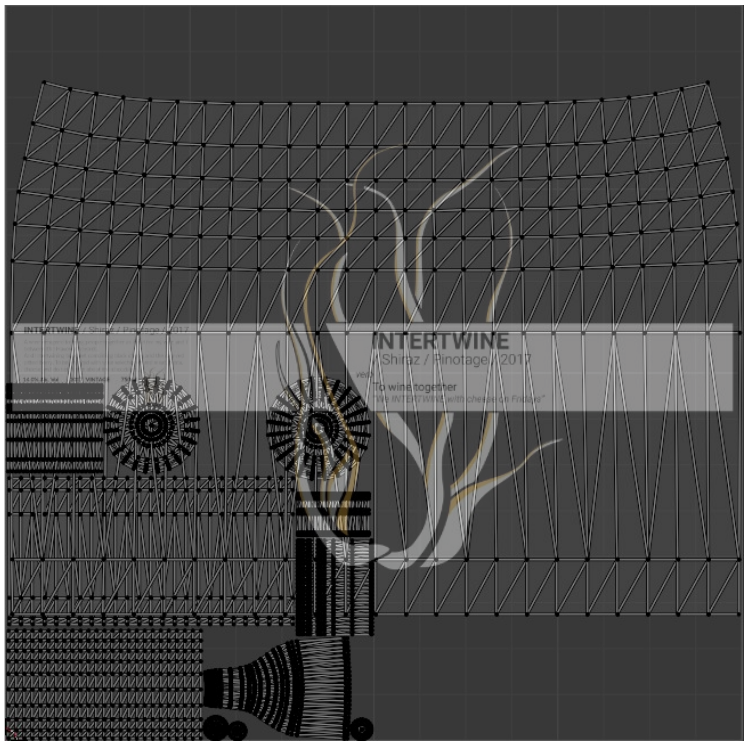








Texture coordinates: mesh unwrap





* Note: this is one way of calculating texture coordinates. Texture coordinates can also be calculated on the fly during rendering or other methods that we will discuss later.

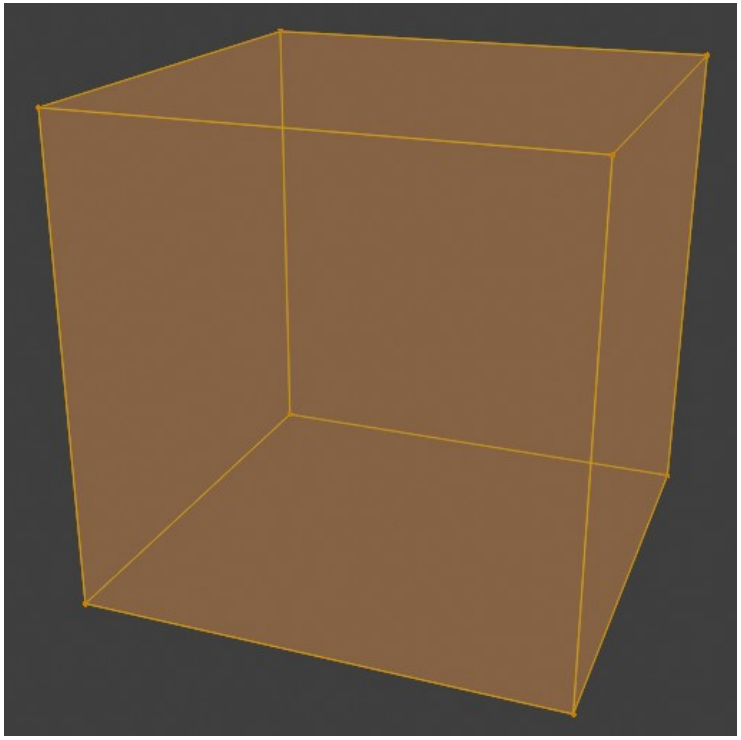
Mesh polygons: storing

Important properties of mesh representation

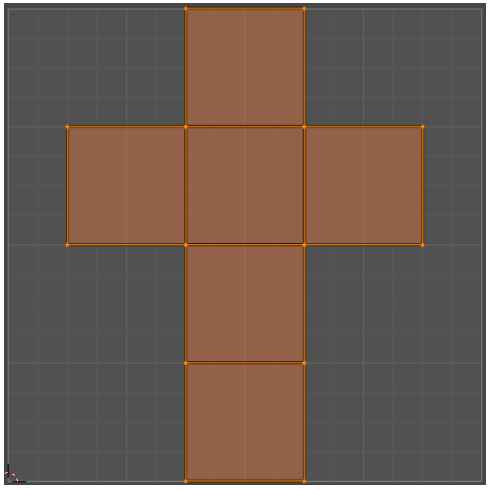
- Efficient topology traversal
- Efficient use of memory
- Efficient updates

Example: representing a cube in a computer

- 8 vertices: **vertex array – geometry information**
  - vert_array = {{-1,1,1},{1,1,1},{1,1,-1},{-1,1,-1},{-1,-1,1},{1,-1,1},{1,-1,-1},{-1,-1,-1}}
- 6 quads for representing faces. Each quad requires 4 vertices: **face index array – topology information**.
  - face_index_array = {4, 4, 4, 4, 4, 4}
  - size(face_index_array) = number of polygons used for representing a shape
- For each face, which indices of vertex array are used: **vertex index array – topology information**.
  - vertex_index_array = {0, 1, 2, 3, 0, 4, 5, 1, 1, 5, 6, 2, 0, 3, 7, 4, 5, 4, 7, 6, 2, 6, 7, 3}
  - size(vertex_index_array) = sum of all values in face index array
  - Note: each face of the cube shares some vertices with other faces



Texture coordinates



- Representing texture coordinates:
  - Texture coordinate per vertex: list of (u,v) coordinates equal to the number of vertices. Example: **texture_coordinates** = {{0,0.5}, {1,0.5}, {0.5,1}, {0,0.5}, {0.5,0}, {1,0.5}}
  - In UV space it is possible that multiple vertices have same texture coordinates thus connectivity information can be used to reduce number of texture coordinates to be written down

Recap: representing and storing mesh

- To describe a mesh we need:
  - Vertex array (vertex positions)
  - Face index array (how many vertices each face is made of)
  - Vertex index array (connectivity)
  - Primitive variables:
    - Vertex color
    - Normals
    - Texture coordinates
    - Other optional application-specific values

Storing polygons: practical note

- Even single polygon mesh in a 3D scene can can be quite large (10^5-10^6 vertices is not unusual)
- Storing vertices and connectivity information must be performed efficiently
  - All vertices must be stored
  - Different techniques exist which try to minimize the amount of data needed for representing connectivity → yielding different standards, formats and API specifications for storing and transferring mesh data, e.g., OBJ or FBX*.

* Those are popular and widely used standards. We will discuss them more when we will be talking about triangle meshes. RenderMan, on the other hand, defines API specification for representing mesh data.

Storing and transferring mesh objects

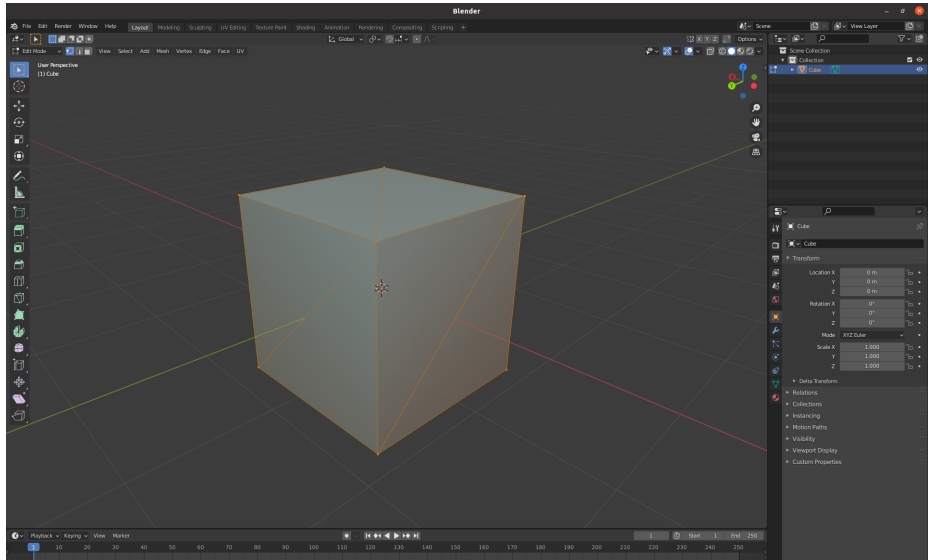
- Mesh data is often transferred between different modeling, rendering and interactive tools.
- Interface between different tools and specification of how mesh should be stored is defined by standards*: https://renderman.pixar.com/resources/RenderMan_20/ribBinding.html
- Different implementations of mesh storage formats exists, which are:
  - Are more or less compact
  - Are more or less human-readable
  - Can contain additional object data which is described with the mesh (textures, materials, etc.)
  - Can contain various metadata (e.g., physical behavior of object described with mesh)
  - Store only mesh information
  - Store whole scene and mesh is only one of elements
- Popular formats:
  - https://all3dp.com/2/most-common-3d-file-formats-model/
  - https://www.sidefx.com/docs/houdini/io/formats/geometry_formats.html
- 3D scene is not necessary created, rendered and used in same software. Usually, whole pipeline of software is used, at least:
  - DCC → game engines
- Formats: OBJ, GLTF, USD
- Interesting: https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-polygon-mesh/polygon-mesh-file-formats

* note that tendency is towards standardization of whole scene description. Which, besides mesh polygons, include materials, lights, cameras, different shape representations, etc.

Example: OBJ file format

OBJ file format. Each line starts with letter representing type of data:

- v – vertices
- vt – texture coordinates
- vn – normals
- F – faces (index of vertex, vertex texture coordinate, vertex normal




```
# Blender v2.92.0 OBJ File: ''
# www.blender.org
o Cube_Cube.002
v -1.000000 -1.000000 1.000000
v -1.000000 1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
vt 0.625000 0.000000
vt 0.375000 0.250000
vt 0.375000 0.000000
vt 0.625000 0.250000
vt 0.375000 0.500000
vt 0.625000 0.500000
vt 0.375000 0.750000
vt 0.625000 0.750000
vt 0.375000 1.000000
vt 0.125000 0.750000
vt 0.125000 0.500000
vt 0.875000 0.500000
vt 0.625000 1.000000
vt 0.875000 0.750000
vn -1.0000 0.0000 0.0000
vn 0.0000 0.0000 -1.0000
vn 1.0000 0.0000 0.0000
vn 0.0000 0.0000 1.0000
vn 0.0000 -1.0000 0.0000
vn 0.0000 1.0000 0.0000
s off
f 2/1/1 3/2/1 1/3/1
f 4/4/2 7/5/2 3/2/2
f 8/6/3 5/7/3 7/5/3
f 6/8/4 1/9/4 5/7/4
f 7/5/5 1/10/5 3/11/5
f 4/12/6 6/8/6 8/6/6
f 2/1/1 4/4/1 3/2/1
f 4/4/2 8/6/2 7/5/2
f 8/6/3 6/8/3 5/7/3
f 6/8/4 2/13/4 1/9/4
f 7/5/5 5/7/5 1/10/5
f 4/12/6 2/14/6 6/8/6
```

Practical note: using different formats

- Professional file formats and representations are commonly used and are very efficient
- Using modeling/interactive/rendering tools, user is often provided with "importer/exporter" – feature which enables importing and exporting different file formats
  - Example: https://www.sidefx.com/docs/houdini/io/formats/index.html
- The problem comes if one writes its own rendering program and parsing file formats for import can be not that easy. Luckily, different libraries can be used for this purposes:
  - Example: https://github.com/assimp/assimp

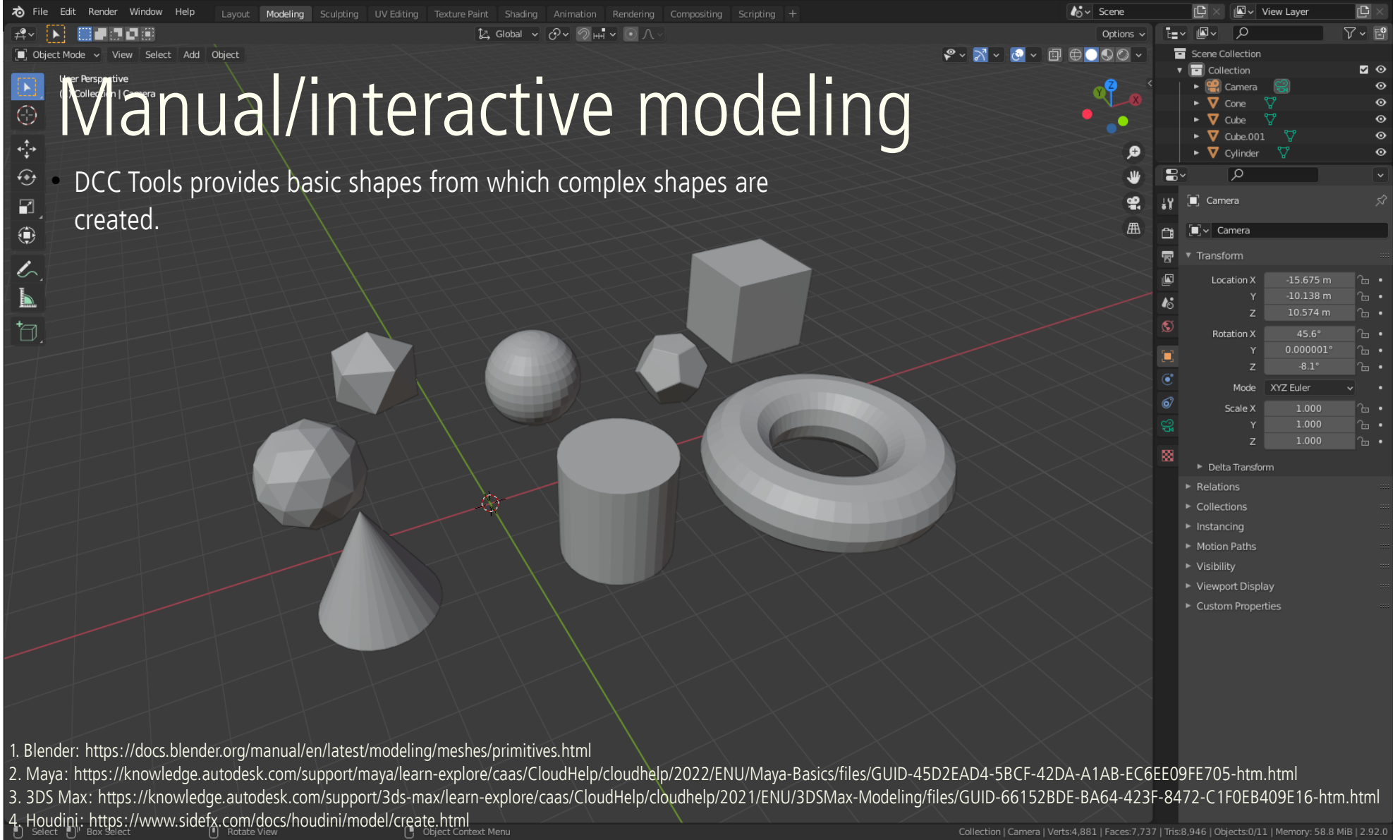
Practical note: units

- Note that until now we haven't discussed units in which shape is stored.
- Units are important for preserving consistent scale among different modeling/rendering tools in which object might be transferred.
- Let's discuss creation of two meshes in Blender. Mesh, that is, vertices are defined in a coordinate system. In this coordinate system we can have two same objects where one is larger and another is smaller. By relation/proportion we can distinguish the scale and we might not care about units.
  - Since positions of vertices are relative to coordinate system, the axis of coordinate system must specify units.
- But what happens if one of this objects in exported into another tool, for example Unity? Unity might use different coordinate system unit scale and object might be too big or too small in this coordinate system!
  - Example: different scales with same coordinates.
- Unit is defined by user who models the object and it must be taken care of during transfer.

Mesh polygons: acquisition and modeling

Manual/interactive modeling

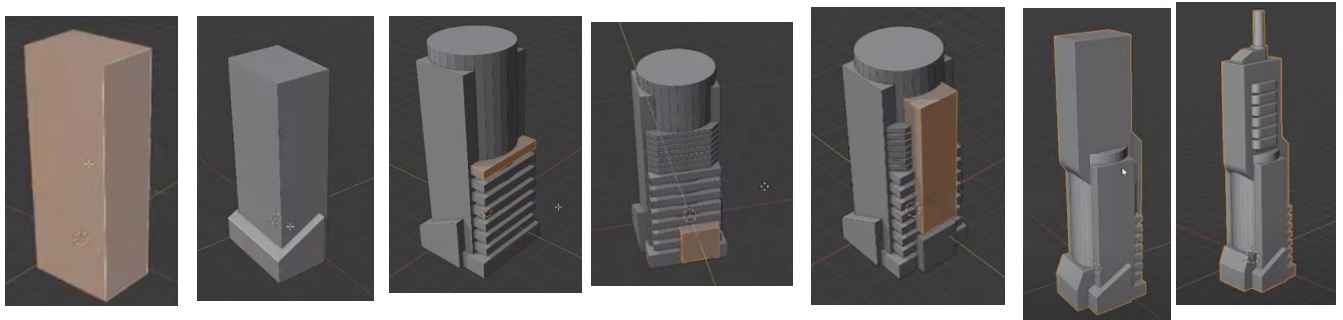
- DCC Tools provides basic shapes from which complex shapes are created.



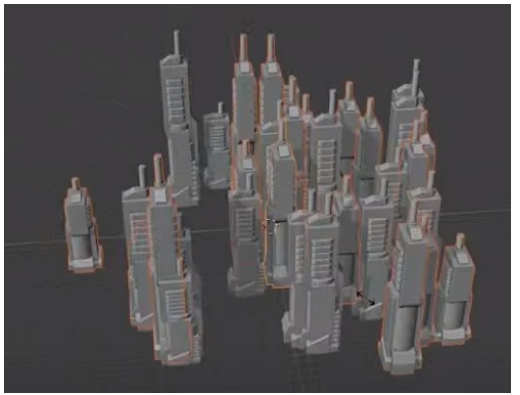
1. Blender: https://docs.blender.org/manual/en/latest/modeling/meshes/primitives.html
2. Maya: https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-Basics/files/GUID-45D2EAD4-5BCF-42DA-A1AB-EC6EE09FE705-htm.html
3. 3DS Max: https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2021/ENU/3DSMax-Modeling/files/GUID-66152BDE-BA64-423F-8472-C1F0EB409E16-htm.html
4. Houdini: https://www.sidefx.com/docs/houdini/model/create.html

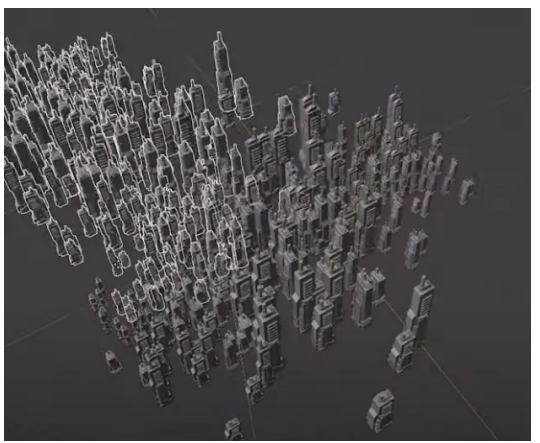
Practical tip: complex shapes from base shapes

- Anything can be decomposed in simple forms[1,2]: box, sphere, cylinder, torus, cones, etc.



https://www.youtube.com/watch?v=Q0qKO2JYR3Y&ab_channel=BlenderSecrets



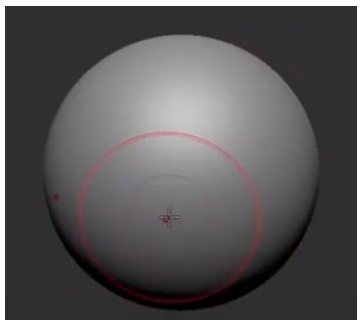
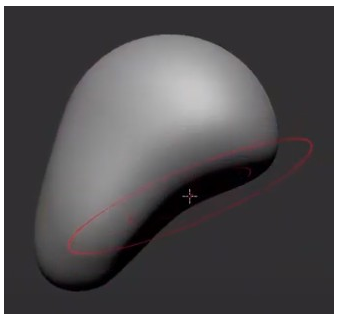
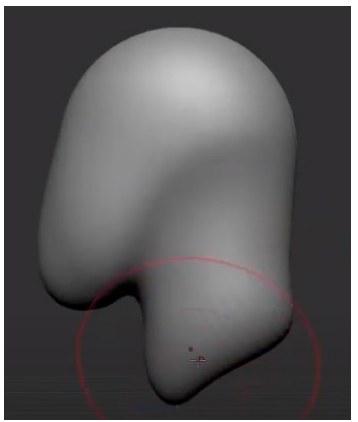


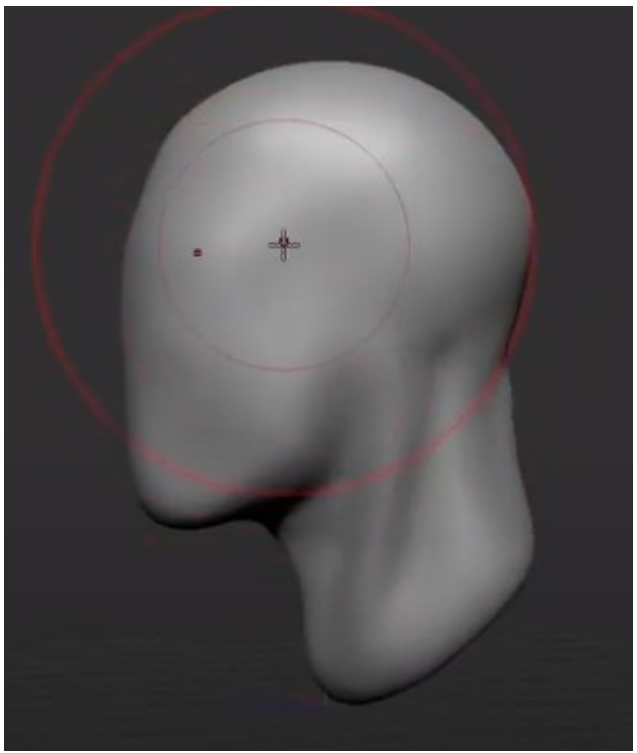
1. http://www.thedrawingwebsite.com/2015/02/18/practicing-your-draw-fu-forms-forms-are-like-sentences/

2. https://www.youtube.com/watch?v=6T_-DiAzYBc&list=RDCMUClM2LuQ1q5WEc23462tQzBg&start_radio=1&rv=6T_-DiAzYBc&t=1343&ab_channel=Proko

Practical tip: complex shapes from base shapes

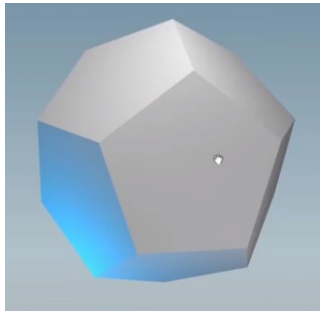
- Sculpting base shapes.

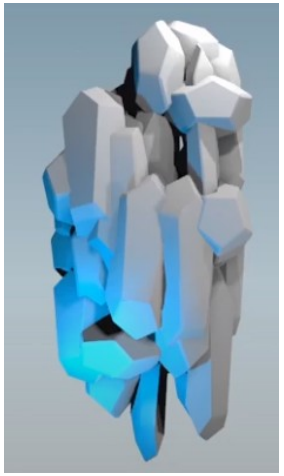




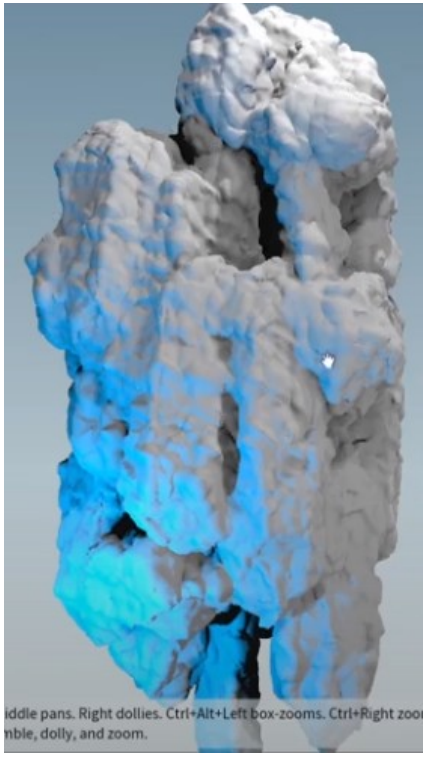


https://www.youtube.com/watch?v=p4Y1hVCMMXw&ab_channel=RealEnay

Practical tip: complex shapes from base shapes





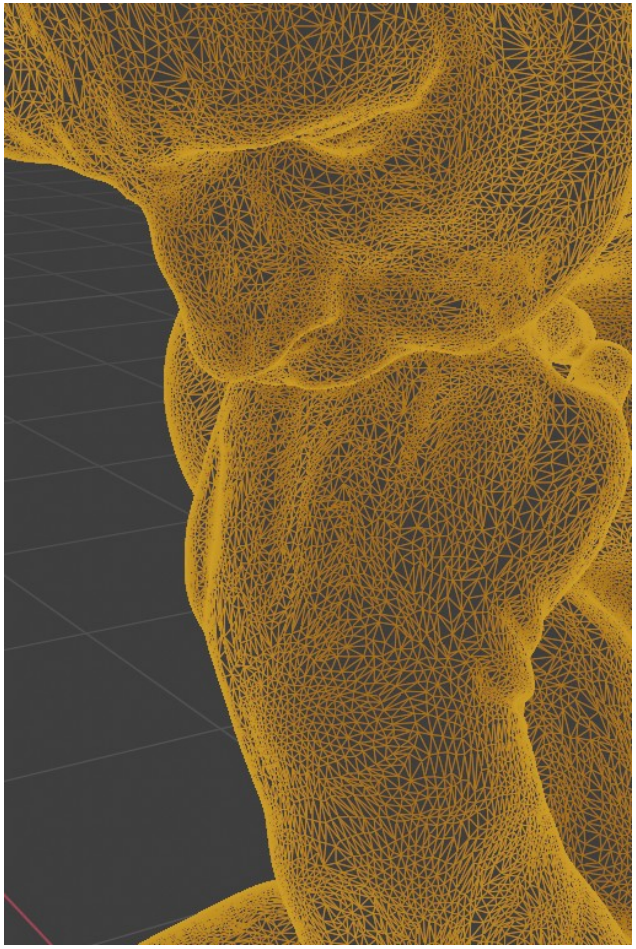


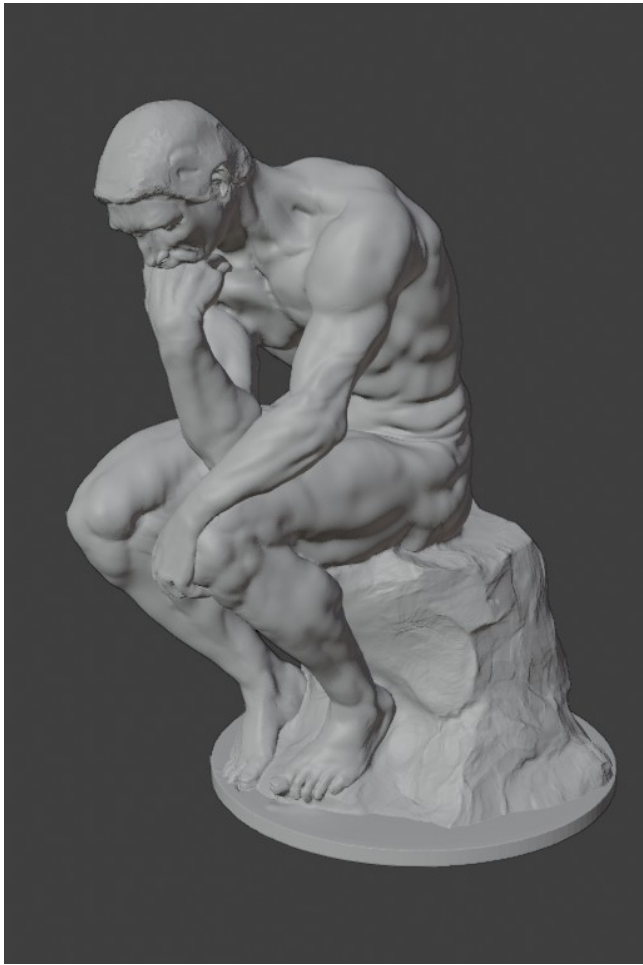
- Procedural modeling

https://www.youtube.com/watch?v=fxOxygaEOFk&ab_channel=SimonHoudini

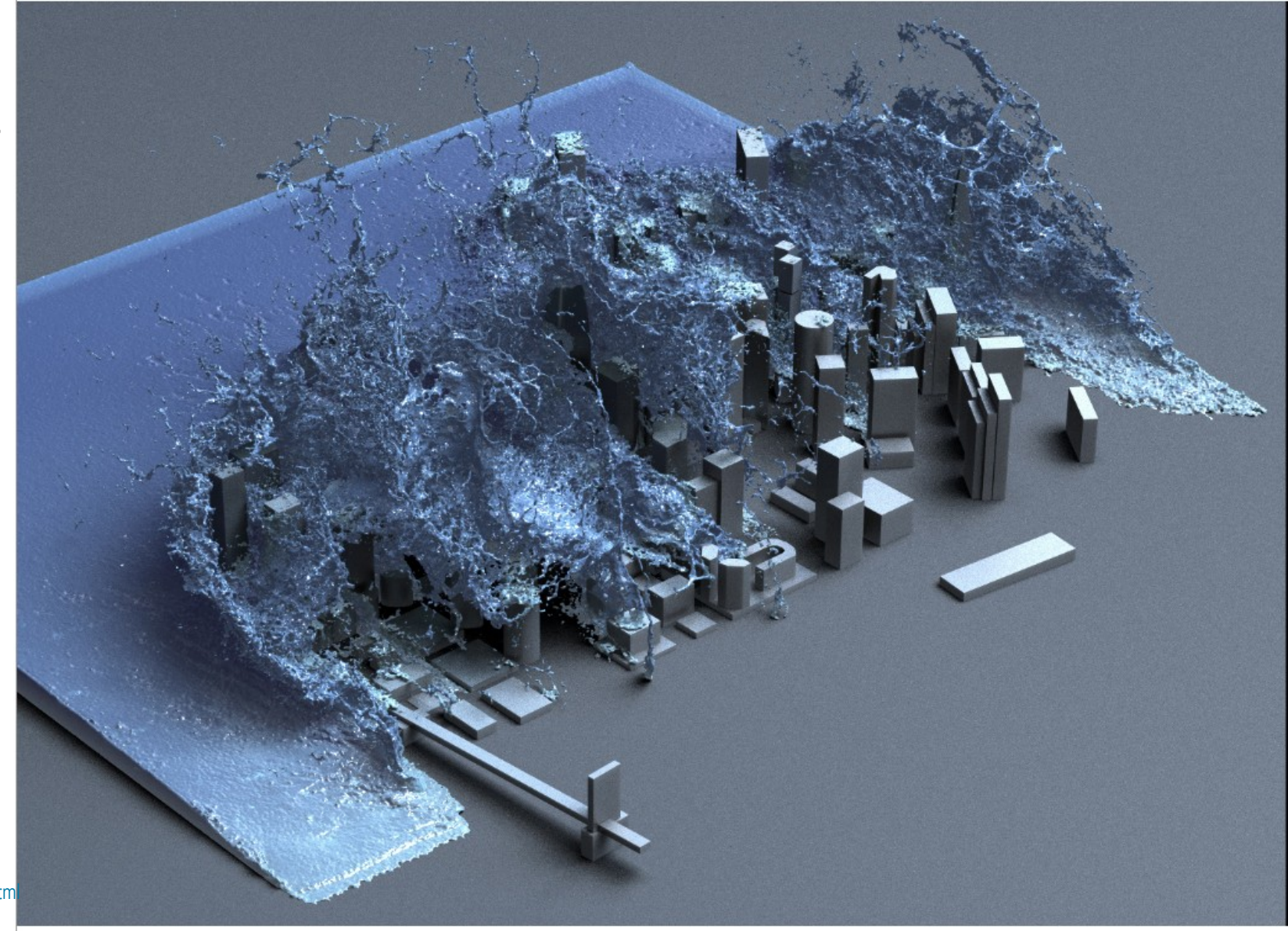
Scanning and photogrammetry

Scan the world: https://www.myminifactory.com/scantheworld/
Sans Factory: https://www.scansfactory.com/
Mega Scans: https://quixel.com/megascans/home/
Art Station 3D scanning and photogrammetry:
https://www.artstation.com/channels/photogrammetry_3d_scanning?sort_by=popular





Simulations



https://people.csail.mit.edu/kuiwu/gvdb_sim.html

Mesh polygons: concluding

Important properties of meshes

- Goal: not to go deep into definitions but rather to verify properties using simpler methods
- **Mesh boundary**: formal sum of vertices
- **Closed mesh**: mesh boundary is zero. Required for defining what is "inside" and "outside" by winding number rule
- **Manifold mesh**: each vertex has arriving and leaving edge
  - Manifolds are desired since it is easy to work with them (both manually and algorithmically)
  - Smooth vs not smooth manifolds (e.g., cube)
  - Self-intersecting meshes are not manifolds
  - In graphics we generally use polyhedral manifolds
- **Oriented vs unoriented meshes**
  - We use oriented meshes so that boundary can be defined

<IMAGES: DEPICT IMPORTANT PROPERTIES!>

Exploring meshes

- **Meshlab** is open-source tool for mesh manipulation and processing
  - Mesh smoothing and sharpening
  - Re-meshing: subdivide, re-sample, simplify,
  - Topological operations: fill holes, fix self-intersections
  - Boolean operations
- **Libigl** library for mesh processing
  - Mesh curvature
- **Blender**
  - Modeling with meshes
  - Procedural meshes with Python
- **Sources of mesh data**:
  - https://casual-effects.com/data/index.html
  - https://polyhaven.com/models
  - https://sketchfab.com/
  - http://graphics.stanford.edu/data/3Dscanrep/
- **More informations:**
  - https://www.realtimerendering.com/#polytech

Deeper into topic

- **CGAL** is advanced geometry processing library
- Polygonal mesh is highly used and researched method in computer graphics. We have covered foundations. There are many other topics. Some of those will be discussed later, some are out of scope for this course:
  - Tessellation and triangulation
  - Consolidation, mesh reparation
  - Representation
  - Simplification, level of detail
  - Compression

Mesh shape representation: verdict

- Pros:
  - Most common surface representation
  - Simple for representing and intuitive for modeling and acquisition
  - A lot of effort has been made to represent various shapes with meshes
  - Lot of research has been done to convert other shape representations to mesh representation
  - Graphics hardware is adapted and optimized to work with (triangle) meshes → fast rendering
- Cons:
  - Not Guaranteeing smoothness
  - Triangle meshes are not efficient or intuitive for manual/interactive modeling
  - Not every object is well suited to mesh representation:
    - Shapes that have geometrical detail at every level (e.g., fractured marble)
    - Some objects have structure which is unsuitable for mesh representation, e.g., hair which has more compact representations