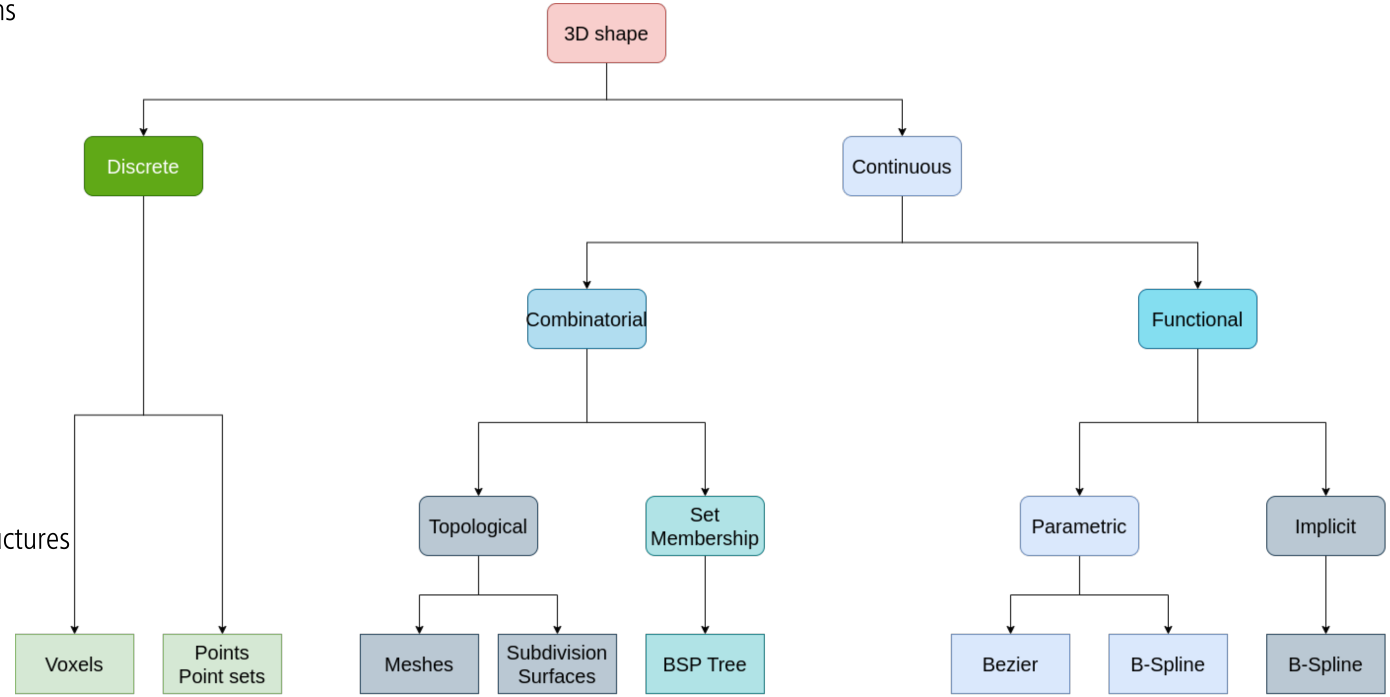
3D Models

Shape representations

Recap

- 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 shape representations found in geometrical modeling are*:

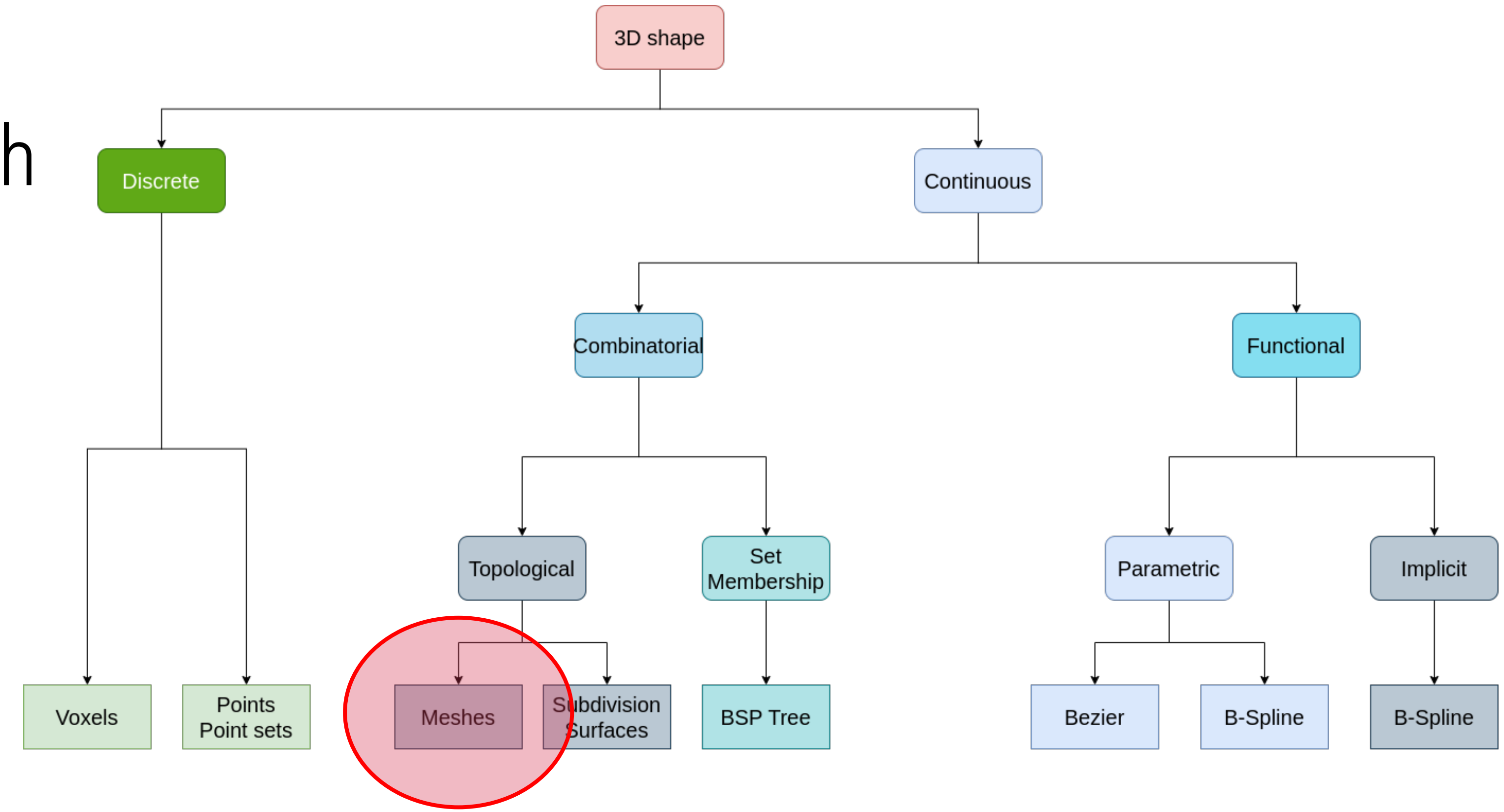
- Polygon meshes
- Parametric surfaces

<IMAGES: mesh vs parametric surface>

  - Mesh is discrete, parametric surface is continuous

*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



Foundations of 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

<IMAGE OF MESH USAGE IN DCC AND GAME ENGINES>

- Blender: https://docs.blender.org/manual/en/latest/modeling/meshes/index.html
- Maya: https://help.autodesk.com/view/MAYAUL/2023/ENU/?guid=GUID-7941F97A-36E8-47FE-95D1-71412A3B3017
- 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 building block: polygon

- Polygon is planar shape which is defined by connecting array of points.

<IMAGE: single polygon>

- 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

- Face can have minimum three vertices
- In case of three vertices the polygon is called **triangle polygon** (shortly triangle).
  - This is very interesting type of polygon in computer graphics and we will return to this one a lot!
- In case of four vertices, the polygon is called **quad polygon** (shortly quad).
  - This is another interesting polygon that we will also shortly cover.
- Polygon with more than four vertices is called **general polygon**.
  - To make calculations easier, we desire vertices making a plane to be in the same plane. This holds true for triangle, but not necessary any other kind of polygon.
  - Polygons can be convex or concave, and more complex, they may also have holes*. We will not focus on those for now.

* Note that we are currently discussing atomic element of a polygonal mesh. 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. Of course, it is generally hard to say what is better – all depend on application and requirements of application. If for some case complex polygons are needed, then using those is also approved. But in this course, we will keep to the simple building blocks and hint more complex element for you to investigate further if needed.

More complex 3D shapes using polygons

- To create more complex 3D shapes we connect faces to each other.
- Simplest example is cube: 6 faces
  - To describe a cube we need: (1) to define 8 vertices and (2) define how are those vertices connected to form faces

<IMAGE: VERTICES, CONNECTIVITY, FACES>

- Description of how vertices are defined is called **connectivity**
- Vertex and connectivity information is basic information needed for describing 3D shapes.

Representing polygons: practical note

- As discussed, we need at least **vertices** (3D coordinate points) and **connectivity** information to represent a **polygonal shape**.
- As 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
  - Therefore, 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.

Mesh: topology and geometry

- Geometry
- Topology: “what is connected to what”
  - Connectivity of polygons

Example: representing a cube in a computer

- Minimal information needed to be stored is 8 vertices. This is stored in **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}}
- Next, we need to decide which polygons will be used for representing faces: **quads** (or triangles)
- We know that 6 quads must be used and that each quad requires 4 vertices. This is stored in **face index array – topology information**. Size of this array defines number of polygons used for representing a shape.
  - face_index_array = {4, 4, 4, 4, 4, 4}
- For each face we need to know which indices of vertex array (thus which vertices) are used: **vertex index array – topology information**. Size of this array is sum of all values in face index array.
  - 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}
  - Note: each face of the cube shares some vertices with other faces

Small example of how modeling is done?

Storing additional information

- Besides storing vertex positions and connectivity information, additional information can be stored.
- 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.
- 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 can be defined per face or per vertex
  - <IMAGE OF SURFACE NORMALS per face and per vertex>
- In case normals are defined per vertex, then multiple normals, one for each face, can be stored per vertex.

Computing normals

- Normal vector is core information that we can obtain from shape representation
  - Normal in surface point is vector perpendicular to tangent in that surface point
  - Intuition: calculate sphere normal TODO
  - Face normals can be calculated using triangle or quad polygon edges. In this case it is important to check winding order of vertices which defines direction of normal EXAMPLE

Using normals: shading

- We mentioned that both shape and material are used for calculating appearance (shading step in rendering).
- Normal defines orientation of surface towards light determining how bright it is and its thus its appearance
  - Light direction and normal direction determine amount of brightness – facing ratio → this is core step in shading process EXAMPLE
- TODO: smooth vs flat 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
  - Note: normals are defined per vertex. Thus, to obtain normal on any point on triangle we use **interpolation**

Using normals: geometric manipulation

- Normal is also used during modeling
- One of the basic modeling operations is **extrusion**
  - This operation uses normal information for moving mesh shape in its direction
  - Example
- Often, proceudural modeling utilizes normal direction for displacing mesh vertices.
  - Example
- 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 information.
- Example: vertex painting (Blender, Unreal)
- This is the simplest way to introduce variation over object surface – a form of **texture**
- This method works fine for meshes with finer structure since more colors can be assigned to more vertices.
  - When it is not possible to have fine mesh, then texture is used.

Additional mesh information: texture coordinates

- To add more details on object, often images and procedural patterns are used <image>
  - Analogy is hanging posters on the wall.
- The problem of applying 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.
  - <example>
- Once mesh is "made 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

    <example>

* note that 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.

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
    - Example: multiple vertex same texture coordinates

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

Storing and transferring mesh objects

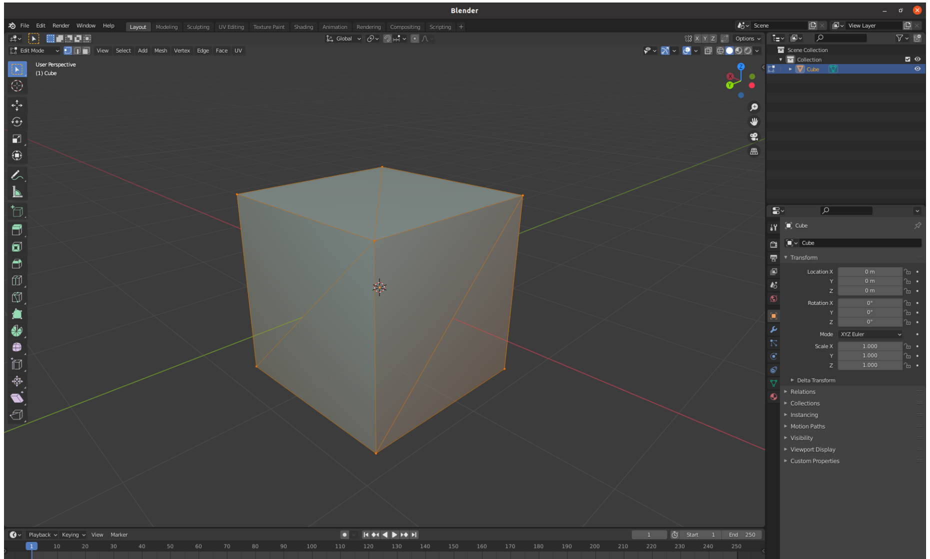
- Data which is modeled is meant to be transferred between different modeling, rendering and interactive tools.
- Interface between different tools and specification of how mesh should be stored is defined by standards*: https://renderman.pixar.com/resources/RenderMan_20/ribBinding.html
- Different implementations 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.

Common types of mesh representations

- Atomic element – face - of mesh can be any polygon. Common types are:
  - Triangle mesh
  - Quad mesh

<IMAGE: triangle vs quad mesh>

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 subdivide in smaller triangles
  - Texture coordinates are easily interpolated across triangle
- 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

Triangle: barycentric interpolation

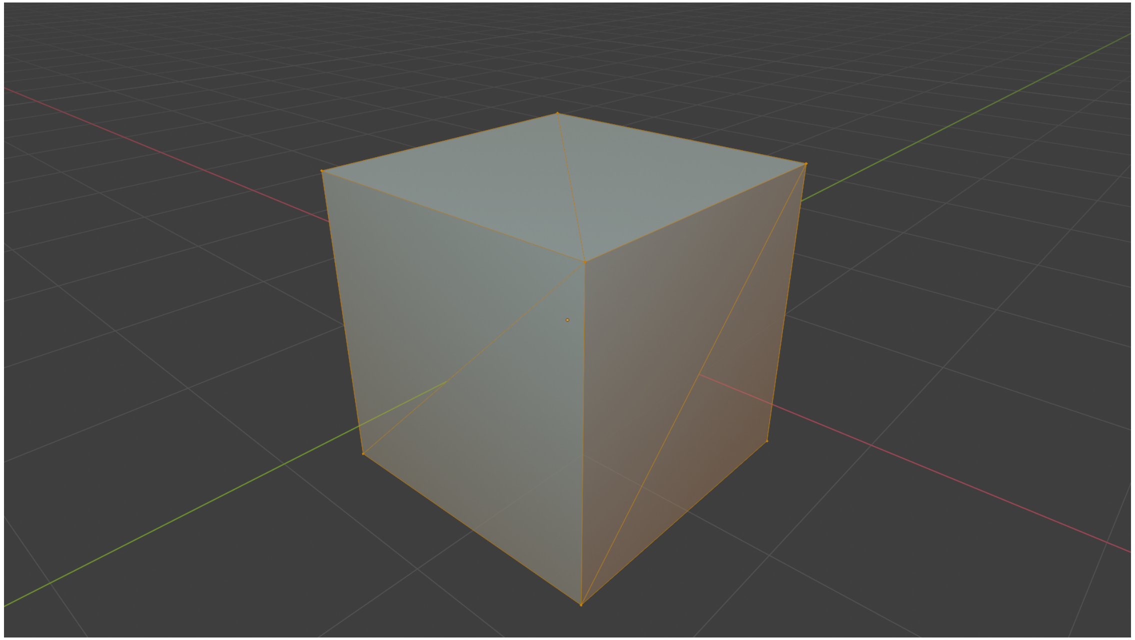
- TODO

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*.

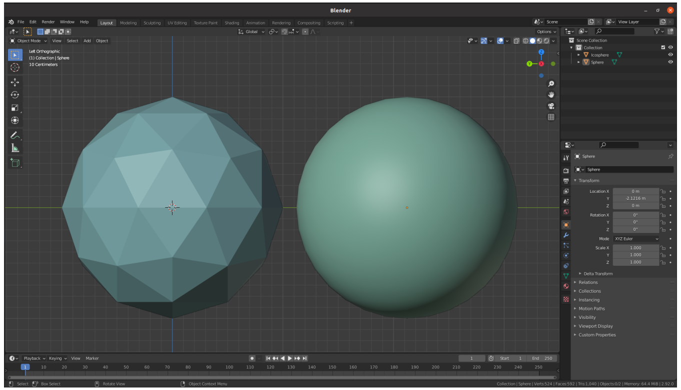
* As we will see, there is always a trade-off between which representation is good for modeling and which representation is good for rendering. Ray-tracing-based rendering can get very flexible with rendering wide representations of shapes but then there is a question if this is feasible to implement 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.

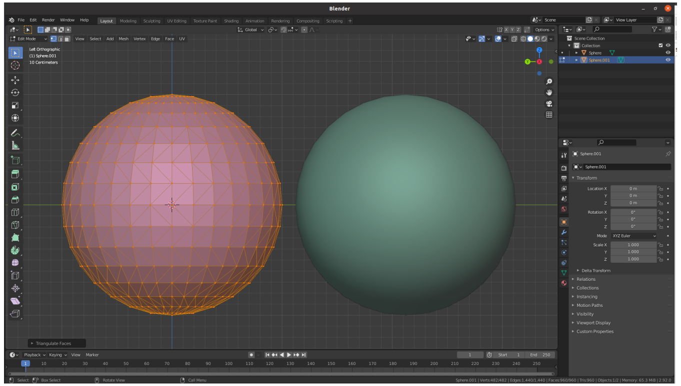
Practical tip: how certain flat shapes are created with triangles



Practical tip: how certain curved shapes are approximated with triangles

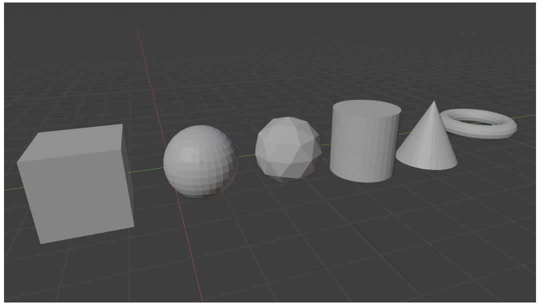
- Conceptual approximation: find points on complex shape and connect adjacent points with a mesh structure
  - For example: scanning and reconstruction
- Example: sphere vs icosahedron
  - Each point on icosahedron is close to point of sphere
  - Each normal vector of icosahedron is close to vector normal 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!





Common basic shapes

- Now we can understand how to represent basic shapes using triangle meshes
- Every DCC Tool provides basic shapes:
  - Blender[1], Maya[2], 3DSMax[3], Houdini[4], etc.



1. https://docs.blender.org/manual/en/latest/modeling/meshes/primitives.html
2. 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. 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. https://www.sidefx.com/docs/houdini/model/create.html

Complex shapes?

- How to represent complex shapes with triangle meshes?
- Digression: drawing complex form (3D shape)
  - Anything can be decomposed in simple forms[1,2]: box, sphere, cylinder, torus, cones, etc.

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=RDCMUCIM2LuQ1q5WEc23462tQzBg&start_radio=1&rv=6T_-DiAzYBc&t=1343&ab_channel=Proko

Practical tip: how complex shapes are made using base shapes

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

Tip: more on modeling of complex shapes in DCC Tools

- Choose right basic shape: https://www.youtube.com/watch?v=DcyY4RAHcA4&ab_channel=CGCookie
- Modeling with basic shapes (Blender):
  - https://www.youtube.com/watch?v=AD3gn2AyzgA&ab_channel=LeeDanielsART
  - Procedural (and funny one): https://www.youtube.com/watch?v=Hf8s1Ckycdo&ab_channel=CGMatter
- Modeling with basic shapes (Maya)
  - https://www.youtube.com/watch?v=j3jwVfN8EcU&ab_channel=AnetaV
- Procedural shapes (Houdini):
  - https://www.youtube.com/watch?v=afHVjiNeH7A&ab_channel=AdrienLambert
  - https://www.youtube.com/watch?v=fxOxygaEOFk&ab_channel=SimonHoudini
  - Note that other geometry representation are used as well

Mesh shape representation: verdict

- Pros:
  - Simple for representing and intuitive for modeling
  - 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.
- Cons:
  - 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

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!>

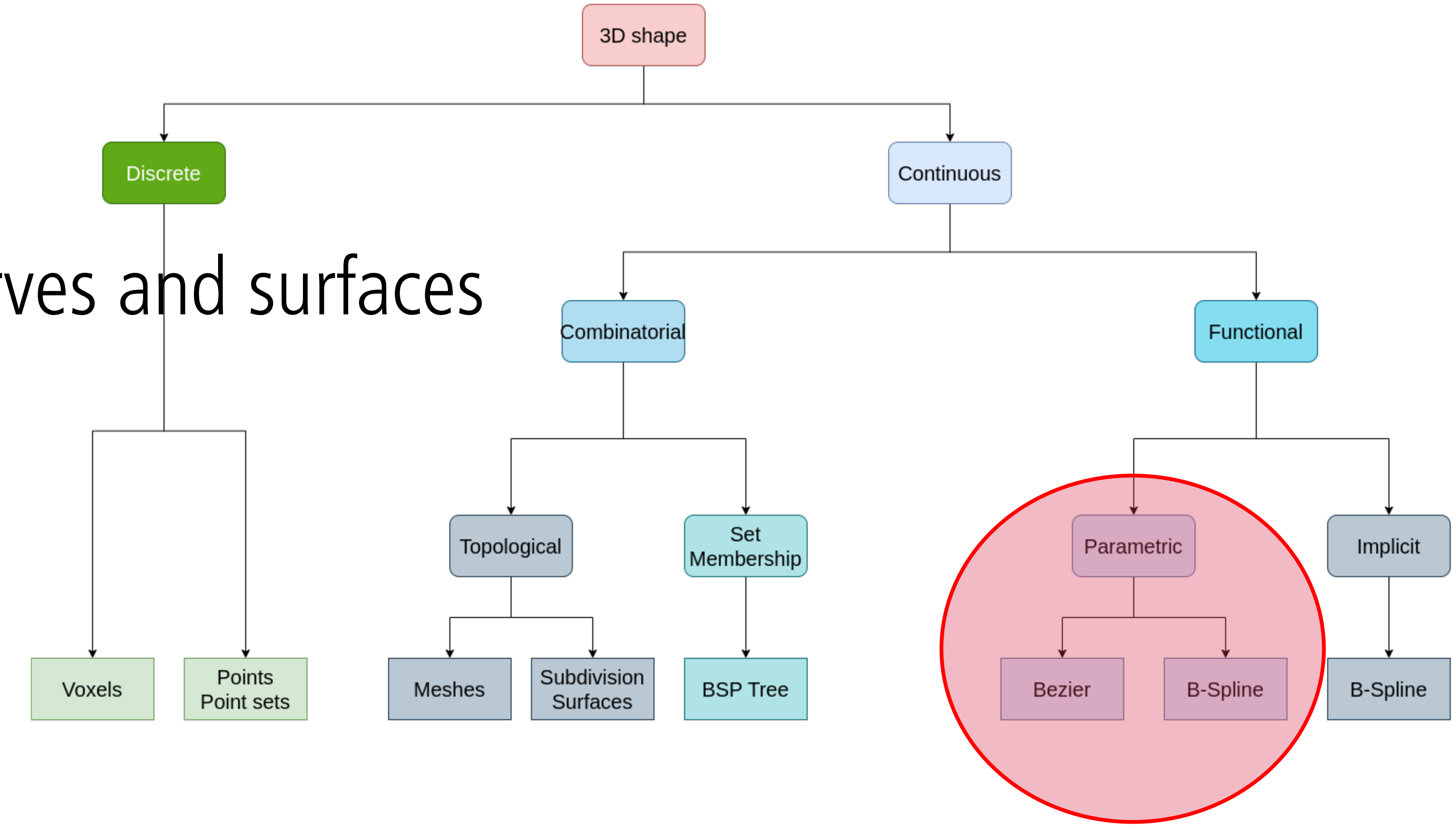
Exploring meshes

- **Meshlab** is open-source tool for mesh manipulation and processing
- Blender mesh modeling
- Sources od meshes:
  - https://casual-effects.com/data/index.html
  - https://polyhaven.com/models
  - https://sketchfab.com/
  - http://graphics.stanford.edu/data/3Dscanrep/
- 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

Parametric curves and surfaces



Different shape representations

- Representing surface using mesh is the most used and widespread option for both authoring and transfer.
  - Triangle mesh and thus triangle is basic atomic rendering primitive for GPU graphics pipelines and most raytracers.
- However, objects made in modeling systems can have many underlying geometric descriptions.
  - Different geometric descriptions enable easier and efficient modeling and representation of shapes on the user side.
  - On the rendering side, all higher-level geometrical descriptions are evaluated as set of triangles and then used.

<IMAGE: show THE CONCEPT OF USER AND RENDERING SIDE AND HOW OBJECTS CAN BE REPRESENTED>

Parametric surfaces vs Mesh

- Parametric surfaces are one of alternative geometric representations that have certain advantages over meshes in certain scenarios.
- Some advantages of curves and curved (subdivision) surfaces are:
  - They are represented by equations and thus have more compact representation than meshes (less memory for storing and transfer) and less transformation operations are needed
  - Since they are represented by equations they provide salable geometric primitives – geometry can be generated on the fly by evaluating the equations (Analogy: vector and raster images)
  - They can represent smoother and more continuous primitives than lines and triangles, thus more convenient for representing object like hair, organic and curved objects
  - Other scene modeling tasks can be performed more simpler and faster, e.g., animation and collision
- <IMAGES: APPLICATION OF CURVES AND CURVED SURFACES>
- To understand curved surfaces, we will start with curves

Parametric curves and splines

- Wide context of usage:
  - Animating object over path: position and orientation <IMAGE>
  - Rendering hair <IMAGE>
- Various implementations
- Described with a formula as a function of parameter t: p(t)
  - t may belong to certain interval [a,b]
  - Generated points are continuous
- Various implementations:
  - Bezier curve
  - Hermite curve
  - Catmull-Rom spline
  - B-Splines

Bezier curves: linear interpolation

- **Linear interpolation** between two points p0 and p1 traces our straight line.
  - $p(t) = (1-t) * p0 + t * p1$
  - lerp(p0,p1,t)
  - For $0 < t < 1$, generated points are on straight line between p0 and p1. Also, $p(0) = p0$ and $p(1) = p1$

<IMAGE: LINEAR INTERPOLATION>

- Linear interpolation is fine for two points. But interpolating between multiple points gives us straight segments with sudden (discontinuous) changes at joints between.

<IMAGE: LINEAR INTERPOLATION MULTIPLE POINTS>

Bezier curves: repeated interpolation

- This problem can be solved by taking linear interpolation one step further and **linearly interpolate repeatedly** → Bezier curves*
- To repeat interpolation, **control points** are added.
- Example: **3 control points**: a, b, c
  - Linearly interpolate a and b to obtain d
  - Linearly interpolate b and c to obtain e
  - Linearly interpolate d and e to obtain curve point f
  - p(t) = lerp(lerp(a,b,t),lerp(b,c,t),t)
  - <image>
- **Degree of curve** is n+1, n – number of control points. <IMAGES>
  - More control points → more degrees of freedom
  - n = 1 → linear interpolation
  - n = 2 → quadratic interpolation
  - n = 3 → cubic interpolation

* Independently discovered by Paul de Casteljau and Pierre Bezier for use in French car industry. This recursive/repeated linear interpolation is called de Casteljau algorithm.

Bezier curves: repeated interpolation

- Bezier curve for n+1 control points:
  - TODO
- Bezier cuves polynomial triangle
  - TODO

Bezier curve: another representation

- As quadratic Bezier, every Bezier curve can be described with algebraic fromula. Therefore, repeated interpolation is not needed.
- Same curve as before can be described using **Bernstein form**:
  - TODO
- Bernstein function contains **Bezier basis function** that defines properties of the curve
  - Curve will stay close to the control points pi. Furthermore, whole Bezier curve will be located in **convex hull** of control points – useful for computing bounding area or volume of curve.
  - Bernstein polynomials can have degrees which define blending – **blending functions**.

Bezier curve: matrix representation

- Bezier equation can be written in matrix form:
  - TODO
  - Geometry matrix
  - Basis matrix

Bezier curves: verdict

- Bezier curves do not pass through all control points (except endpoints)
- Not many degrees of freedom: only control points can be chosen freely.
  - Alternative is rational Bezier curve TODO
- Not every curve can be described with Bezier curve (e.g., circle which must be described with collection of Bezier curves)
- Degree increases with number of control points → complex evaluation for lot control points.
  - For this reason, lower degree curves are concatenated to form larger spline
- Compact form: power form and matrix representation
- Derivative of curve is straightforward

Combining Bezier curves

- Often, multiple bezier curves of lower degree – **cubic** – are joined togeather
  - Cubic curves are lowest degree curves that can describe S-shaped curve called **inflection**
  - This way complexity of computation is simpler since smaller number of control points must be evaluated
  - Resulting cuves will go through set of of points.
- Point where curves are joined are called **joint**
- In simplest case, when last control point of first curve is the first control points of the second curve, results in composite of curves which is not smooth at joint position and called **piecewise Bezier curve**.
- Each curve in this composite is defined by t in [0,1]. Using $t > 1$ requires combining points and parameters of neighboring cuves.
  - TODO

Curves continuity

- Combining curves requires understanding of continuity of composited curves at joints.
- Two measures for continuity: C^n and G^n (geometrical continuity)
  - C^0 – segments should joint at the same point
  - C^1 – derivation of any point (including joints) must be continuous
  - C^2 – first and second derivatives are continuous functions
  - G^0 – positional continuity: holds when the end points of two curves or surfaces coincide
  - G^1 – tangent vectors from curve segments that meet at joint should be parallel and have same direction – no sharp edges. Continuous edges make splines look natural – often sufficient measure
  - G^2 – curvature continuity - tangent vectors from curve segments that meet at joint should be of same length and rate of length change – perfectly smooth surface – two joined surfaces appear as one

Cubic Hermite interpolation

- Bezier curves are good for describing theory behind smooth curves but are not predictable and controllable for authoring.
- Cuves with cubic **Hermite interpolation** are easier to control: instead of 4 control points needed for cubic Bezier cuve, Hermite interpolation requires:
  - 2 points: starting and ending
  - 2 vectors: starting and ending tangents.
  - TODO: FORMULA of hermite interpolant
- Cubic Hermite interpolant is also called cubic Hermite segment or cubic spline segment
- When interpolating more than two points, several Hermite curves can be connected together (similary as done with Bezier).
- Example: https://developer.nvidia.com/gpugems/gpugems2/part-iii-high-quality-rendering/chapter-23-hair-animation-and-rendering-nalu-demo

Assembly of curves

- Connecting multiple points P0...Pn with associated vectors v0...v1?
- Hermite or Bezier interpolation can be used for points (P0,P1)...(Pn-1,Pn) with parameter t [0,1]
- To connect all points, we use **assembly of curves** called **spline**
- Elements of assembly are called **segments** (e.g., Bezier or Hermite sements)
- <IMAGE: ASSEMBLY OF CURVES>
- Splines:
  - Catmull-Rom (special case of Kochanek-Bartel cuves)
  - B-Spline

Catmull-Rom spline

- Assume we have n points, we would like to find a smooth curve passing through point i in time t = i.
- Several cubic* Hermite curves can be joined together. The question is how to find tangents at control points so that desired properties of the spline are found.
- Method for computing tangent is called **Kochanek-Bartels Curves/Spline**
  - Assume that there is only one tangent per control point
  - Tangent at Pi can be computed as combination of two chords: Pi-Pi-1 and Pi+1-Pi
    - FORMULA:
      - Tension parameter: length of tangent (higher values → sharper bends)
      - Bias parameter: direction of tangent
      - Additional: continuity parameter: introducing additional tangent at joint, resulting in outgoing and incoming tangent
- Idea of **Catmull-Rom** is to use previous and next control point as guides to pick tangent. It is a special case of **Kochanek-Bartels Curves/Spline** where all parameters are set to 0 (default).
  - **FORMULA**
    - Rapid finding points on Catmull-Rom spline is possible
    - Interpolating curve: passing through all control points but doesn't stay inside convex hull of points

* Other degrees can be used. Cubic are most often used.

B-spline

- B-spline is similar to Bezier curve: it is function of t and weighted control points.
  - FORMULA
  - Segment can be expressed in matrix form
- Often cubic B-Spline is used
- B-spline is similar to Catmull-Rom, except:
  - It is C^2
  - It is non-interpolating (it is passing near control points, not through them)
- Two flavours:
  - Uniform
  - Non-Uniform
- Generalizations:
  - Rational
- NURBS – non-uniform rational B splines

Uniform cubic B-spline

- Uniform – spacing between control points is uniform
- Basis function
  - FORMULA
  - Has C2 continuity everywhere: if several B-splines are joined, the composite curve will be C2
    - Curve of degree n has C^{n-1} continuity
  - Basis function is built using integration of previous basis function
- Multiple uniform cubic B-splines curves can be joined together as a spline
  - C2 continuity everywhere
  - No guarantee that it is interpolating the control points
  - TODO

Non-uniform rational B-splines

- Non-Uniform – spacing between control points is not uniform
- Very often used in CAD tools
- EXAMPLE

Parametric surfaces

- After getting familiar with curves, natural extension are **parametric surfaces**
  - Similarly as triangle or polygon is extension of a line segment
- Very useful for modeling curved surfaces
  - EXAMPLE
- As curves, parametric surfaces are defined with small number of control points
- Model made with parametric surfaces is tessellated for efficient rendering process.
  - Surface can be tessellated in any number of triangles making it perfect for tuning trade-off between quality and speed (more triangles → better shading and silhouettes)
  - Another advantage is that animation can be done on control points and then surface is tessellated for rendering.
- Parametric surfaces:
  - Bezier patches
  - Bezier triangle
  - B-spline patch

Bezier Patches

- Bezier curve is extended so it has two parameters (u,v) which define surface
- Similarly as we started with Bezier curve by explaining linear interpolation, we explin Bezier patch by explaining **bilinear interpolation***.
  - IMAGE: bilinear interpolation using 4 points: a,b,c,d
  - e(u) = lerp(a,b,u), f(u) = lerp(c,d,u), p(u,v) = lerp(e(u), p(u), v)
  - p(u,v) is simplest, non-planar parametric surface with (u,v) in [0,1]. It has **rectangular domain** and thus resulting surface is called a **patch**.

* Bilinear interpolation is crucial for computations in computer graphics. It is extensively used and one example is texture mapping.

Bezier Patches

- Similarly as we added more points to linear interpolation to obtain Bezier curve, we add more points to bilinear interpolation to obtain Bezier patch
  - EXAMPLE: biquadratic Bezier patch
  - Nine points arranged in 3x3 grid

Bezier patches

- Repeated bilinear interpolation is extension of de Casteljau's algorithm to patches.
- De Casteljau patches
  - FORMULA
  - Degree of surface: n
  - Control points Pij where i and j belong to [0...n]
- Point on Bezier Patch can be described in **Bernstein form** using Bernstein polynoms
  - EXAMPLE
  - Parameters m and n: bilinear interpolation is performed n times and linear interpolation m-n times
- Properties:
  - Passes through only corner control points
  - Boundary of the patch is described with Bezier curve of degree n formed by the points on the boundary
  - Tangents at border points are described with Bezier curve at border points – each corner control point has two tangents: for u and v direction
  - Patch lies within convex hull of its control points.
  - Control points can be generated and then points on patch will be rotated when evaluated (faster than other way around)
  - Derivative is straightforward.

Rational Bezier patches

- Extension of bezier patches (similarly as for Bezier curves)
- TODO

Bezier patches: examples

- EXAMPLE: how surface look defined with several control points. How moving of the points influences the surface.

Other parametrized surfaces

- **Bezier triangles**
  - Useful when parametric surface is constructed from a triangle using PN triangles of Phong tessellation methods*.
  - Control points are located in triangulated grid
  - Based on repeated interpolation: de Casteljau, Bernstein triangles
  - Constructing complex object requires stitching Bezier triangles so that composite surface contains desired properties and look: continuity

<EXAMPLES>

* Game engines (e.g., unity and unreal) support those methods since triangle mesh is basic building primitive.

Other parametrized surfaces

- **Point-Normal (PN) Triangles**
  - Given triangle mesh with normals at each vertex, the goal is to construct “better looking” surface using just triangles
  - This data is enough to construct surface
  - PN methods tries to improve mesh shading and silhouettes by creating **curve surface to replace each triangle**
- Properties:
  - Creases in PN triangles are hard to control
  - Continuity between Bezier triangles is C0 but looks acceptable for certain applications
- <EXAMPLES>

Other parametrized surfaces

- **Phong tessellation**
  - Similar as PN triangles, given the triangle points with normals, construct surface
  - Phong tessallation attempts to create geometric version of Phong shading normal using repeated interpolation resulting in Bezier triangles.
  - <EXAMPLES>

B-Spline surfaces

- **B-Spline curves** can be extended to B-Spline surfaces which are similar to Bezier Surface
- Often bicubic B-Spline surface is used:
  - to form composite surface
  - Essencial for Catmull-Clark subidivison surfaces
  - <EXAMPLES>
- Non-uniform rational B-Spline surface is often used in 3D modeling software
  - EXAMPLES

Rendering of curved surfaces

- For rendering purposes (both rasterization- and raytracing-based) it is beneficial to transform curved surface into triangulated mesh – tessellation
- EXAMPLES

Modeling with parametric surfaces

- CAD software

Storing and transferring parametric surfaces

- STEP file format
- IGES file format

Exploring parametric curves and surfaces

- Blender NURBS surfaces: https://docs.blender.org/manual/en/latest/modeling/surfaces/introduction.html
- Blender NURBS and Bezier curves: https://docs.blender.org/manual/en/latest/modeling/curves/index.html
- Tutorial on curves in Blender: https://behreajj.medium.com/scripting-curves-in-blender-with-python-c487097efd13
- Houdini: https://www.sidefx.com/docs/houdini/nodes/sop/curve.html
- Library for creating and manipulating NURBS surfaces and curves: http://verbnurbs.com/
- More examples: https://www.realtimerendering.com/#curves

Further into topic

- Parametric curves and surfaces are highly used and researched method in computer graphics. Until now we have covered foundations. There are many other topics. Some of those will be discussed later, some are out of scope for this course:
  - NURBS: https://www.gamedeveloper.com/programming/using-nurbs-surfaces-in-real-time-applications

Literature

- https://github.com/lorentzo/IntroductionToComputerGraphics/wiki/Foundations-of-3D-scene-modeling