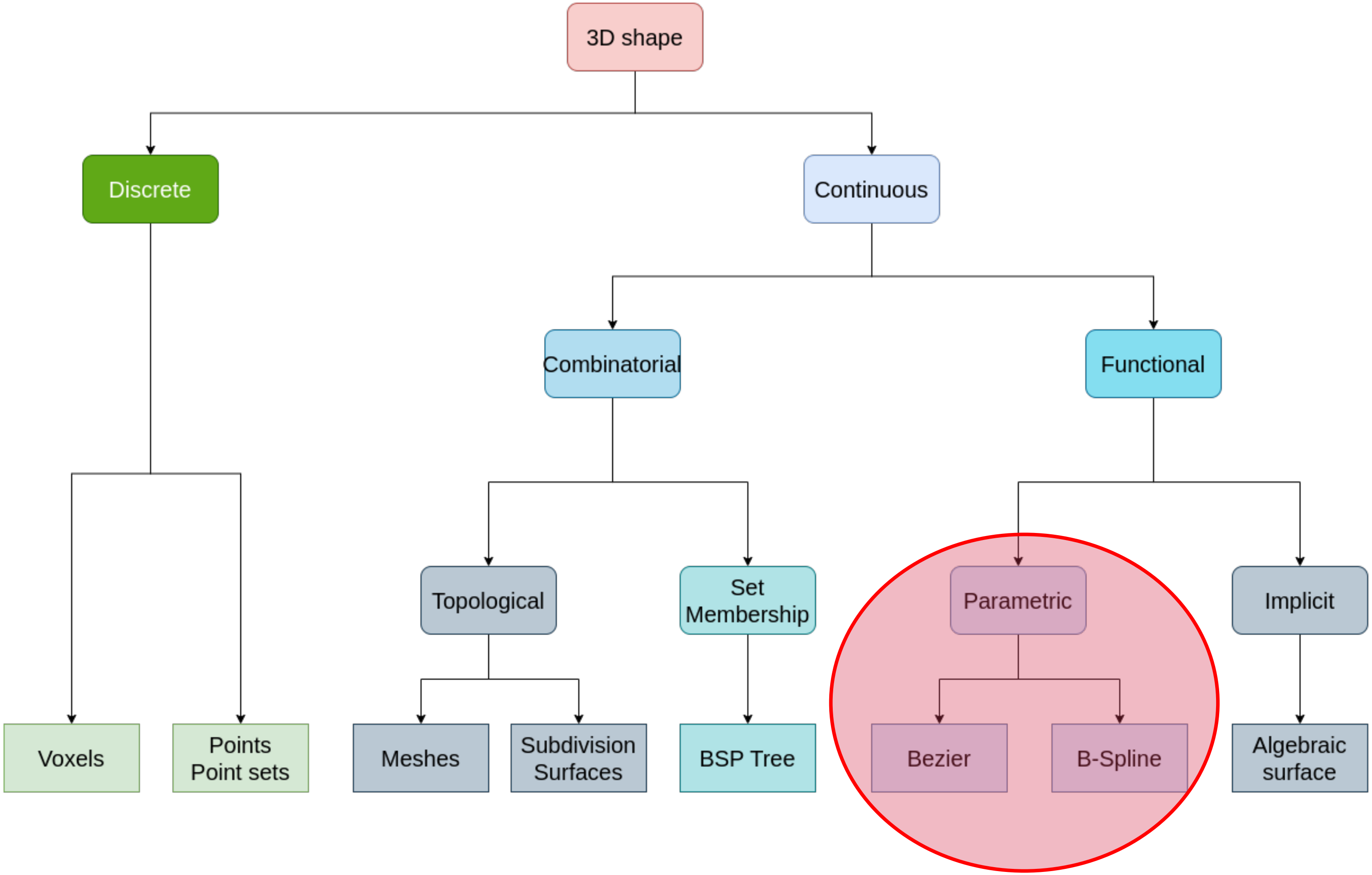
Parametric curves and surfaces


3D shape
Discrete
Continuous
Combinatorial
Functional
Voxels
Points
Point sets
Topological
Set
Membership
Parametric
Implicit
Meshes
Subdivision
Surfaces
BSP Tree
Bezier
B-Spline
Algebraic
surface

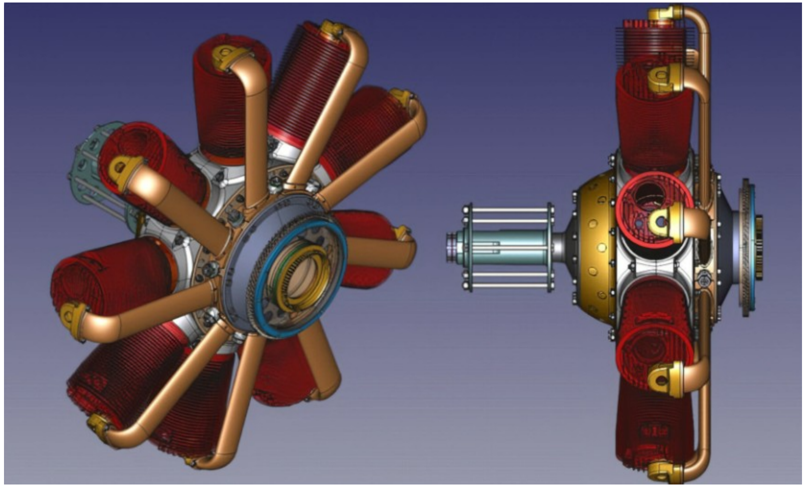
Another shape representation

- 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 ray-tracers.
- 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.

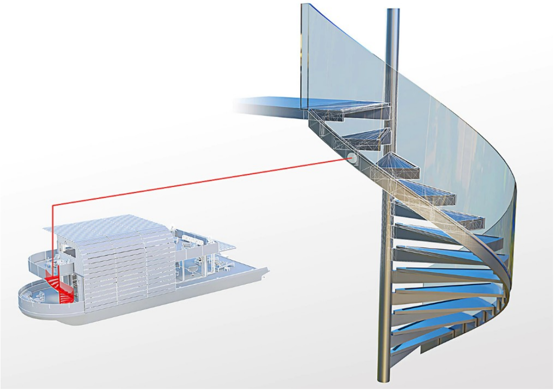
<IMAGE: show THE CONCEPT OF USER AND RENDERING SIDE AND HOW OBJECTS CAN BE REPRESENTED>

Applications

- Design: cars, ships, airplanes, etc.
- architecture,
- Engineering,
- Nature,
- Animation,
- Vector graphics,



https://www.freecadweb.org/



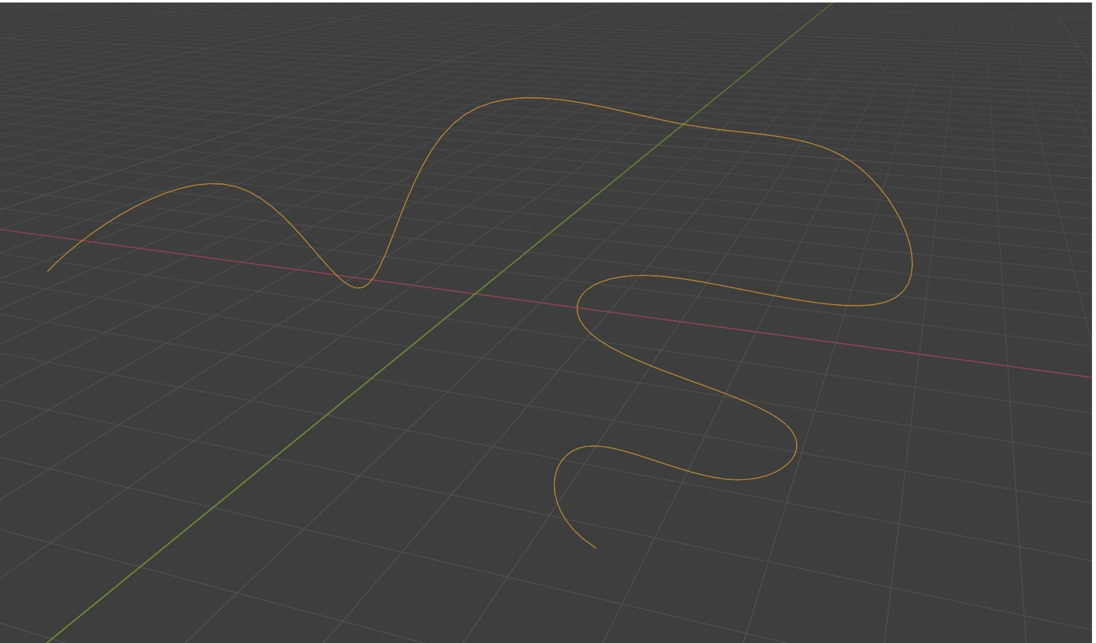
https://www.autodesk.com/products/autocad/included-toolsets/autocad-architecture

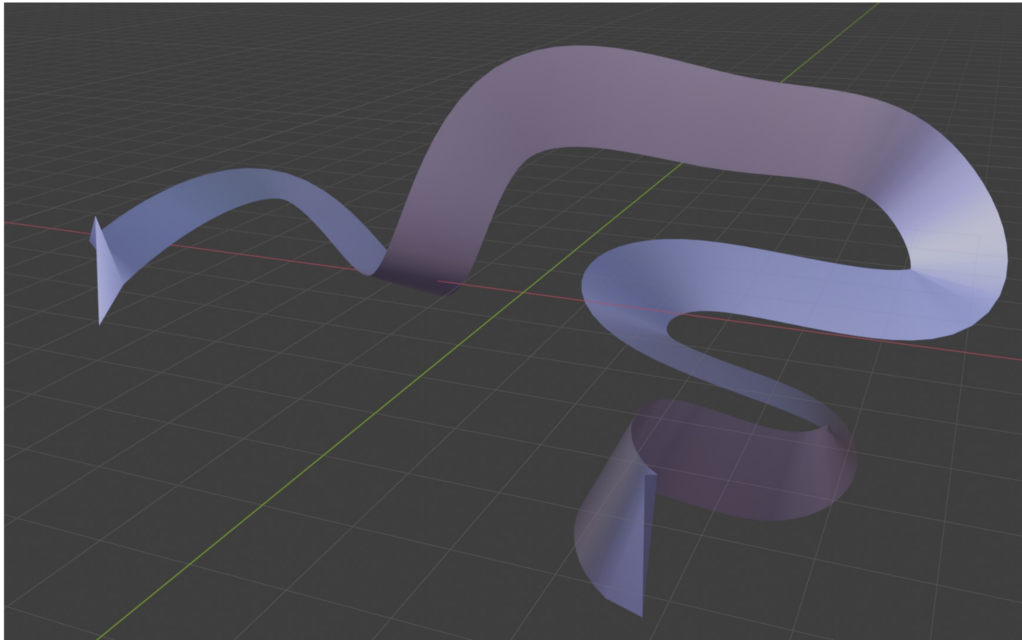
Parametric surfaces vs Mesh

- 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>

Parametric curves and surfaces

- To understand parametric surfaces, we will start with parametric curves

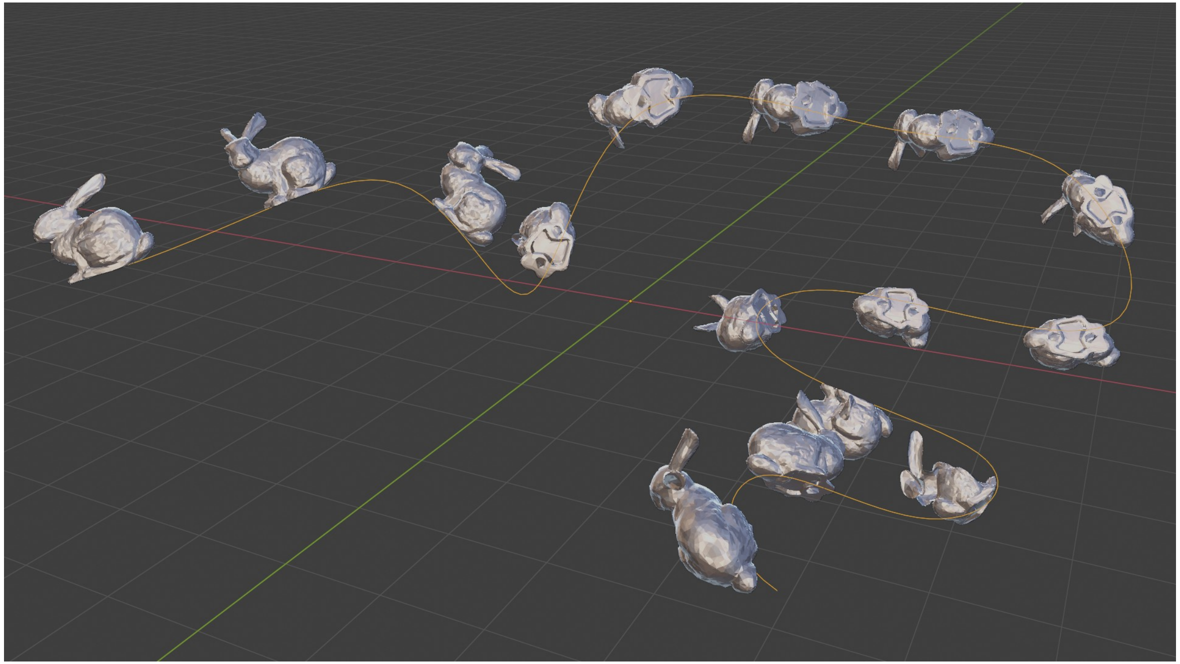




Parametric curves

Parametric curves

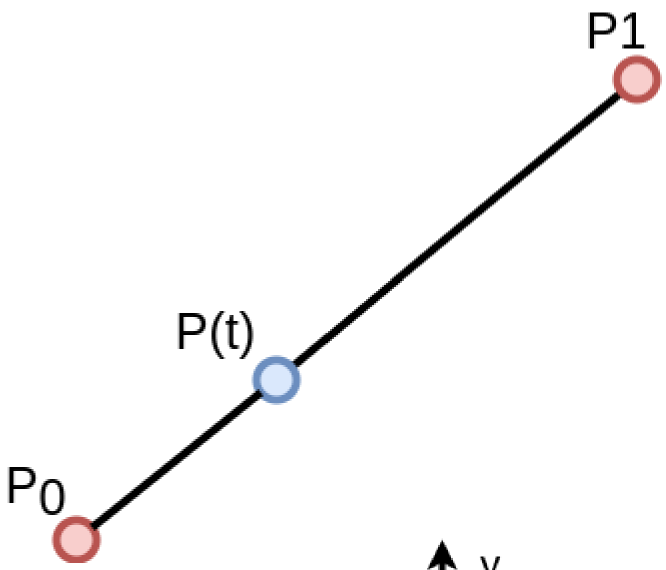
- Wide context of usages:
  - Animating object/camera/light over path: position and orientation
  - Rendering hair

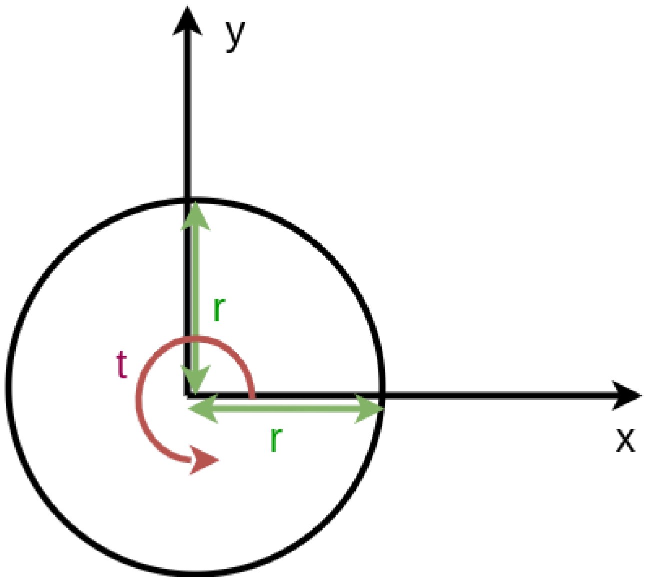


https://developer.nvidia.com/gpugems/gpugems2/part-iii-high-quality-rendering/chapter-23-hair-animation-and-rendering-nalu-demo

Parametric curves

- Described with a formula as a function of parameter t: p(*t*), *t* in [a,b]
  - Generated points are continuous
- Example: **line segment**
  - $p(t) = (1-t)\, p_0 + t\, p_1$
- Example: **circle**
  - $x(t) = r \cos(2 * PI * t)$, t in [0,1]
  - $y(t) = r \sin(2 * PI * t)$, t in [0,1]

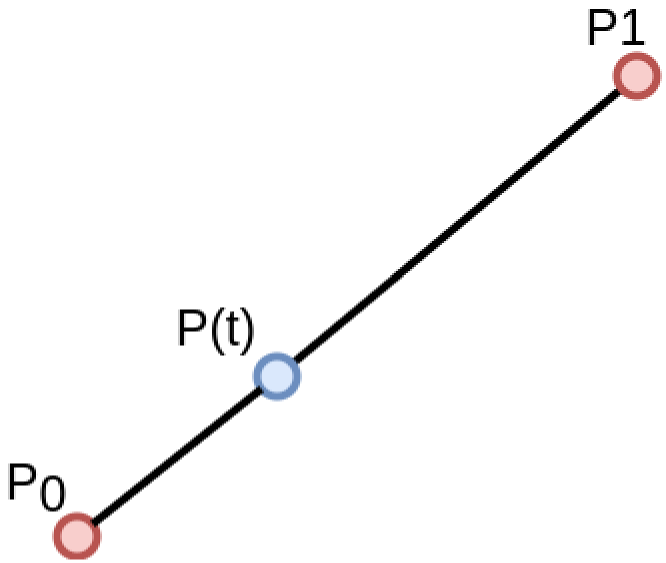




- Arbitrary curves? Various implementations:
  - Bezier curve
  - Hermite curve
  - Catmull-Rom spline
  - B-Splines

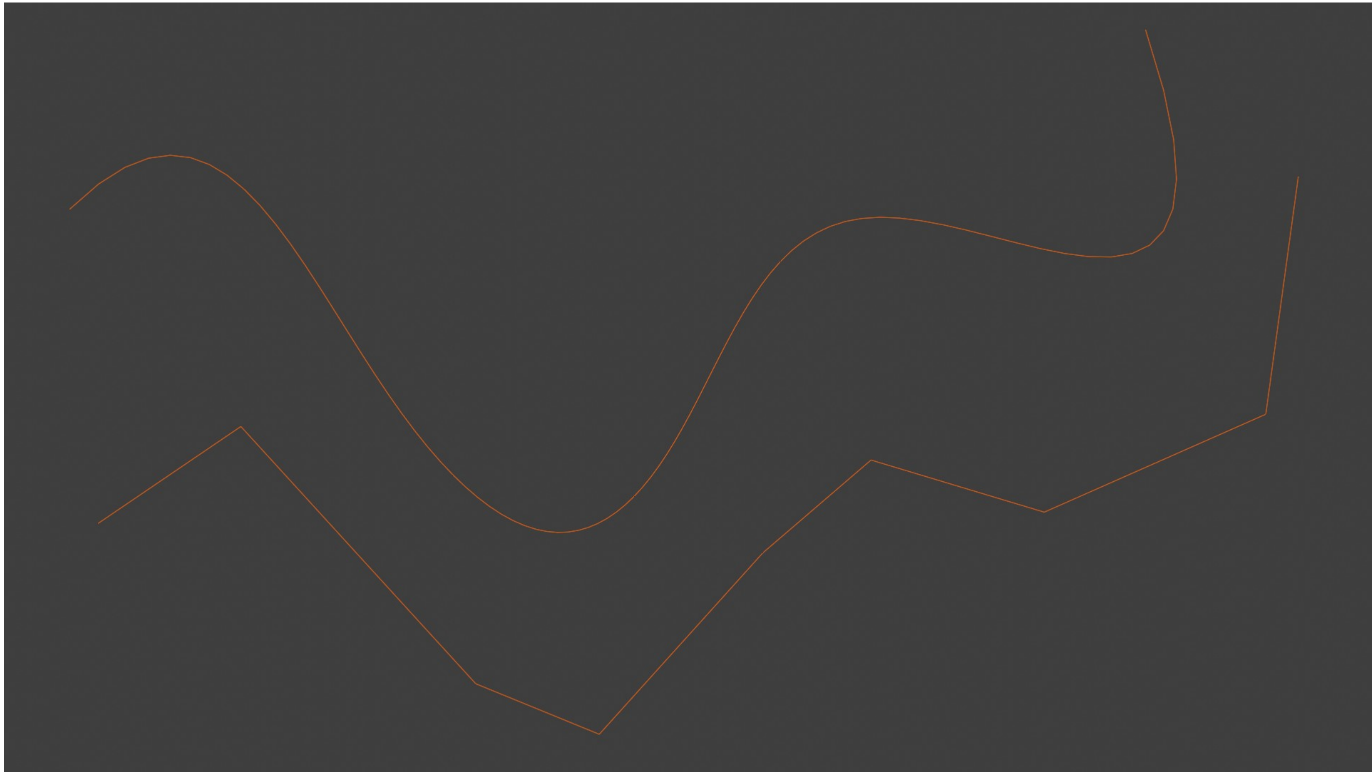
Bezier curves: linear interpolation

- **Linear interpolation** between two points – **control points** - p_0 and p_1 traces out straight line.
  - $p(t) = (1-t)\, p_0 + t\, p_1$, short: `lerp(p₀, p₁, t)`
  - For $0 < t < 1$, generated points are on straight line between p_0 and p_1. Otherwise outside.
  - $p(0) = p_0$ and $p(1) = p_1$

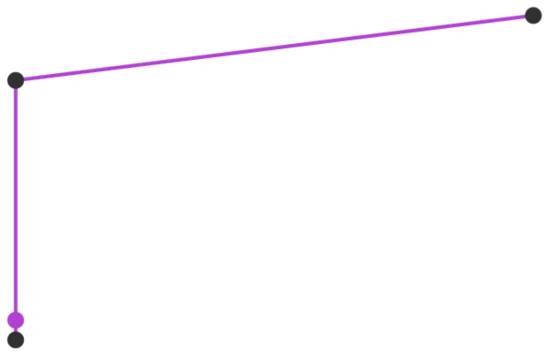


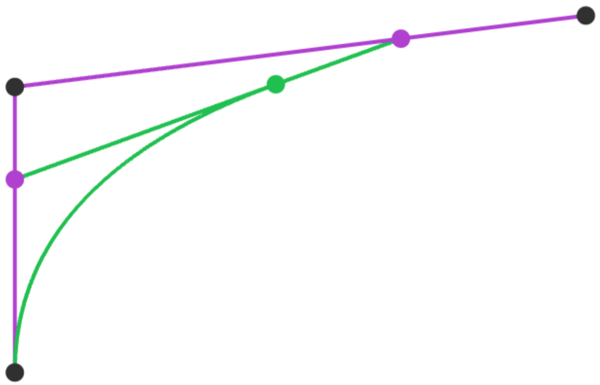
Bezier curves: linear interpolation

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



Bezier curves: repeated interpolation

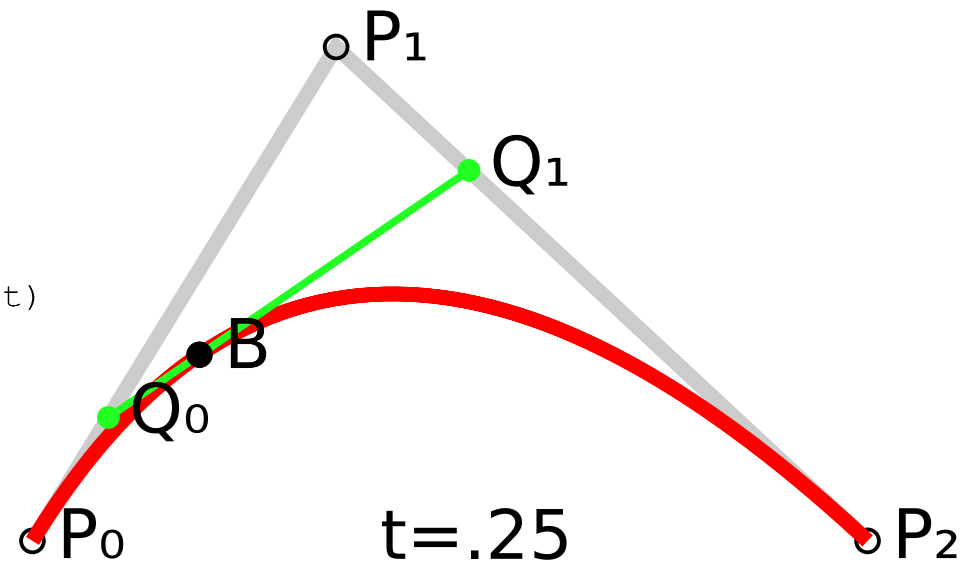




https://acko.net/files/fullfrontal/fullfrontal/wdcode/online.html

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: P_2
  - Example: **3 control points**: P_0 ,P_1, P_2
  - Linearly interpolate P_0 and P_1 to obtain Q_0
  - Linearly interpolate P_1 and P_2 to obtain Q_1
  - Linearly interpolate Q_0 and Q_1 to obtain curve point B
  - `B(t) = lerp(lerp(P₀,P₁,t),lerp(P₁,P₂,t),t)`

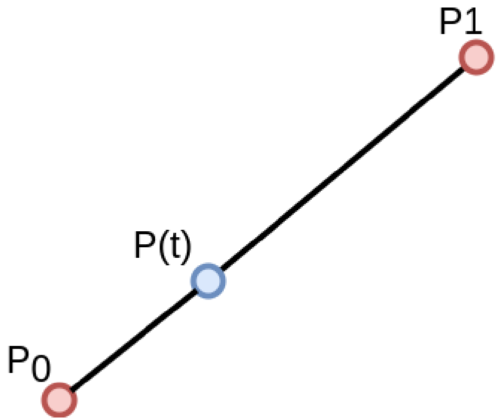


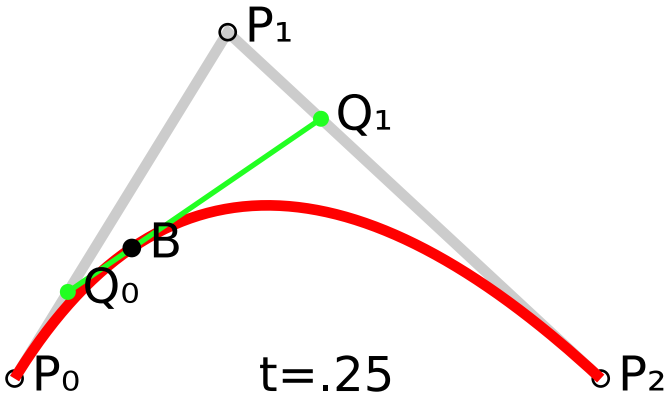
https://en.wikipedia.org/wiki/B%C3%A9zier_curve

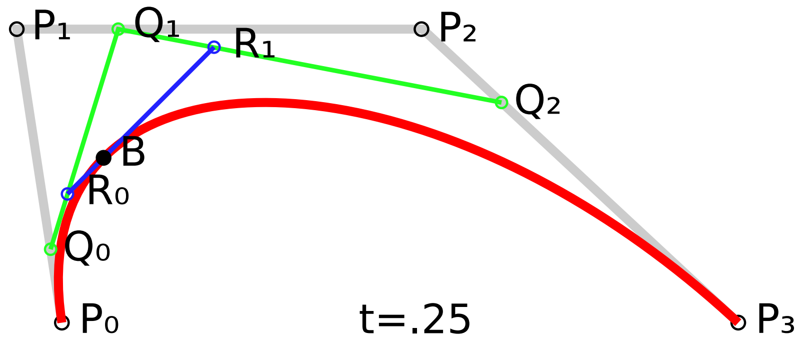
* Independently discovered by Paul de Casteljau and Pierre Bezier for use in French car industry.

Bezier curves: degrees

- **Degree of curve** is n, if n +1 control points are used.
  - More control points → more degrees of freedom
  - $n = 1 \rightarrow 1^{st}$ degree, **linear curve**
  - $n = 2 \rightarrow 2^{nd}$ degree, **quadratic curve**
  - $n = 3 \rightarrow 3^{rd}$ degree, **cubic curve**







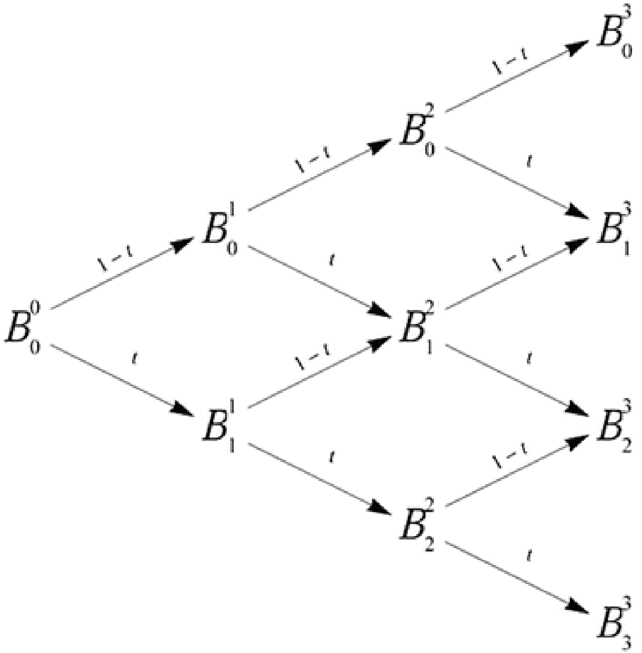
Bezier curves: repeated interpolation

- Repeated or recursive linear interpolation is often referred as **de Casteljau algorithm**.
- Linear Bezier: $$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1,\ 0 \le t \le 1$$
- Quadratic Bezier: $$\mathbf{B}(t) = (1-t)[(1-t)\mathbf{P}_0 + t\mathbf{P}_1] + t[(1-t)\mathbf{P}_1 + t\mathbf{P}_2],\ 0 \le t \le 1,$$
- Cubic Bezier: …
- Higher degree: recursive formula:

$$\mathbf{B}_{\mathbf{P}_0}(t) = \mathbf{P}_0, \text{ and}$$

$$\mathbf{B}(t) = \mathbf{B}_{\mathbf{P}_0\mathbf{P}_1\ldots\mathbf{P}_n}(t) = (1-t)\mathbf{B}_{\mathbf{P}_0\mathbf{P}_1\ldots\mathbf{P}_{n-1}}(t) + t\mathbf{B}_{\mathbf{P}_1\mathbf{P}_2\ldots\mathbf{P}_n}(t)$$

- De Casteljau algorithm gives pyramid of coefficients



Bezier curve: another representation

- Quadratic Bezier: $\mathbf{B}(t) = (1-t)[(1-t)\mathbf{P}_0 + t\mathbf{P}_1] + t[(1-t)\mathbf{P}_1 + t\mathbf{P}_2],\ 0 \le t \le 1,$
- Quadratic Bezier re-arranged → algebraic description

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2,\ 0 \le t \le 1$$

- Every Bezier curve can be described with algebraic fromula. Therefore, **repeated interpolation is not needed.**
- Generalized algebraic description: **Bernstein form**:

$$\begin{aligned}\mathbf{B}(t) &= \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i \\ &= (1-t)^n \mathbf{P}_0 + \binom{n}{1}(1-t)^{n-1} t\mathbf{P}_1 + \cdots + \binom{n}{n-1}(1-t)t^{n-1}\mathbf{P}_{n-1} + t^n\mathbf{P}_n, \qquad 0 \leqslant t \leqslant 1\end{aligned}$$

Bezier curve: Bernstein form

- Bernstein form:

$$\mathbf{B}(t) = \sum_{i=0}^{n} b_{i,n}(t)\mathbf{P}_i, \quad 0 \leq t \leq 1$$

- **Bernstein polynomials** aka **Bezier basis functions**:

$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, \ldots, n$$

- If n = 2 (quadratic)

$$B(t) = \sum_{i=0}^{2} \binom{2}{i} t^i (1-t)^{(2-i)}$$

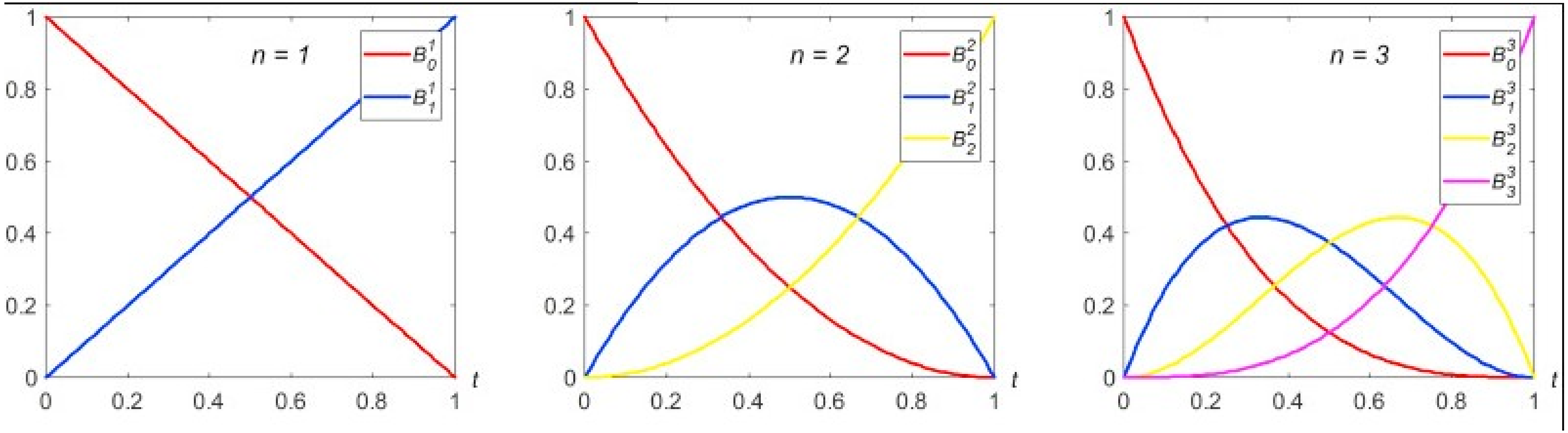
$$B(t) = \binom{2}{0} t^0 (1-t)^2 P_0 + \binom{2}{1} t^1 (1-t)^1 P_1 + \binom{2}{2} t^2 (1-t)^0 P_2, 0 \leq t \leq 1$$

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2,\ 0 \leq t \leq 1$$

Bernstein polynomials

- When t increases, blending weight form P0 decreases and blending weight for p1 increases, and so on…

$$\mathbf{B}(t) = \sum_{i=0}^{n} b_{i,n}(t)\mathbf{P}_i, \quad 0 \le t \le 1 \qquad b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, \ldots, n$$



Bernstein polynomials aka blending functions

https://www.sciencedirect.com/science/article/abs/pii/S0997753817306381

Bernstein polynomials

- Bernstein function contains **Bezier basis function (Bernstein polynomials)** that defines properties of the curve:

$$b_{i,n}(t) \in [0,1] \text{ when } t \in [0,1]$$

- Polynomials are in [0,1] when t is given in [0,1]

$$\sum_{i=0}^{n} b_{i,n}(t) = 1$$

- Curve will stay close to the control points P_i.
  - Furthermore, whole Bezier curve will be located in **convex hull** of control points – useful for computing bounding area or volume of curve.

Bezier curve: matrix representation

- In practice, it is useful (faster calculation) to represent Bezier curve in matrix form.
- $$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3,\ 0 \le t \le 1$$

$$B(t) = \begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ 1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

https://blog.demofox.org/2016/03/05/matrix-form-of-bezier-curves/

Bezier curves: good properties

- Intuitive theory, good for understanding curves: repeated interpolation
- Compact form: Bernstein form and matrix representation, power form*
- Derivative of curve is straightforward: derivation of polynomial → tangent vector of curve point
- Useful: Arbitrary number of points can be generated on curve
  - If rotation of those points is needed, then curve (few control points) are rotated and then points are generated.
- More on topic: https://www.youtube.com/watch?v=aVwxzDHniEw&ab_channel=FreyaHolm%C3%A9r

* Real-time rendering book, ch 17.1.

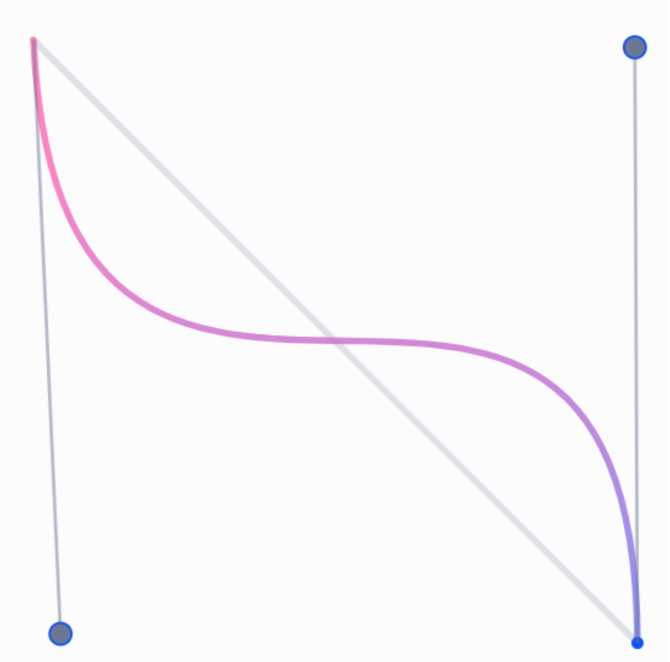
Bezier curves: problems

- Bezier curves do not pass through all control points (except endpoints)
- Not many degrees of freedom:
  - Only control points can be chosen freely
  - Not every curve can be described with Bezier curve (e.g., simple circle can not be described with one or collection of Bezier curves)
  - Alternative is rational Bezier curve*
- Degree increases with number of control points
  - Hard to control if higher degree curve is used, complex computation
  - Bernstein polynomials do not interpolate well
  - For this reason, lower degree curves (often cubic curves) are concatenated to form larger spline

* Hoschek, J. and Lasser, D. (1993) Fundamentals of Computer Aided Geometric Design. A K Peters.

Joining Bezier curves

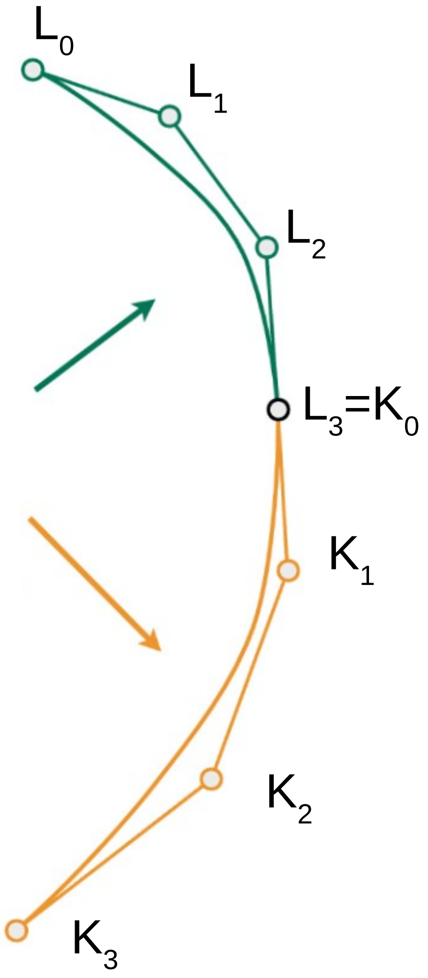
- Often, **multiple Bezier curves** of lower degree are joined together to form **splines**
  - Lower degree curves lower the complexity of computation
  - Resulting curves will go through set of of points
- For this purpose, **cubic degree** is often used
  - Cubic curves are lowest degree curves that can describe S-shaped curve called **inflection**



Cubic Bezier with inflection: https://blog.maximeheckel.com/posts/cubic-bezier-from-math-to-motion/

Joining cubic Bezier curves

- Example: two cubic Bezier curves with 4 control points:
  - L_i, i = 0, 1, 2, 3
  - K_i, i = 0, 1, 2, 3
- To join the curves we can set $L_3 = K_0$ → **joint** - point where curves are joined
- Composite curve formed from several curve pieces is called **piecewise Bezier curve, p(t)**
  - Now t is in [t1, t2]
  - First curve is p(t'), where t' in [0,1].
  - Second curve is p(t'), where t' = (t – t1) / (t2 - t1)
- Two curves connected just using $L_3 = K_0$ will not be smooth at joint.
- Improved smoothness is achieved using tangent constraint: $(K_1 - K_0) = c\,(L_3 - L_2)$, $c > 0$



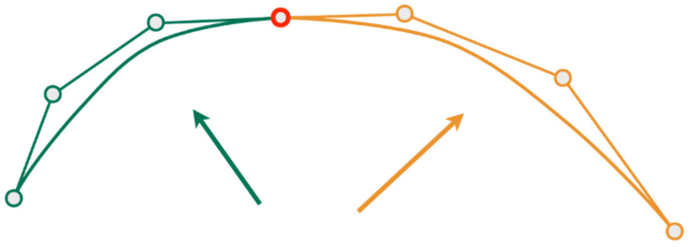
* Curves and Surfaces in Computer Aided Geometric Design, Farin, Gerald

https://cs184.eecs.berkeley.edu/sp22/lecture/7-77/intro-to-geometry-splines-and-be

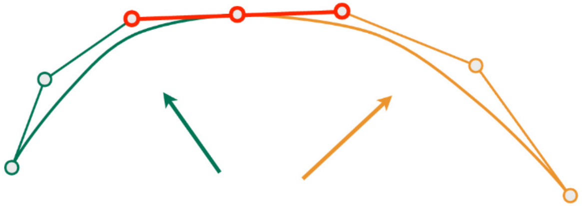
Joined curves continuity

- This way, many cubic Bezier curves are chained into piecewise cubic Bezier line
- Continuity of joined curves:
  - C^0 – **positional continuity** - segments should joint at the same point
  - C^1 – **velocity continuity** - derivation of any point (including joints) must be continuous
  - C^2 - **acceleration continuity** - first and second derivatives are continuous functions

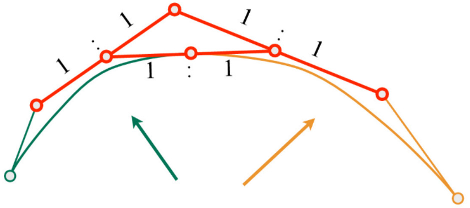
C^0



C^1



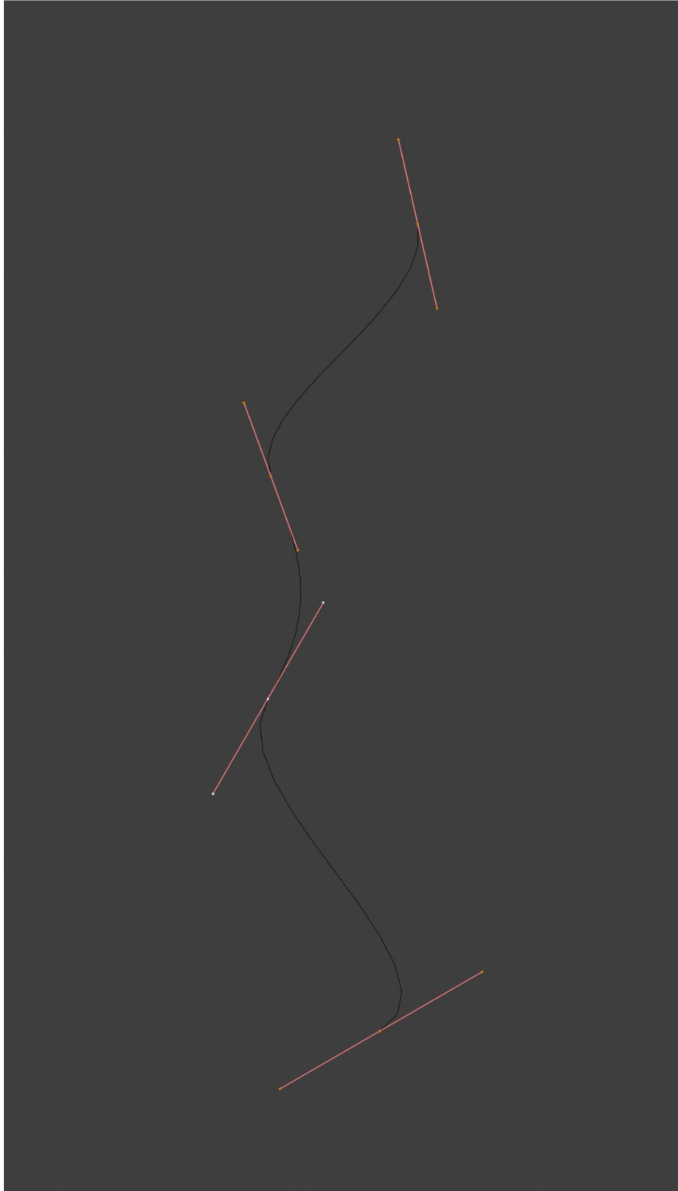
C^2 – "A Frame" construction



https://cs184.eecs.berkeley.edu/sp22/lecture/7/intro-to-geometry-splines-and-be

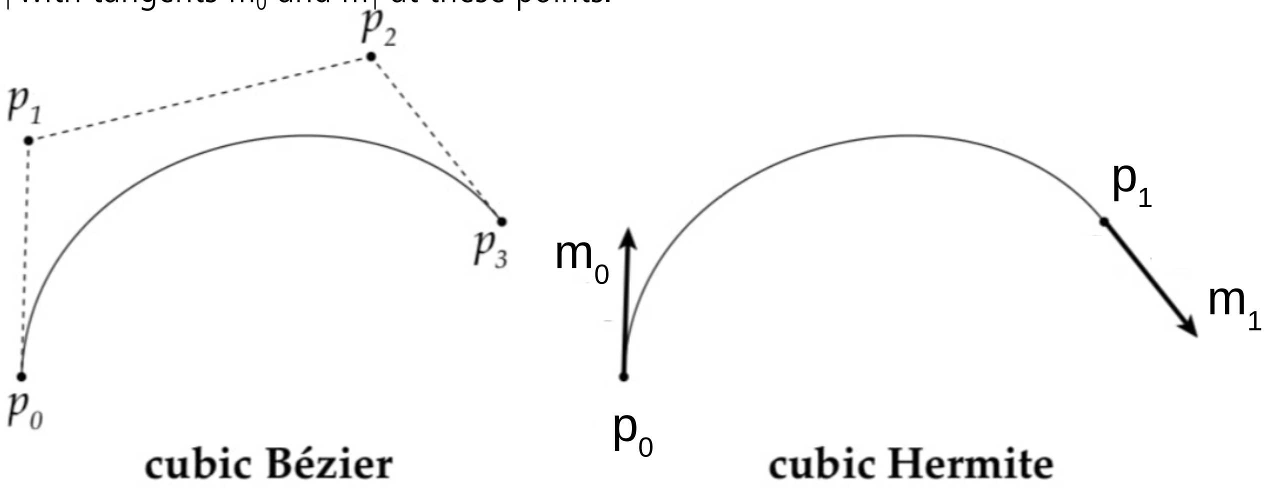
Joined curves continuity

- Geometrical continuity for describing joined curve smoothness
  - G^0 – **positional continuity**: holds when the end points of two curves or surfaces coincide
  - G^1 – **tangent continuity** - 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 curve

- Bezier curves are good for describing theory behind smooth curves but are not controllable for authoring.
- Cuves with cubic **Hermite interpolation** are easier to control and require:
  - 2 points. Starting and ending control points: p_0 and p_1
  - 2 vectors. Starting and ending tangents: m_0 and m_1 (longer tangents influence the shape)
- Cubic Hermite curve **interpolates** p_0 and p_1 with tangents m_0 and m_1 at these points.



http://15462.courses.cs.cmu.edu/fall2020/article/10

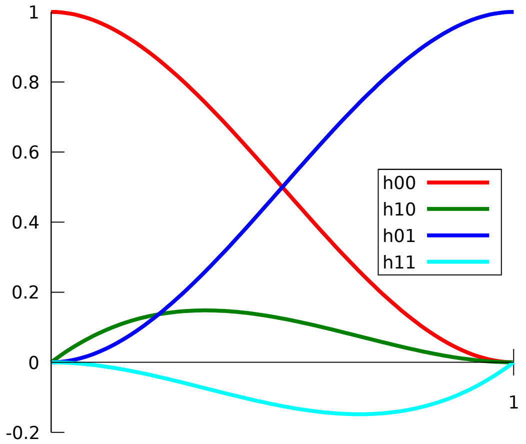
Joining cubic Hermite curve

- Cubic Hermite curve (aka interpolant aka segment aka spline segment) p(t), t in [0,1]:

$$\boldsymbol{p}(t) = (2t^3 - 3t^2 + 1)\boldsymbol{p}_0 + (t^3 - 2t^2 + t)\boldsymbol{m}_0 + (-2t^3 + 3t^2)\boldsymbol{p}_1 + (t^3 - t^2)\boldsymbol{m}_1$$

- Similarly as discussed with Bezier: **Bernstein form** with **Hermite basis function** exist
- When interpolating more than two points, several Hermite curves can be connected together (similarly as done with Bezier).

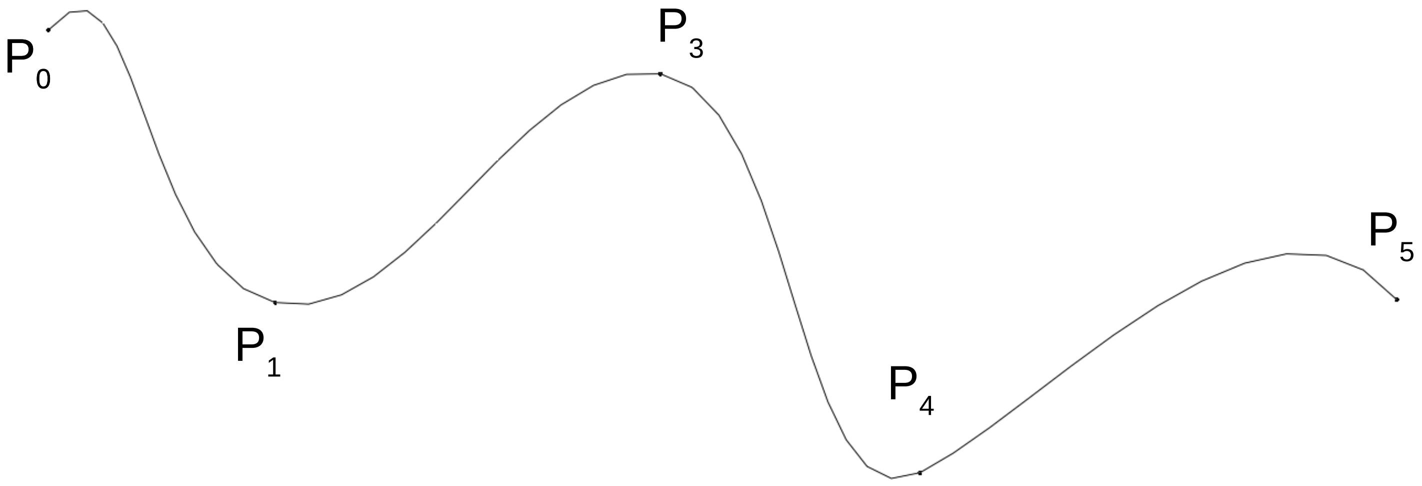
$$p(t) = (t^3\ t^2\ t\ 1\ ) \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$



https://en.wikipedia.org/wiki/Cubic_Hermite_spline

Splines: assembly of curves

- Connecting multiple points $P_0 \ldots P_n$
  - n – 1 cubic Hermite curves must be used
  - resultant is **assembly of curves** called **spline**
  - elements of assembly are called **segments** (e.g., Hermite sements)



Catmull-Rom spline

- Given points $P_0 \ldots P_n$, the goal is to calculate tangents for Hermite curves which would create spline: continuous differentiable curve passing through each point.
- Solution:
  - Catmull-Rom spline

Catmull-Rom spline

- **Kochanek-Bartels:** method for computing tangents at joints
- Assume that there is only one tangent per control point
  - Tangent at Pi can be computed as combination of two chords: P_i-P_{i-1} and P_{i+1}-P_i:

$$m_i = \frac{(1-a)(1+b)}{2}(p_i - p_{i-1}) + \frac{(1-a)(1-b)}{2}(p_{i+1} - p_i)$$

  - **Tension parameter - a**: length of tangent, higher values → sharper bends
  - **Bias parameter - b**: direction of tangent
- **Kochanek-Bartels method** can be extended with additional, continuity, parameter and which is incorporated with calculating 2nd tangent at joint*.

* Real-time rendering book, ch 17.1.

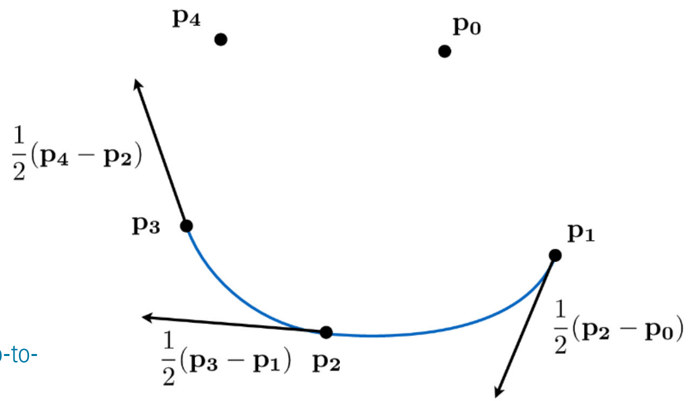
Catmull-Rom spline

- **Catmull-Rom** spline is a special case of **Kochanek-Bartels Spline** where tension and bias parameters are set to 0. Tangents in $P_0 \ldots P_n$ are calculated using:

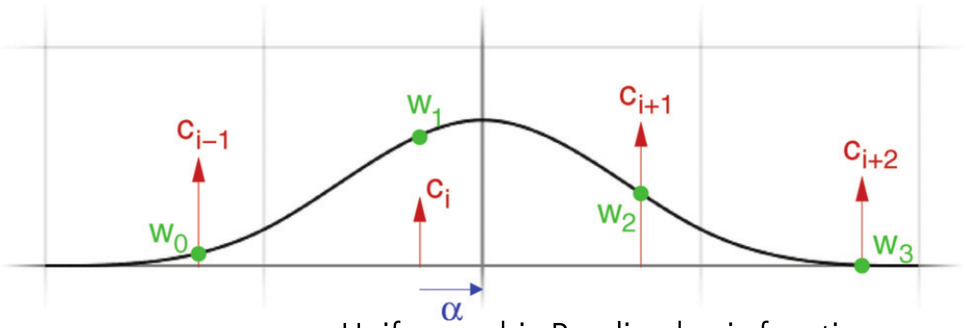
$$m_i = \frac{1}{2}(p_i - p_{i-1}) + \frac{1}{2}(p_{i+1} - p_i)$$

- Properties:
  - 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

https://cs184.eecs.berkeley.edu/sp22/lecture/7-50/intro-to-geometry-splines-and-be



B-spline



Uniform cubic B-spline basis function

- Two types:
  - Uniform: uniformly space control points
  - Non-Uniform
- Similarly to Bezier curve, it is parameterized function of x

$$s_n(x) = \sum_{k \in \mathbb{Z}} c(k) \beta_n(x - k).$$

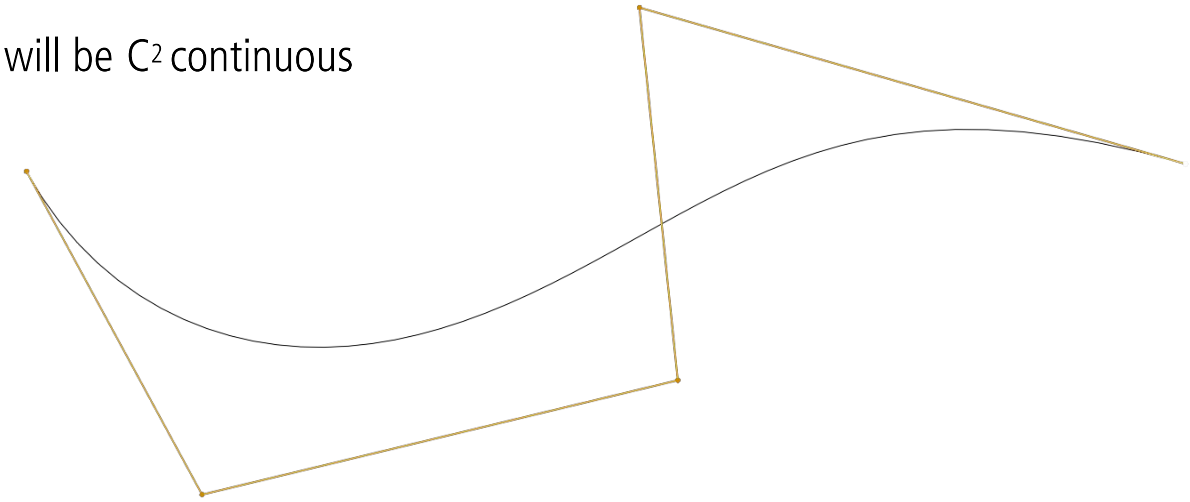
  - Where: **n-1** – degree, **c(k)** – control points, β_n - basis function
- Cubic B-spline is most widely used. Uniform cubic B-spline basis function:

$$\beta_3(x) = \begin{cases} 0, & |x| \geqq 2 \\ \frac{1}{6} \cdot (2 - |x|)^3, & 1 \leqq |x| < 2 \\ \frac{2}{3} - \frac{1}{2}|x|^2 \cdot (2 - |x|), & |x| < 1 \end{cases}$$

https://www.researchgate.net/publication/220494113_Efficient_GPU-Based_Texture_Interpolation_using_Uniform_B-Splines

Uniform cubic B-spline

- Cubic B-spline is **non-interpolating**
  - it is passing near control points, not through them. No guarantee that it is interpolating the control points
- Multiple uniform cubic B-splines curves can be joined together as a spline
  - Cubic B-spline is C^2 continuous
  - Curve composited of multiple B-splines will be C^2 continuous



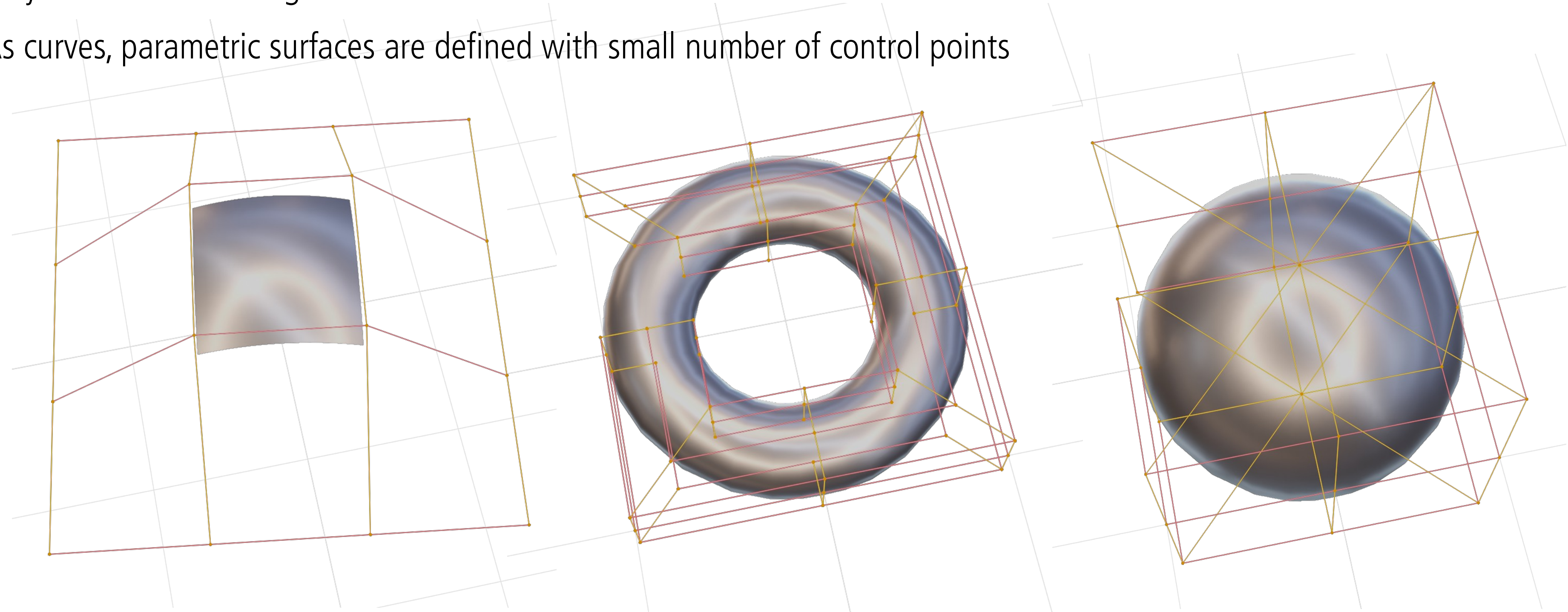
B-spline generalizations

- B-splines explained so far have limitations
- One limitation is that representing circle is not possible
  - Solution, a generalization is to introduce additional coordinate to B-spline
  - **Rational B-spline →** traversal of circles and conic sections is now possible
- Another limitation is requirement of uniformly spaced control points: animation when positions of object are known at $t = 1, 2, 3$ and 10.
  - Futher generalization: **Non-uniform rational B-splines (NURBS)**
  - Very often used in CAD tools

Parametric surfaces

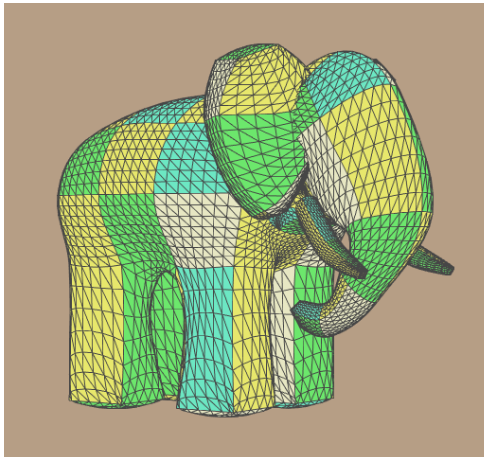
Parametric surfaces

- Natural extension of parametric curves are **parametric surfaces**
  - Similarly as triangle or polygon is extension of a line segment
- Very useful for modeling curved surfaces
- As curves, parametric surfaces are defined with small number of control points



Parametric surfaces

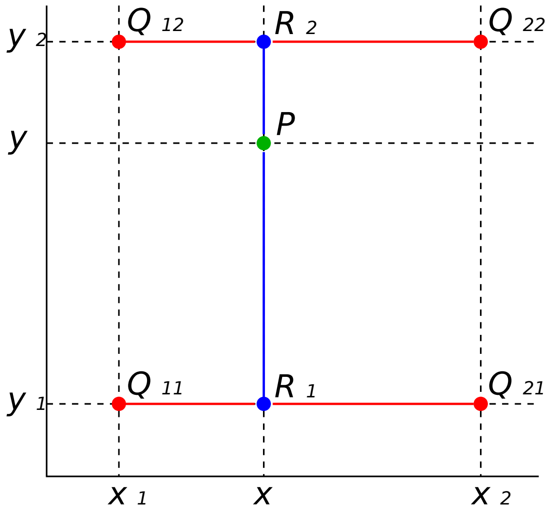
- Common types
  - Bezier surfaces
  - B-spline surfaces



Ed Catmull's "Gumbo" model, composed from Bezier patches

Bezier surface: bilinear interpolation

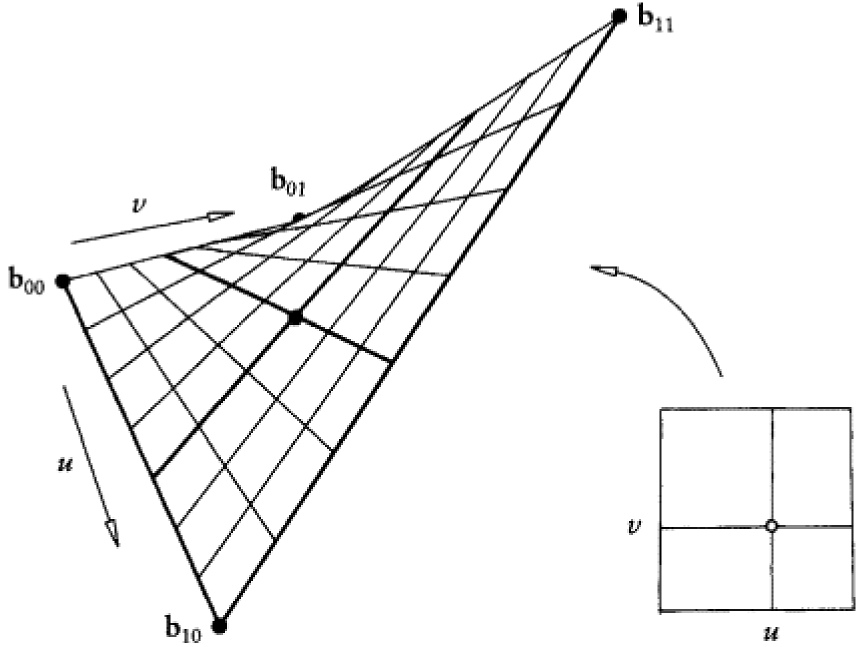
- Bezier curve is extended so it has two parameters (u,v) which define surface.
- Similarly as we started with Bezier curve by explaining linear interpolation, we explain Bezier patch by explaining **bilinear interpolation***.
    - $R_1(u) = lerp(Q_{11}, Q_{12}, u)$
    - $R_2(u) = lerp(Q_{12}, Q_{22}, u)$
    - $P(u,v) = lerp(R_1(u), R_2(u), v)$



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

Bezier surface: patch

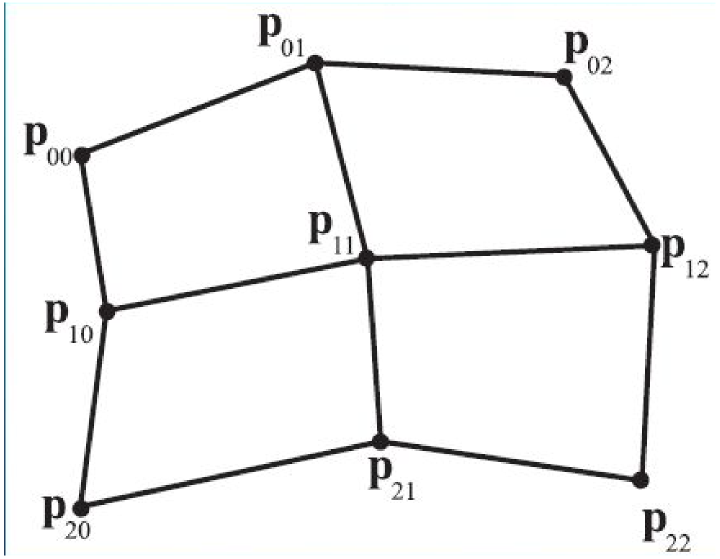
- p(u,v) is simplest, non-planar parametric surface with (u,v) in [0,1]. It has **rectangular domain** and thus resulting surface is called a **patch**.



https://www.sciencedirect.com/topics/computer-science/bilinear-interpolation

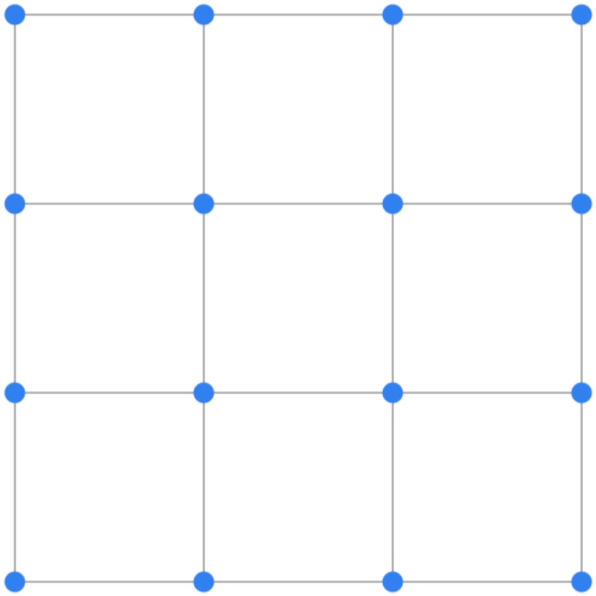
Bezier surface

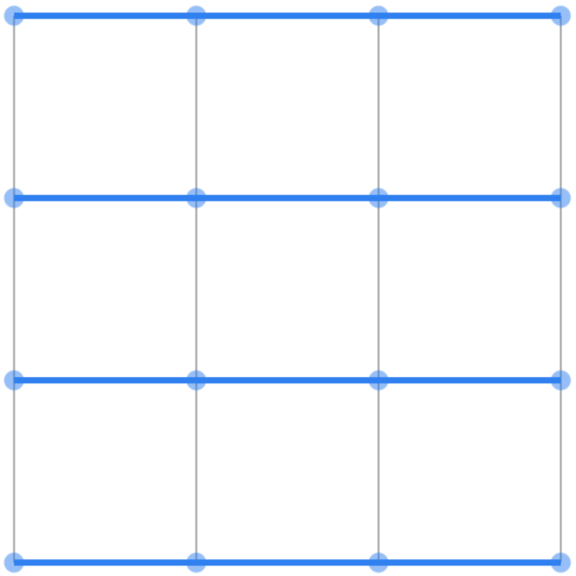
- Similarly as we added more points to linear interpolation to obtain Bezier curve, we add more points to bilinear interpolation to obtain Bezier surface
- Example: biquadratic Bezier surface:
  - 9 control points



Bezier surface: repeated bilinear interpolation

- Repeated bilinear interpolation is extension of de Casteljau's algorithm to patches.
- Starting control points:

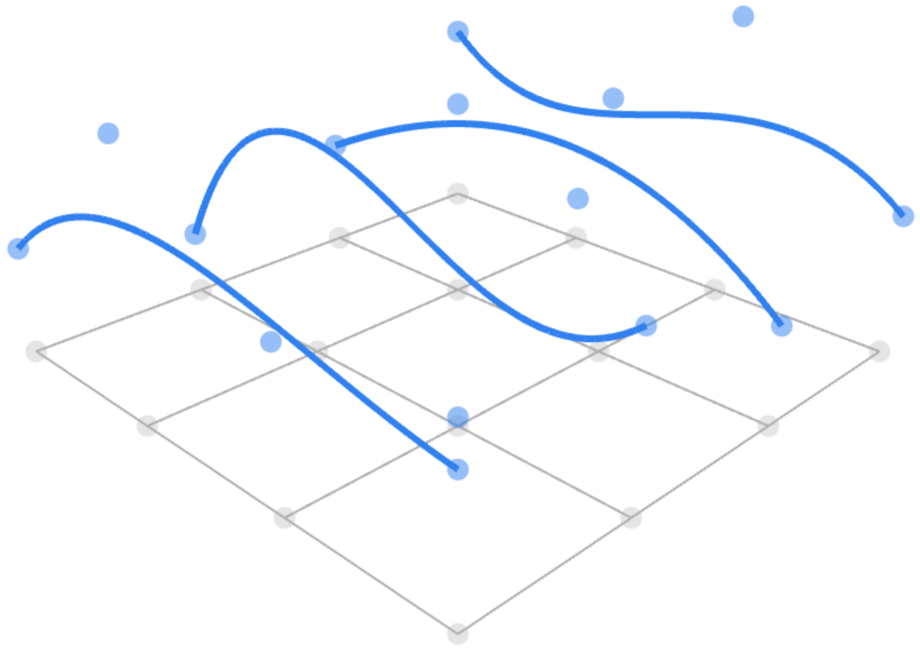




https://acko.net/files/fullfrontal/fullfrontal/wdcode/online.html

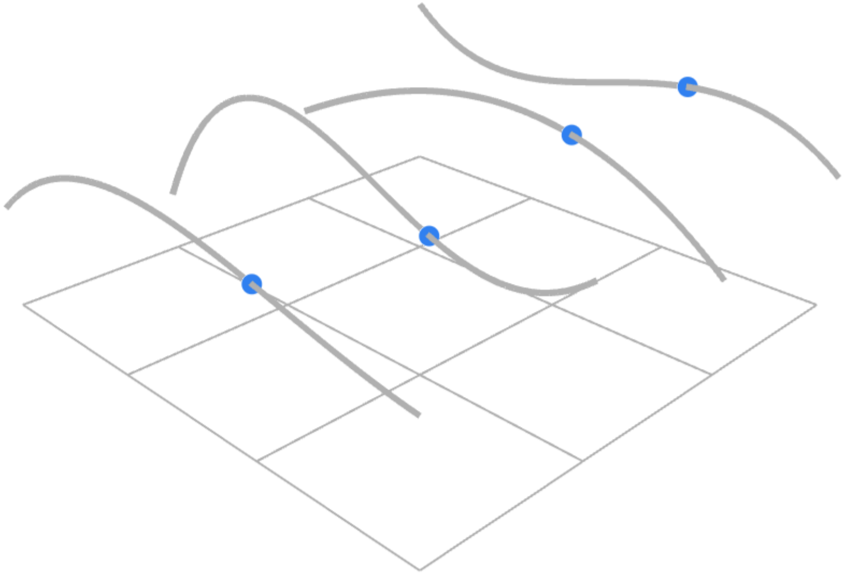
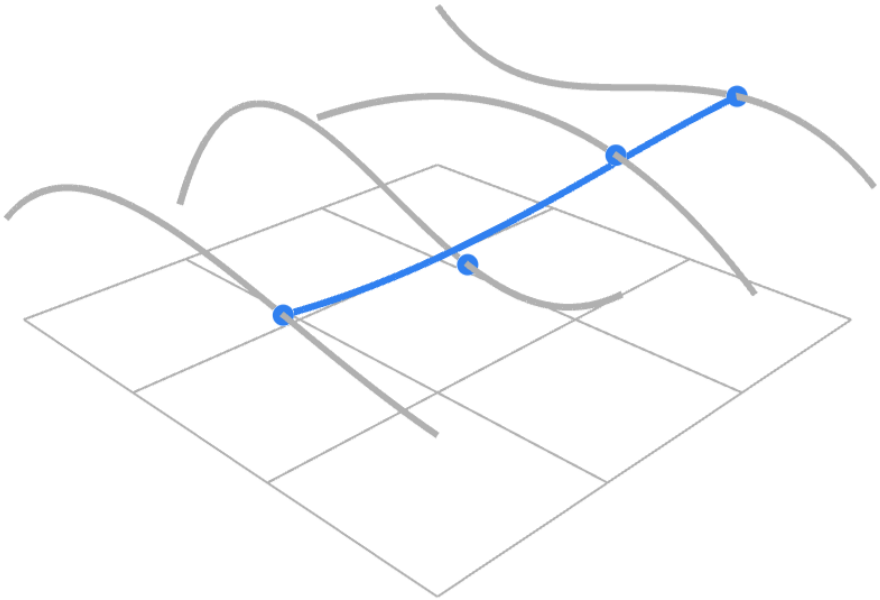
Bezier surface: repeated bilinear interpolation

- Repeated linear interpolation:



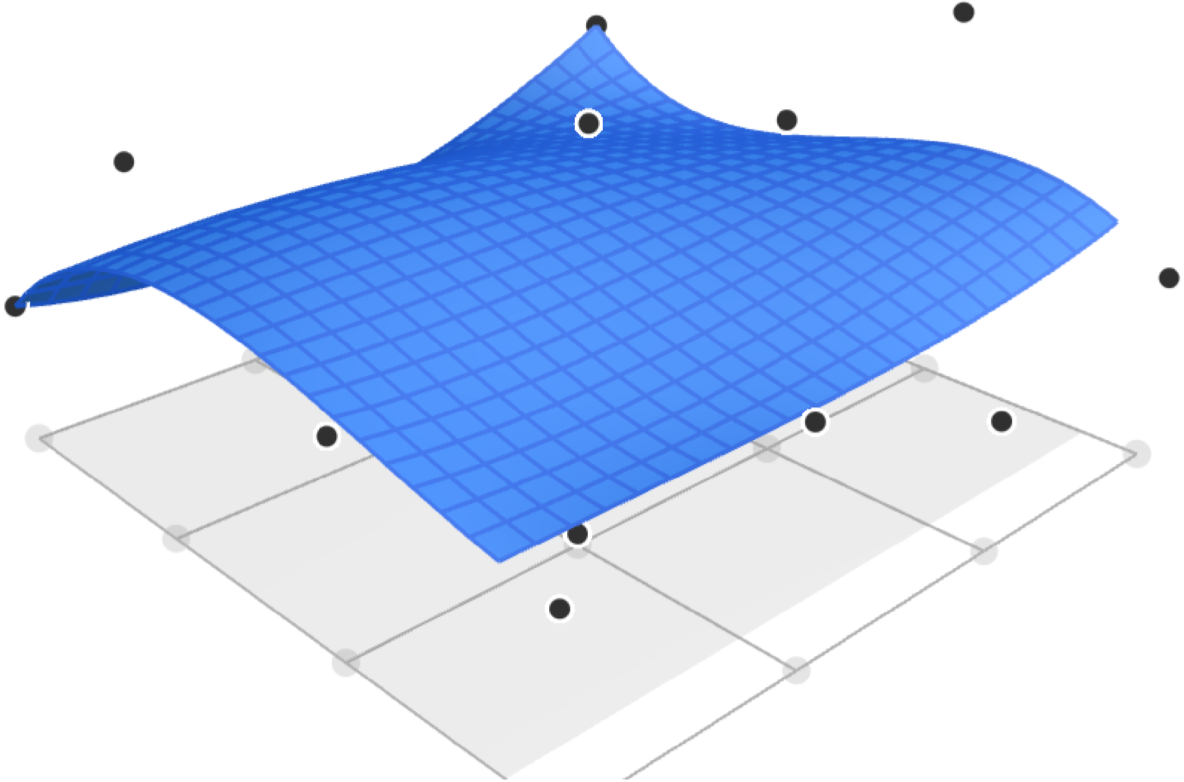
Bezier surface: repeated bilinear interpolation

- Repeated bilinear interpolation

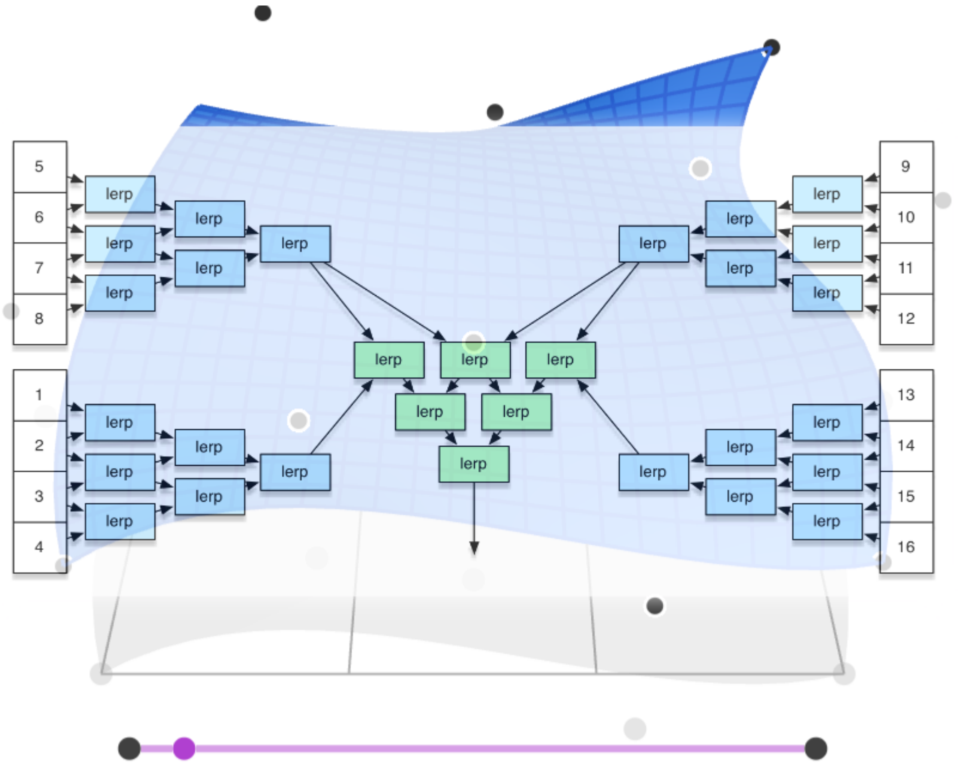
Bezier surface: repeated bilinear interpolation

- Final Bezier surface



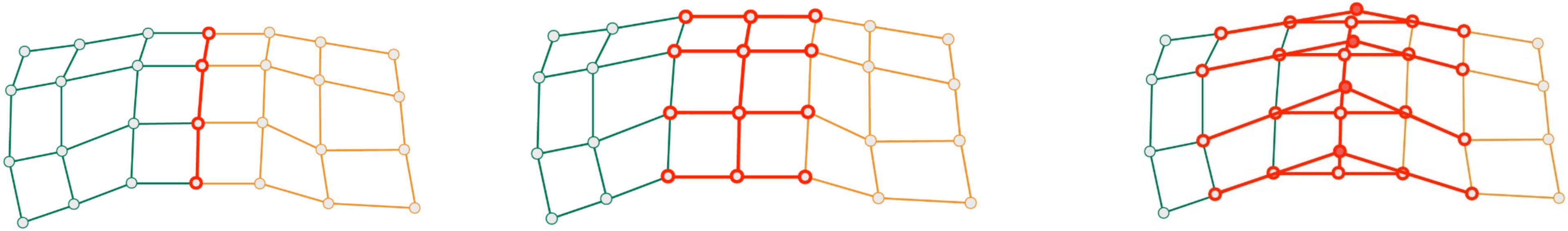
Bezier surface

- Repeated bilinear interpolation is extension of **De Casteljau algorithm** to patches patches
- Point on Bezier Patch can be described in **Bernstein form** using Bernstein polynoms



Bezier surface: continuity

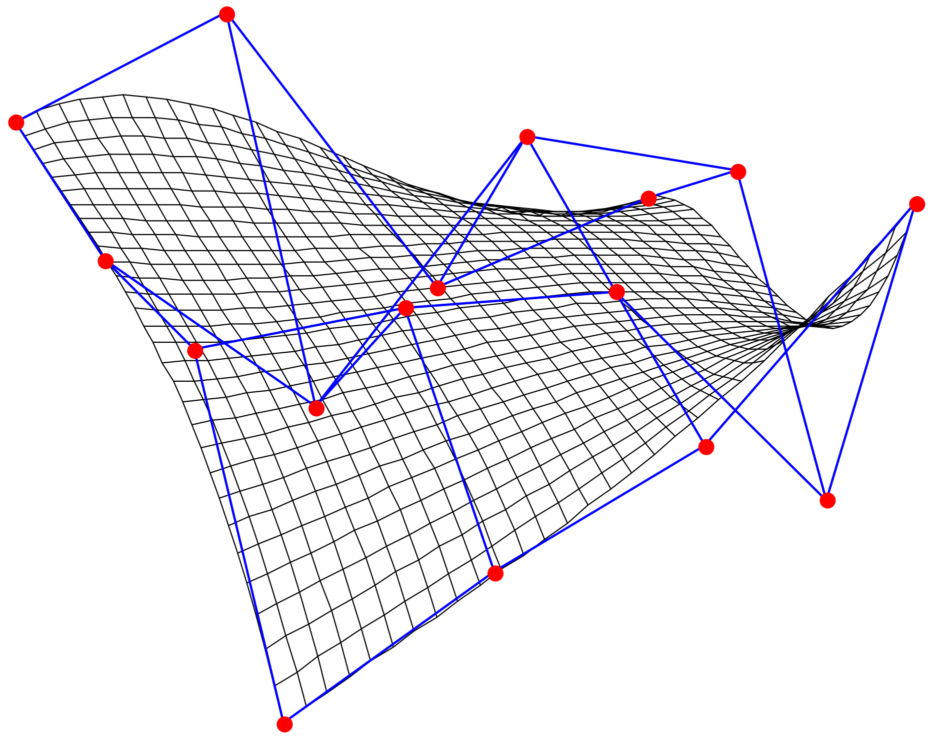
- C^0 – **positional continuity** – joined at the same point
- C^1 – **velocity continuity** - derivation of any point (including joints) must be continuous
- C^2 - **acceleration continuity** - first and second derivatives are continuous functions



https://cs184.eecs.berkeley.edu/sp22/lecture/7-96/intro-to-geometry-splines-and-be

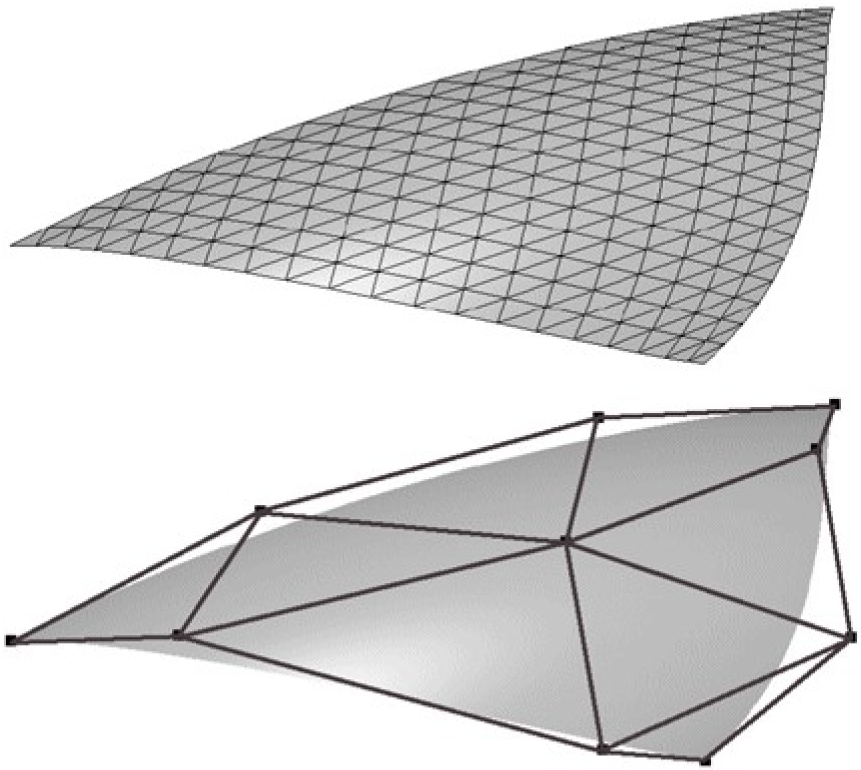
Bezier surface properties

- 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 (derivation of polynomials)
- More on the topic: rational Bezier patches



Other parametrized surfaces

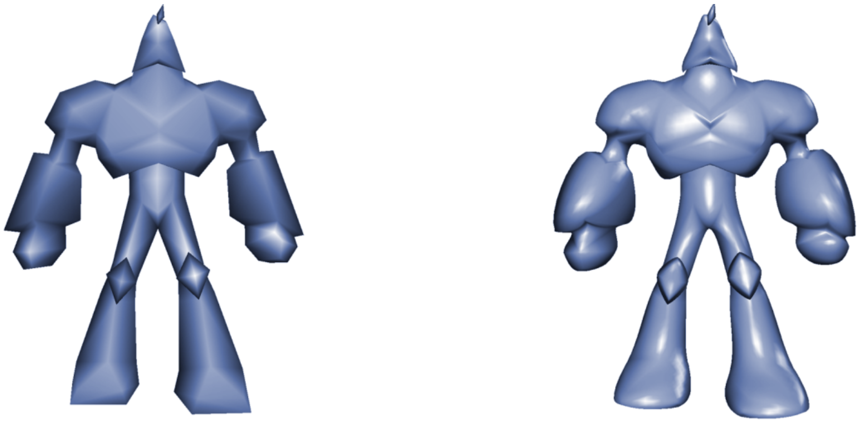
- **Bezier triangles**
  - Control points are located in triangulated grid
  - Based on repeated interpolation: de Casteljau, Bernstein triangles
  - Constructing complex object requires stitching Bezier triangles so that composite surface contains desired properties and look: continuity



Real-time rendering book, ch 17.2.

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 using Bezier triangles*
  - 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

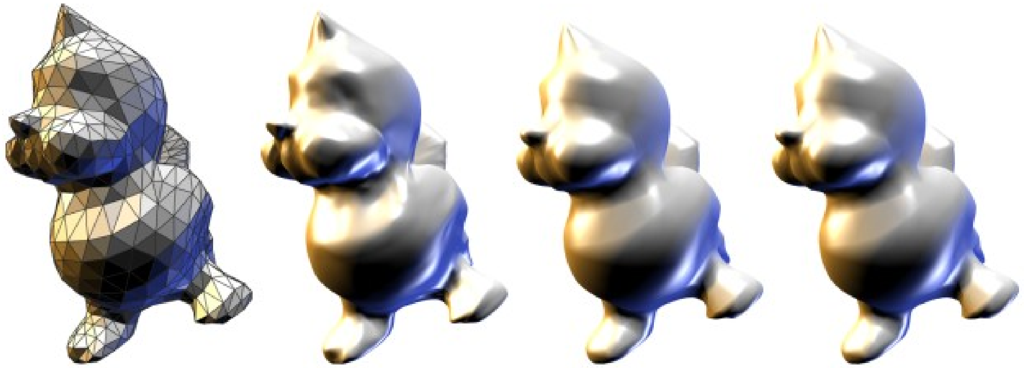


https://alex.vlachos.com/graphics/CurvedPNTriangles.pdf

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

Other parametrized surfaces

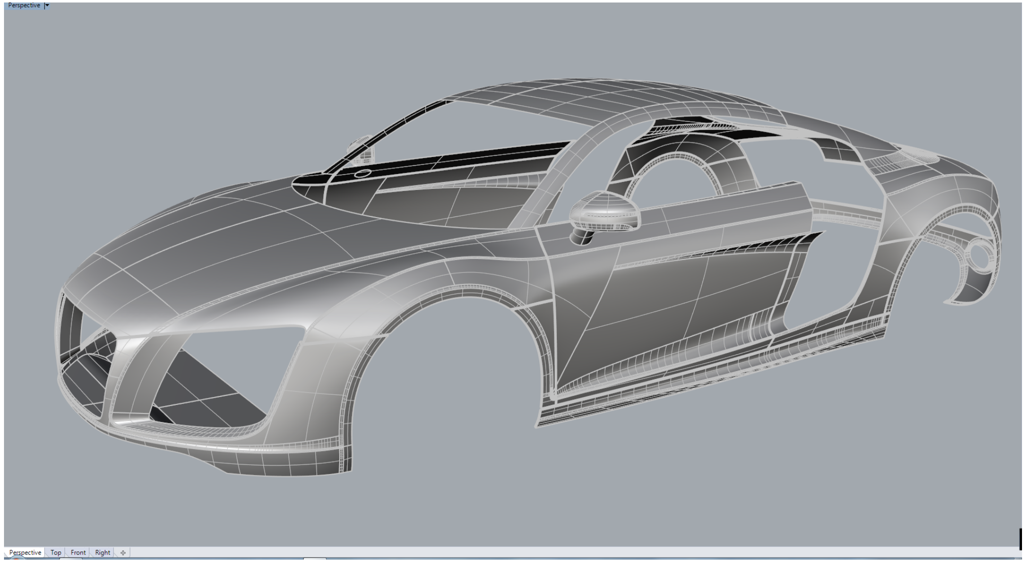
- **Phong tessellation**
  - Similar as PN triangles, given the triangle points with normals, construct “better-looking” surface
  - Phong tessallation attempts to create geometric version of Phong shading normal using repeated interpolation resulting in Bezier triangles.



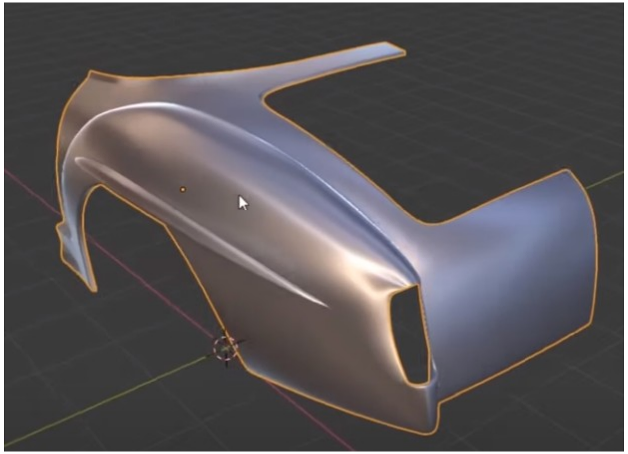
Mesh, Butterfly subidivision, PN Triangles, Phong Tessellation
https://perso.telecom-paristech.fr/boubek/papers/PhongTessellation/

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 (will be discussed later)
- **Non-uniform rational B-Spline surface (NURBS)** is often used in 3D modeling software



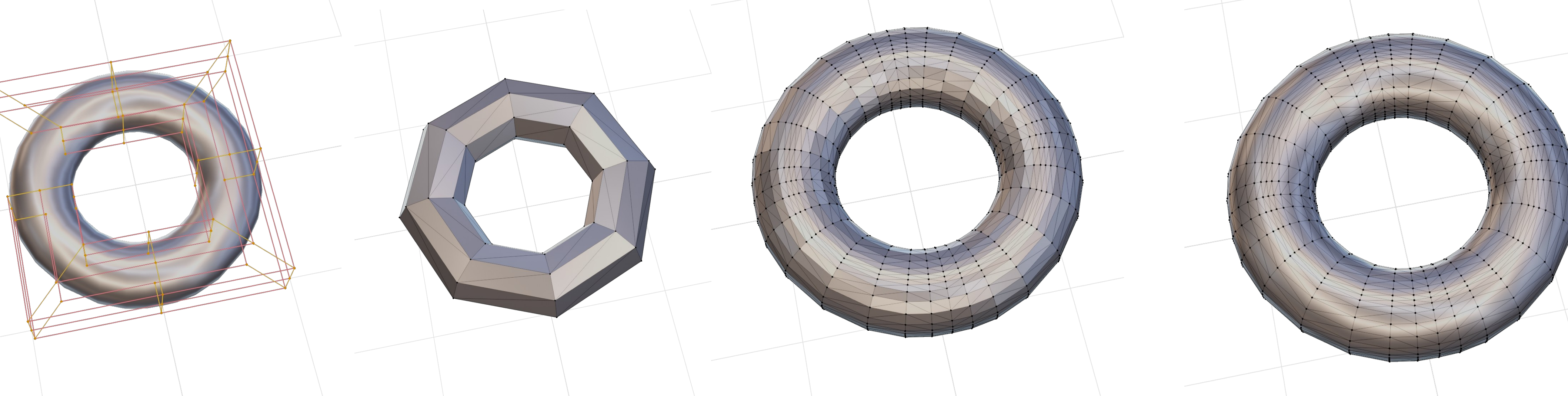
https://forums.cgsociety.org/t/audi-r8-v10-2010-nurbs-model-with-rhino-3d/1761843/3



https://www.youtube.com/watch?v=1zNfh_g8jXM&ab_channel=BlenderSecrets

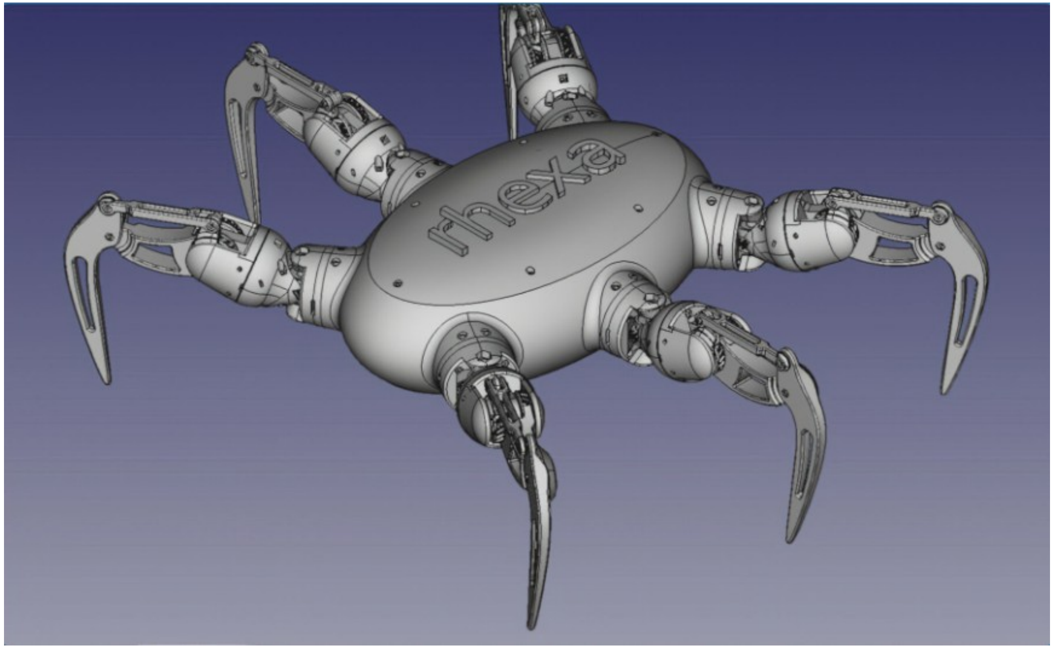
Rendering of curved surfaces

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



Modeling and transferring parametric surfaces

- Parametric surfaces are stored using:
  - STEP file format
  - IGES file format
- CAD programs are used to create and edit parametric surfaces.



https://www.freecadweb.org/

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