

Lecture 6: Parametric curves and surfaces, implicit shapes

DHBW, Computer Graphics

Lovro Bosnar

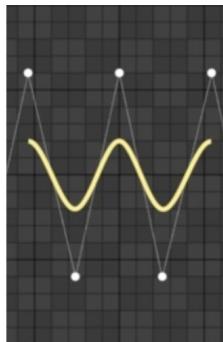
8.2.2023.

Syllabus

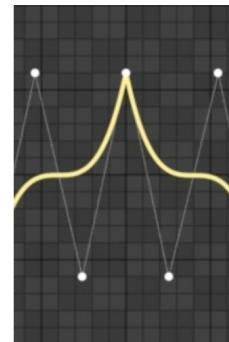
- 3D scene
 - Objects
 - Shape
 - Material
 - Lights
 - Cameras
 - Rendering
 - Image and display
-
- A blue arrow points from the 'Shape' bullet point in the first list to a rounded rectangular callout box. The callout box contains three bullet points: 'Parametric curves', 'Parametric surfaces', and 'Implicit surfaces'. The 'Shape' bullet point is also enclosed in a small blue rectangle.
- Parametric curves
 - Parametric surfaces
 - Implicit surfaces

Recap: shape representations

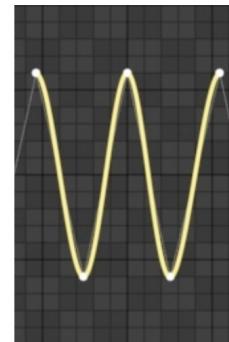
- Points
 - Point clouds
 - Particles and Particle systems
- Lines
 - Parametric curves
- Surfaces:
 - Polygonal mesh
 - Subdivision surfaces
 - **Parametric surfaces**
 - **Implicit surfaces**
- Volumetric objects/solids
 - Voxels
 - Space partitioning data-structures



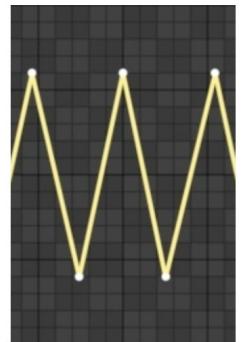
B-Spline



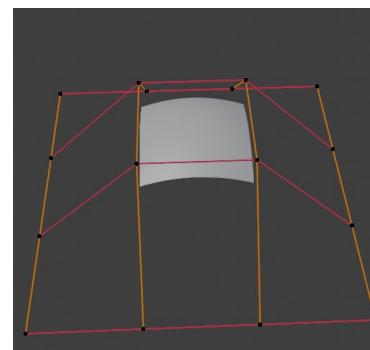
Bezier



Catmull-Rom



Linear



NURBS

Recap: shape representations

- Representing surface using mesh is the most used and widespread option for authoring, transfer and rendering.
 - Triangle mesh and thus triangle is basic atomic rendering primitive for GPU graphics pipelines and most ray-tracers.
 - All higher-level geometrical descriptions are evaluated as set of triangles and then used.
- 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.

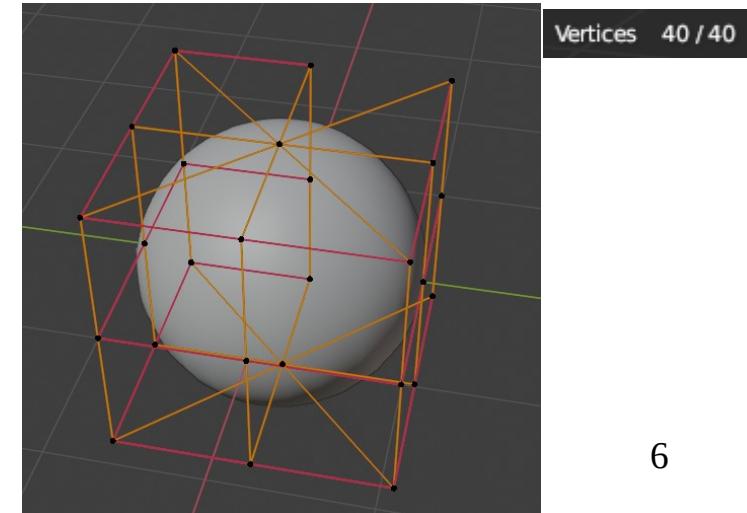
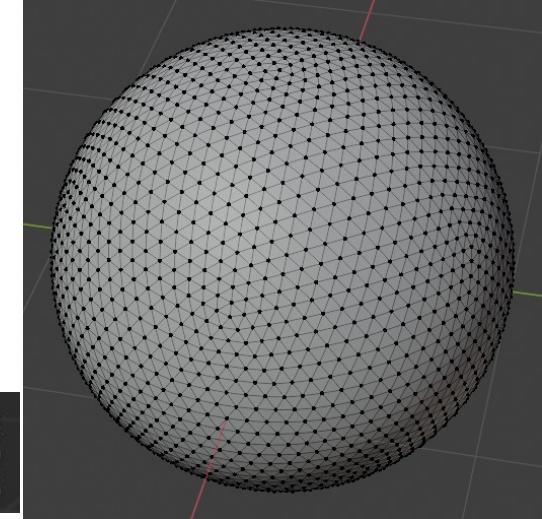
<IMAGE: show THE CONCEPT OF USER AND RENDERING SIDE AND HOW OBJECTS CAN BE REPRESENTED>

Parametric surfaces vs Mesh

Parametric surfaces have advantages over meshes in certain scenarios:

- More **compact representation** than meshes:
 - Less memory requirements
 - Less transformation operations
- **Scalable geometric primitives**
 - Geometry can be generated on the fly by evaluating the equations
- They can represent **smoother and more continuous primitives** than polygons
 - Convenient for representing objects like hair, organic and curved objects
- Certain modeling tasks can be efficiently performed, e.g., animation and collision

Vertices	2,562 / 2,562
Edges	7,680 / 7,680
Faces	5,120 / 5,120



Applications

- Manufacturing
- Product design
- architecture
- Engineering
- Nature modeling
- Animation
- Vector graphics



Paul de Casteljau



1959



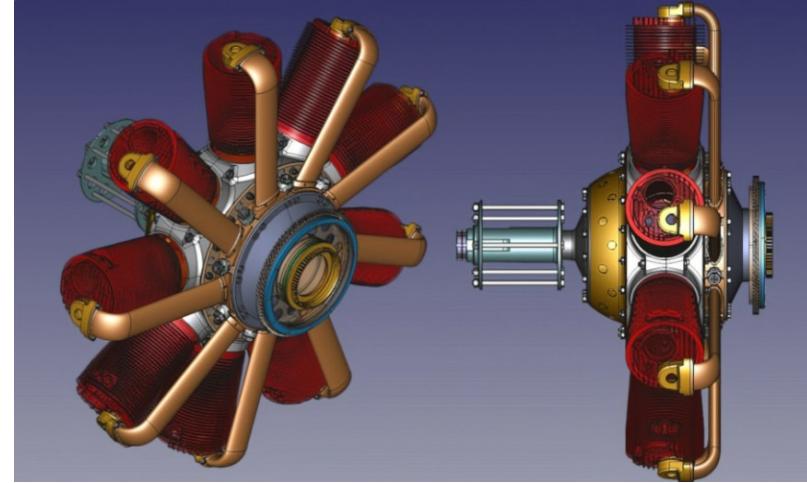
Pierre Bézier



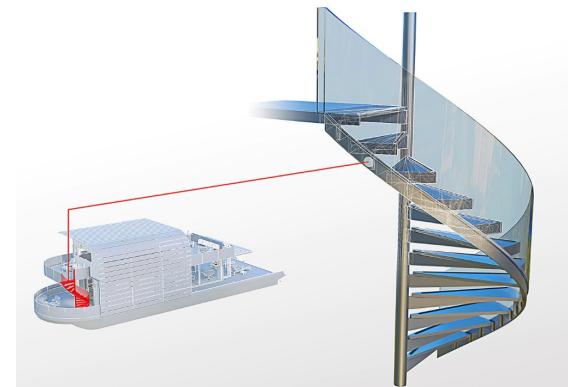
1962



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

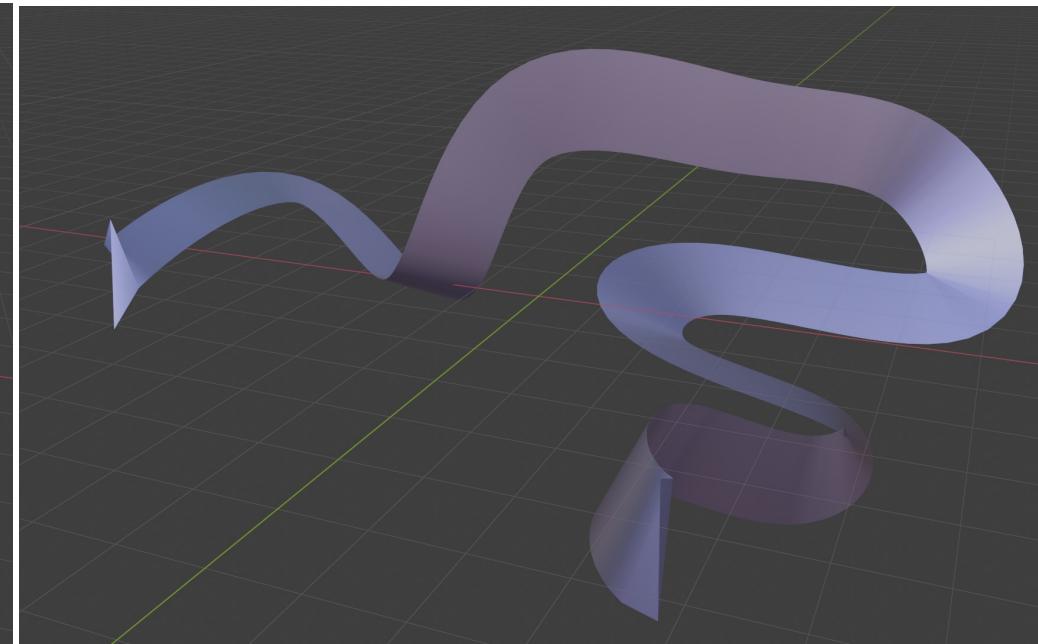
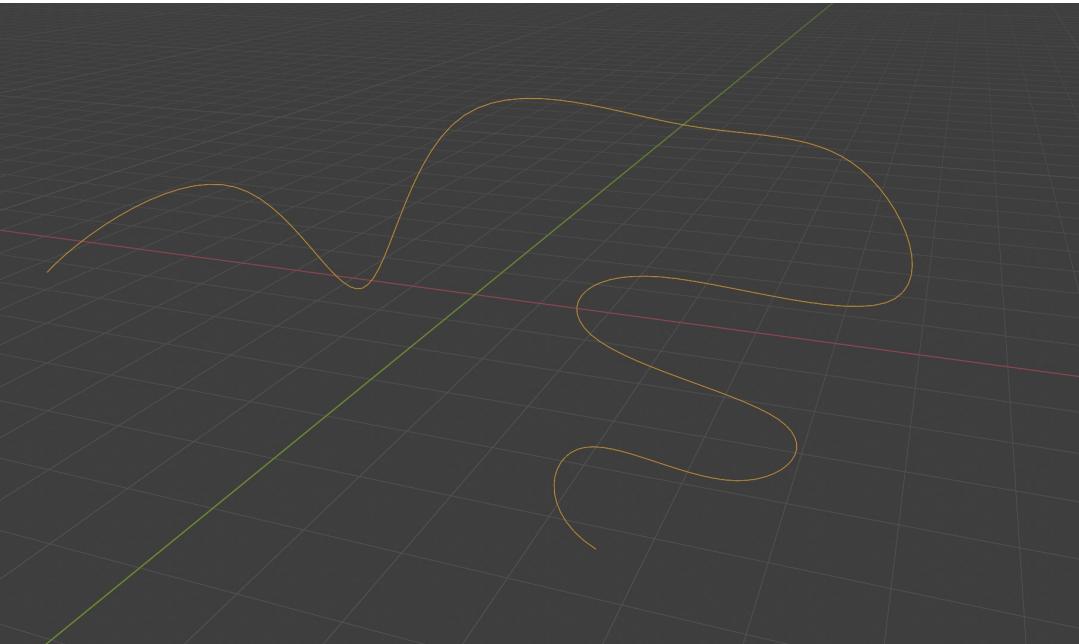


<https://www.freecadweb.org/>



Parametric curves and surfaces

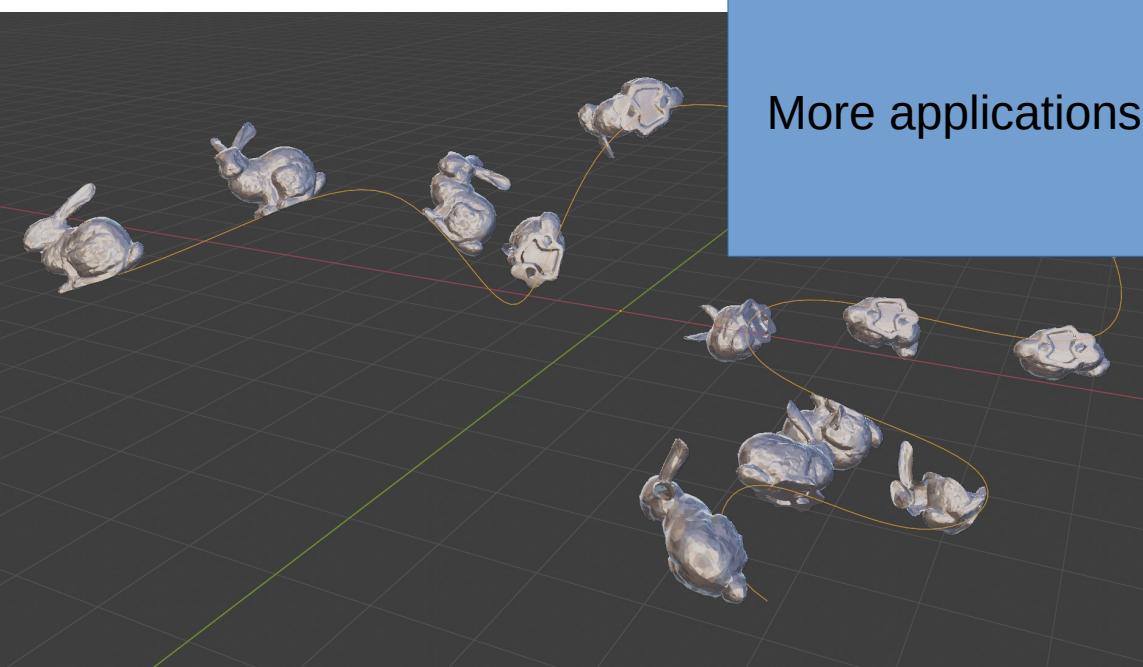
- To understand parametric surfaces, we will start with parametric curves



Parametric curves

Parametric curves

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

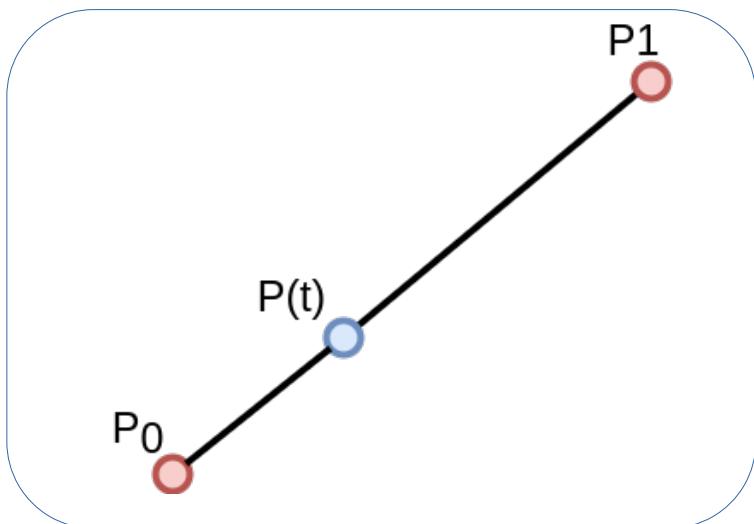


More applications



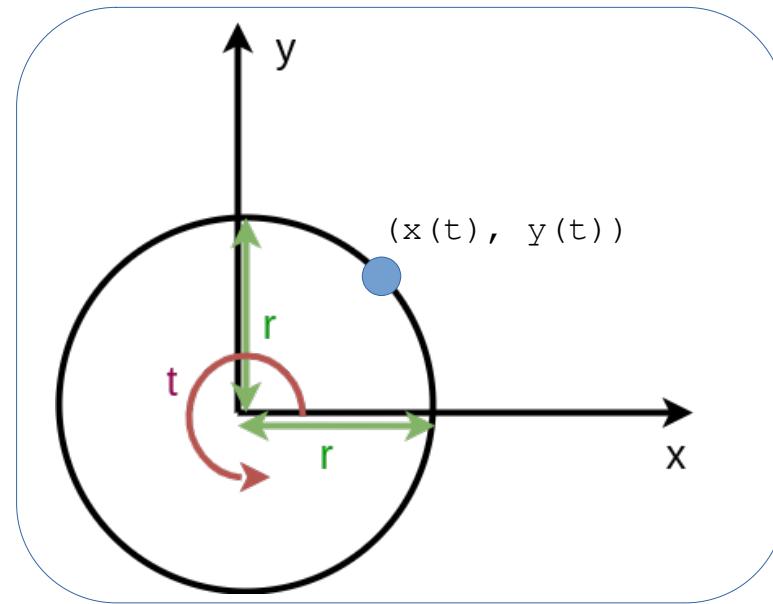
Parametric curves

- 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**

$$x(t) = r \cos(2 * \pi * t), t \text{ in } [0, 1]$$

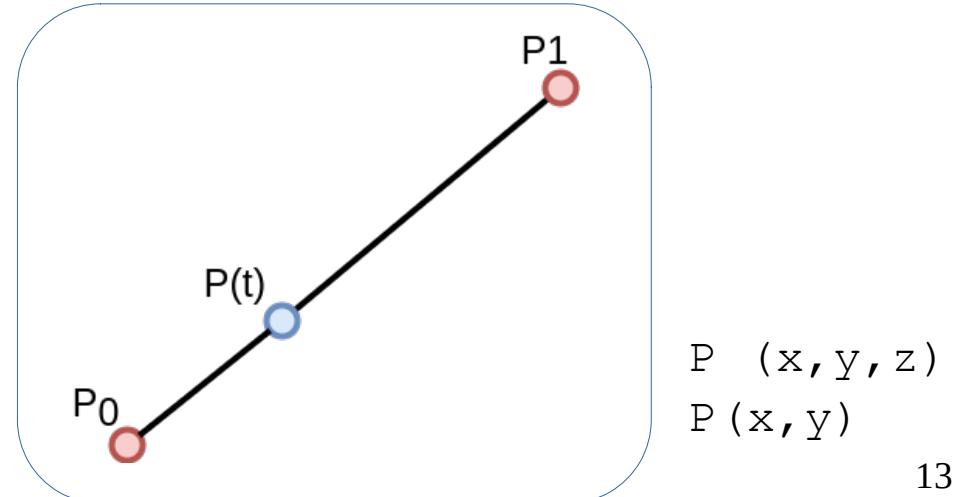
$$y(t) = r \sin(2 * \pi * t), t \text{ in } [0, 1]$$

Arbitrary curves?

- Bezier curve
- Hermite curve
- Catmull-Rom spline
- B-Splines

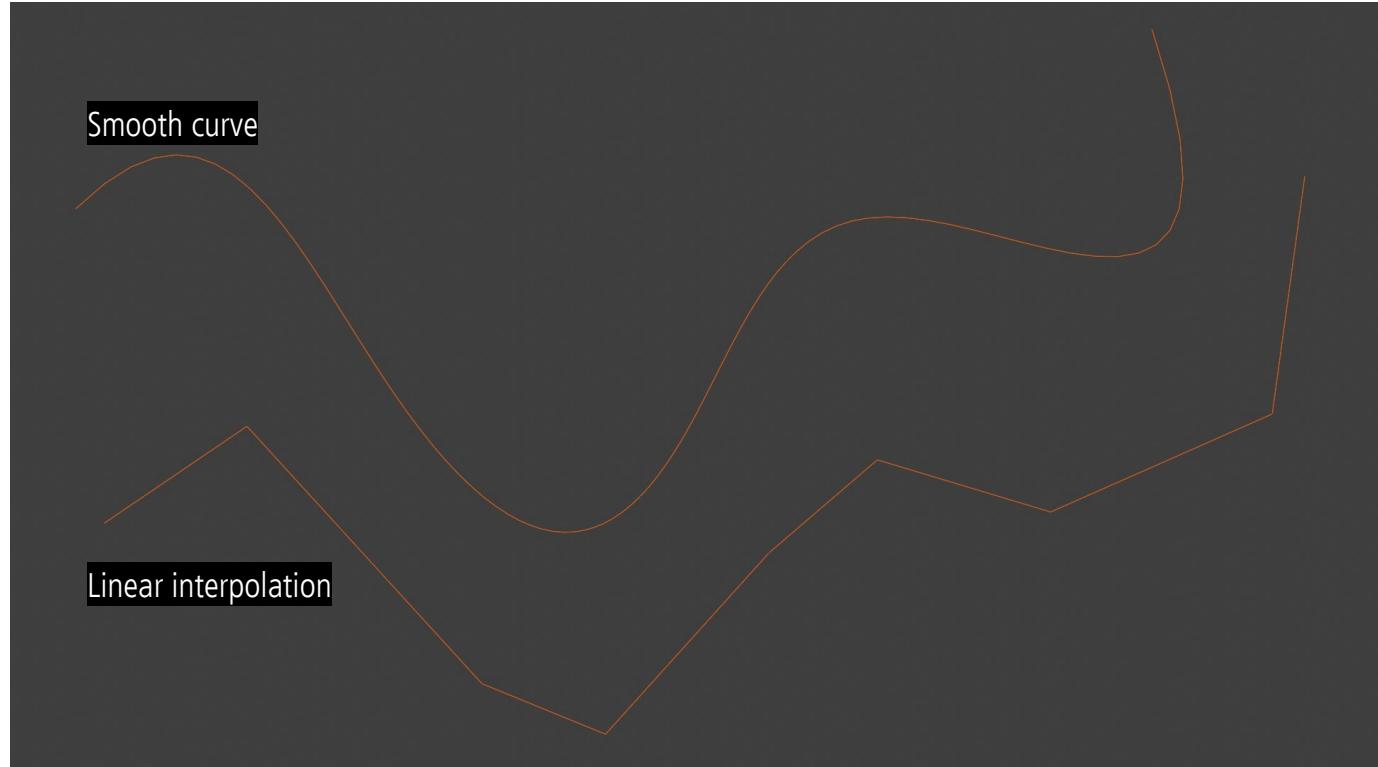
Bezier curves: linear interpolation

- Linear interpolation between two control points P_0 and P_1 traces out straight line.
 - $P(t) = (1-t) P_0 + t P_1$
 - Short: `lerp(P_0 , P_1 , t)`
 - For $0 < t < 1$, generated points are on straight line between P_0 and P_1 . Otherwise outside.
 - $P(0) = P_0$ and $P(1) = P_1$



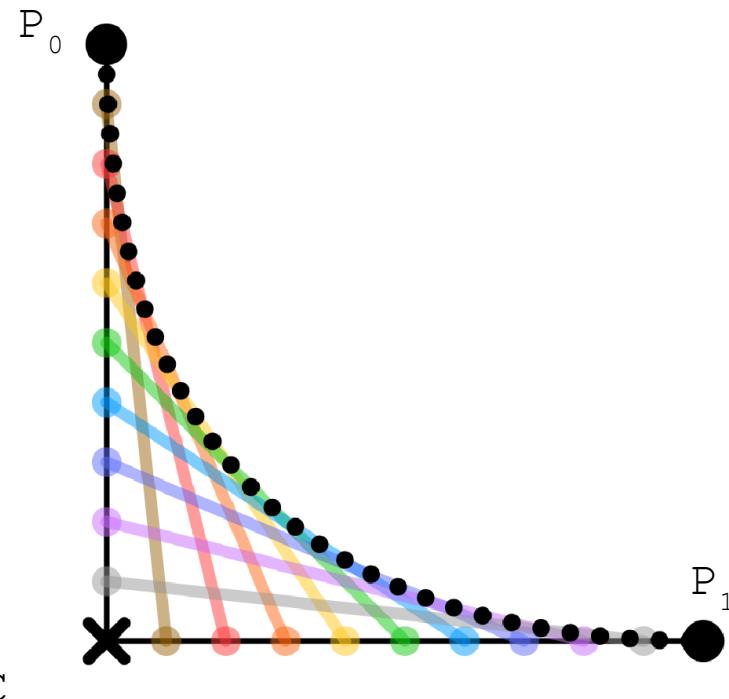
Bezier curves: linear interpolation

- Linear interpolation between multiple points gives us straight segments with sudden (discontinuous) changes at joints between.



Bezier curves: repeated interpolation

- Instead of linear interpolation between two points: P_0 and P_1 – **end points**, add another point C – **control point** and use it for **repeated interpolation**



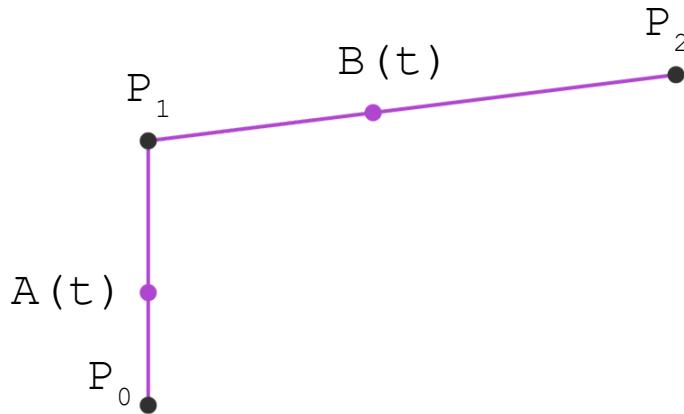
* Independently discovered by Paul de Casteljau and Pierre Bezier for use in French car industry.

Bezier curves: repeated interpolation

Linear interpolation



$$A(t) = \text{lerp}(P_0, P_1, t)$$

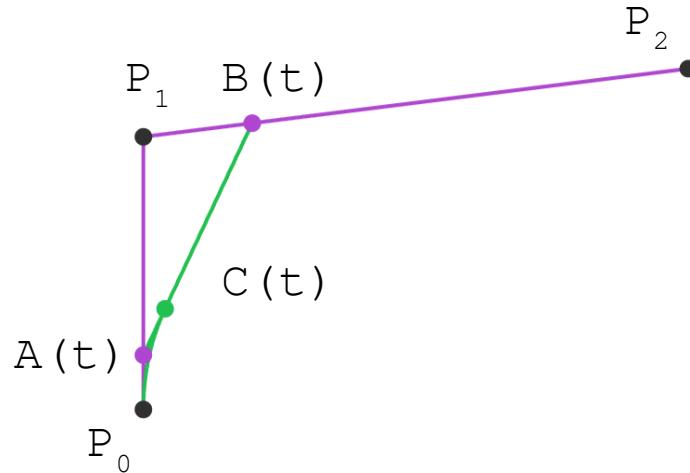


$$A(t) = \text{lerp}(P_0, P_1, t)$$

$$B(t) = \text{lerp}(P_1, P_2, t)$$

Quadratic Bezier curve

- Repeated interpolation:
 - $A(t)$: linear interpolation between P_0 and P_1
 - $B(t)$: linear interpolation between P_0 and P_1
 - $C(t)$: linear interpolation between $A(t)$ and $B(t)$



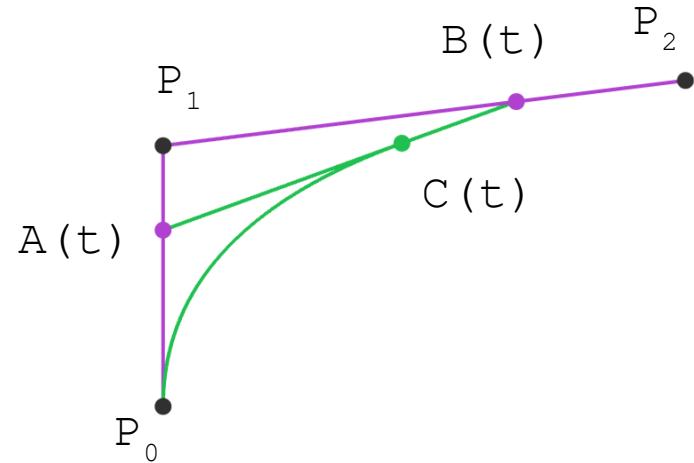
$$A(t) = \text{lerp}(P_0, P_1, t)$$

$$B(t) = \text{lerp}(P_1, P_2, t)$$

$$C(t) = \text{lerp}(A(t), B(t), t)$$

Quadratic Bezier curve

- Repeated interpolation:
 - $A(t)$: linear interpolation between P_0 and P_1
 - $B(t)$: linear interpolation between P_0 and P_2
 - $C(t)$: linear interpolation between $A(t)$ and $B(t)$



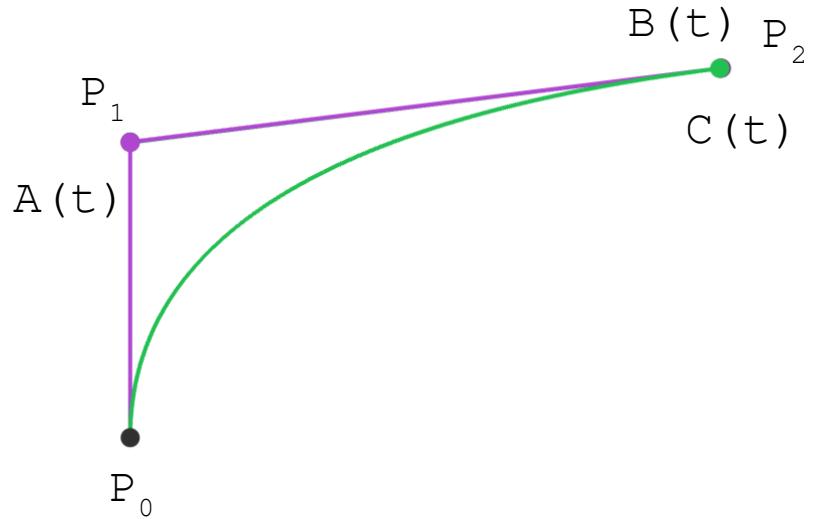
$$A(t) = \text{lerp}(P_0, P_1, t)$$

$$B(t) = \text{lerp}(P_1, P_2, t)$$

$$C(t) = \text{lerp}(A(t), B(t), t)$$

Quadratic Bezier curve

- Repeated interpolation:
 - $A(t)$: linear interpolation between P_0 and P_1
 - $B(t)$: linear interpolation between P_0 and P_2
 - $C(t)$: linear interpolation between $A(t)$ and $B(t)$

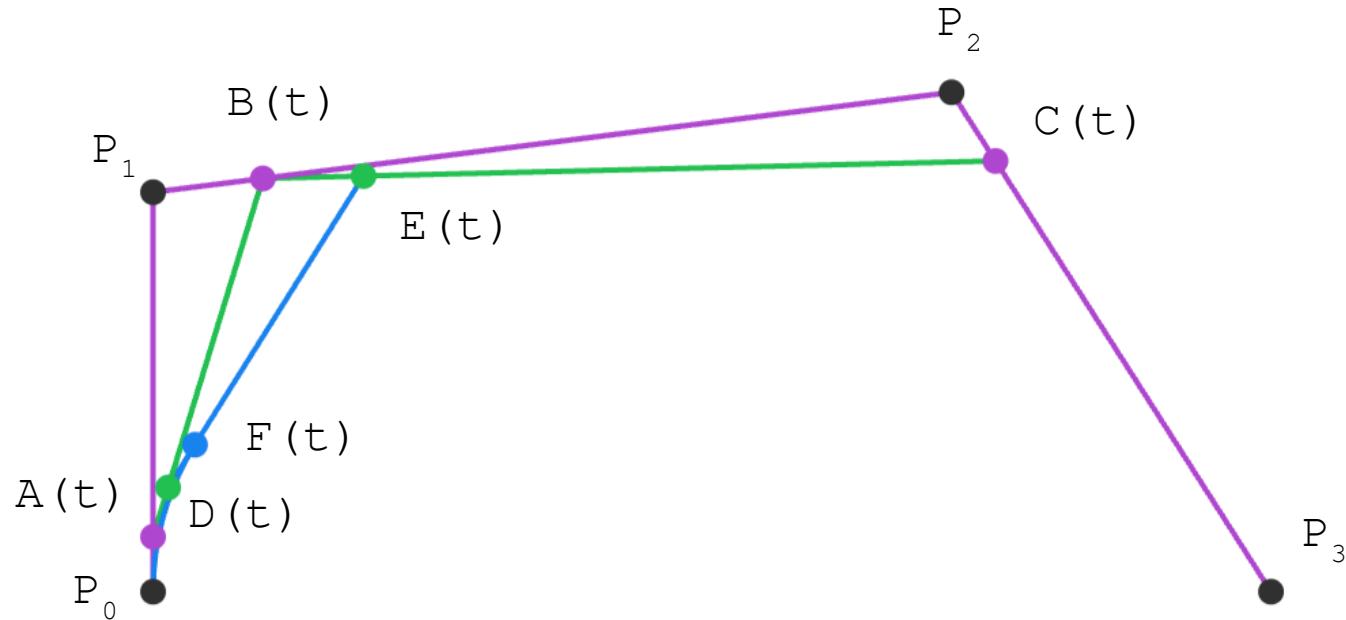


$$A(t) = \text{lerp}(P_0, P_1, t)$$

$$B(t) = \text{lerp}(P_1, P_2, t)$$

$$C(t) = \text{lerp}(A(t), B(t), t)$$

Cubic Bezier curve



$$A(t) = \text{lerp}(P_0, P_1, t)$$

$$B(t) = \text{lerp}(P_1, P_2, t)$$

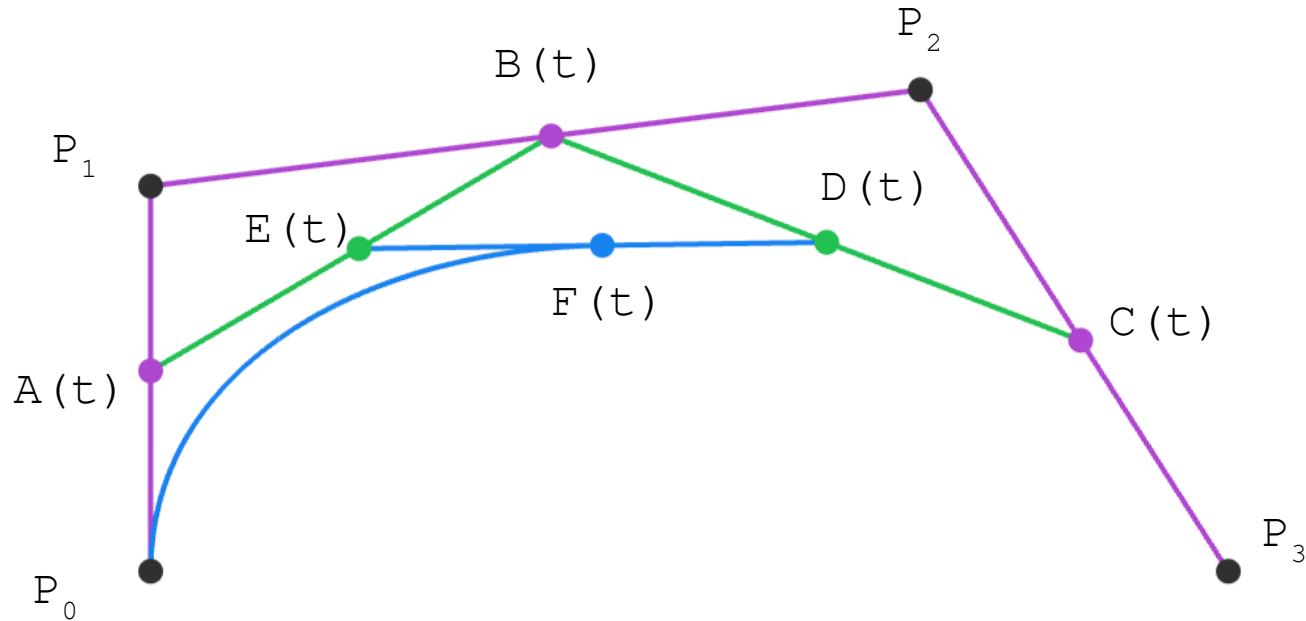
$$C(t) = \text{lerp}(P_2, P_3, t)$$

$$D(t) = \text{lerp}(A(t), B(t), t)$$

$$E(t) = \text{lerp}(B(t), C(t), t)$$

$$C(t) = \text{lerp}(D(t), E(t), t)$$

Cubic Bezier curve



$$A(t) = \text{lerp}(P_0, P_1, t)$$

$$B(t) = \text{lerp}(P_1, P_2, t)$$

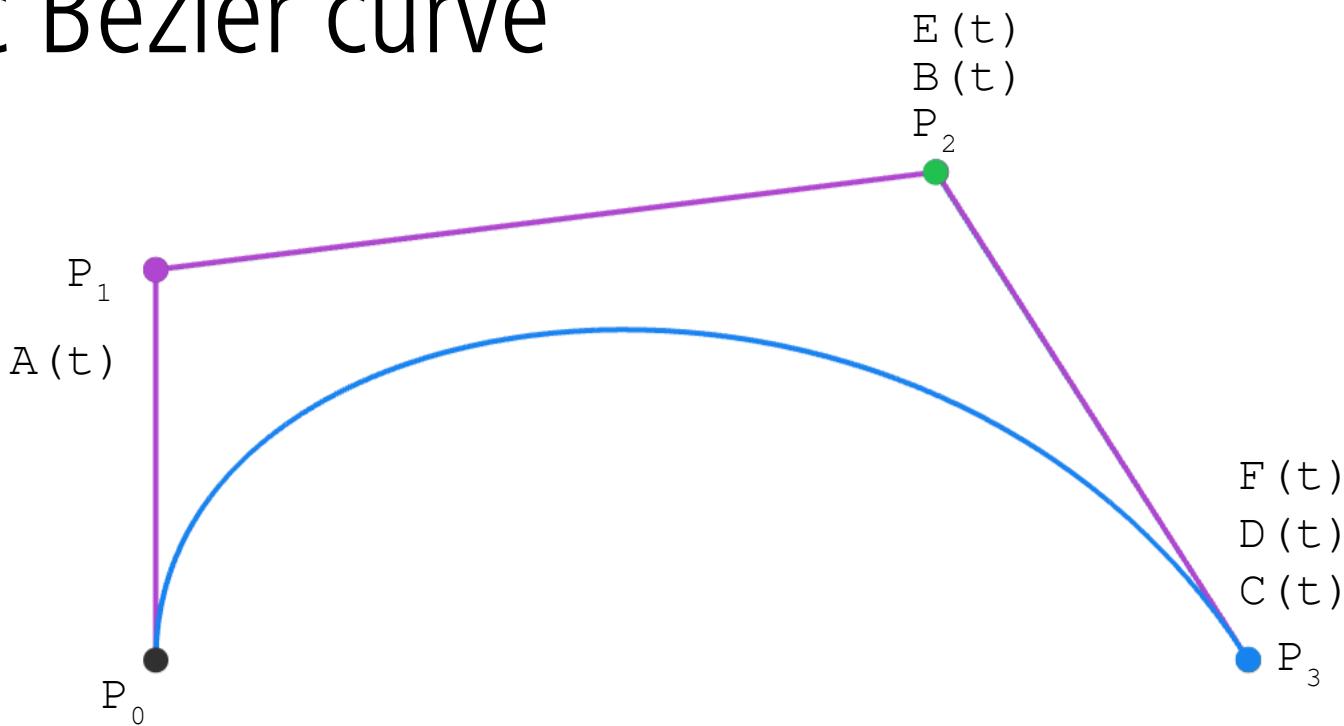
$$C(t) = \text{lerp}(P_2, P_3, t)$$

$$D(t) = \text{lerp}(A(t), B(t), t)$$

$$E(t) = \text{lerp}(B(t), C(t), t)$$

$$C(t) = \text{lerp}(D(t), E(t), t)$$

Cubic Bezier curve



$$A(t) = \text{lerp}(P_0, P_1, t)$$

$$B(t) = \text{lerp}(P_1, P_2, t)$$

$$C(t) = \text{lerp}(P_2, P_3, t)$$

$$D(t) = \text{lerp}(A(t), B(t), t)$$

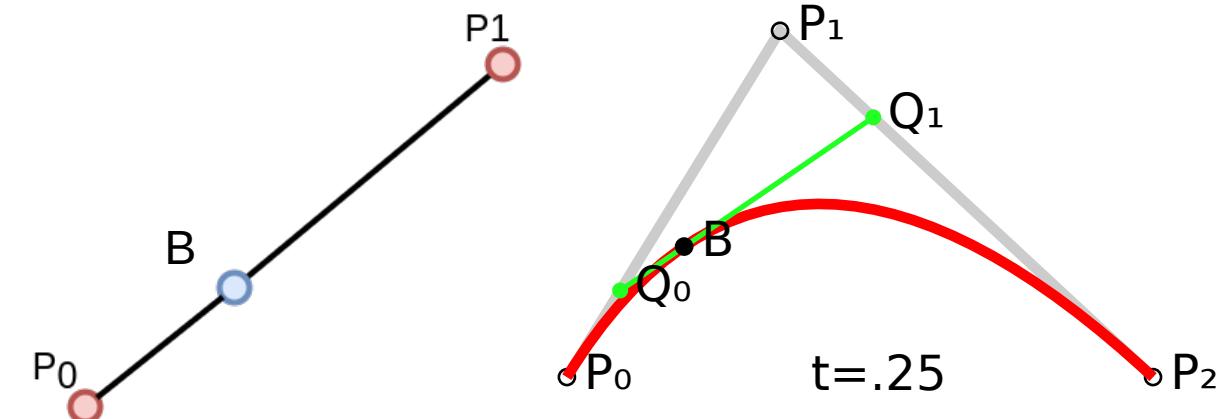
$$E(t) = \text{lerp}(B(t), C(t), t)$$

$$C(t) = \text{lerp}(D(t), E(t), t)$$

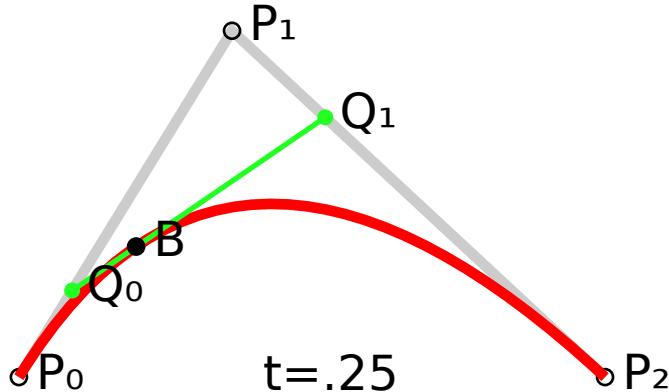
Bezier curves

- Degree of curve is n , if $n + 1$ control points are used.
 - More control points → more degrees of freedom
- Most often, cubic Bezier curve is used
 - Higher degree Bezier curves are expensive to evaluate

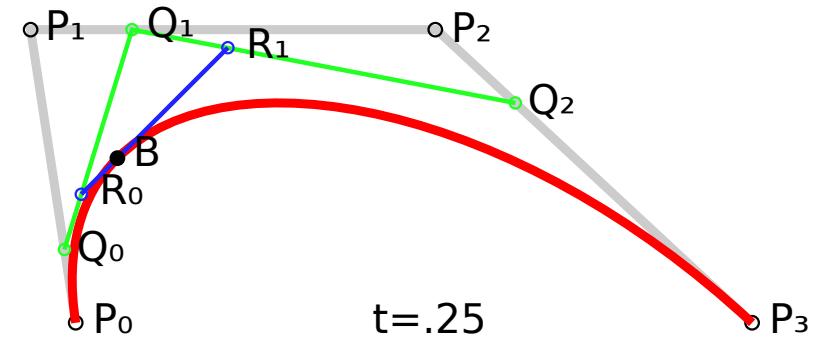
$n = 1 \rightarrow 1^{\text{st}}$ degree, linear Bezier



$n = 2 \rightarrow 2^{\text{nd}}$ degree, quadratic Bezier



$n = 3 \rightarrow 3^{\text{rd}}$ degree, cubic Bezier



Bezier curves: de Casteljau algorithm

- Repeated or recursive linear interpolation is often referred as **de Casteljau algorithm**.

Linear Bezier:

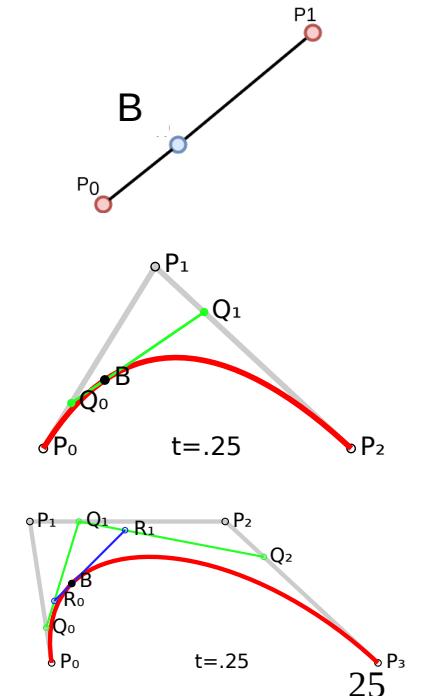
$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1, \quad 0 \leq t \leq 1$$

Quadratic Bezier:

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

Cubic Bezier:

$$\mathbf{B}(t) = (1-t)\mathbf{B}_{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2}(t) + t\mathbf{B}_{\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3}(t), \quad 0 \leq t \leq 1.$$

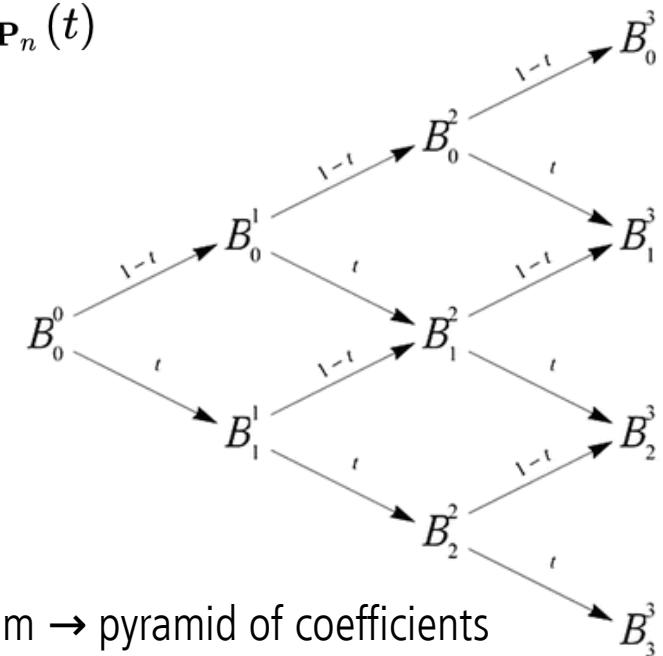


De Casteljau algorithm: recursive formula*

- Recursive formula for arbitrary Bezier curve degree:

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

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



De Casteljau algorithm → pyramid of coefficients

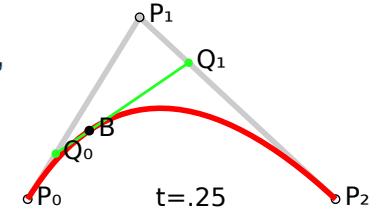
Bezier curve: Bernstein form

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

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

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

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



Bezier curve: Bernstein form

- Bernstein form:

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

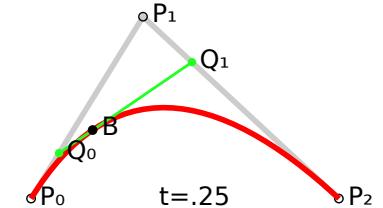
- Bernstein polynomials aka Bezier basis functions:

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

Bezier curve: Bernstein form

- Example: $n = 2$ (quadratic)

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



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

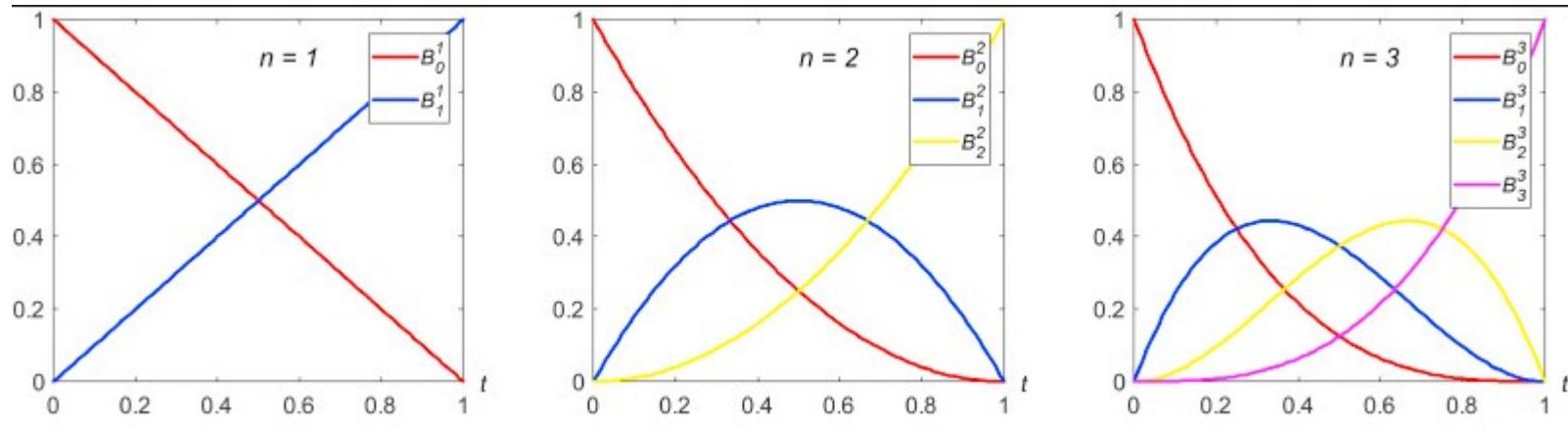
$$\mathbf{B}(t) = (1-t)^2 \mathbf{P}_0 + 2(1-t)t \mathbf{P}_1 + t^2 \mathbf{P}_2, \quad 0 \leq t \leq 1$$

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

Bernstein polynomials

- When t increases, blending weight for P_0 decreases and blending weight for P_1 increases, etc.

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



Bernstein polynomials aka blending functions

Bernstein polynomials

- Bernstein polynomials (Bezier basis function) defines properties of a Bezier curve:

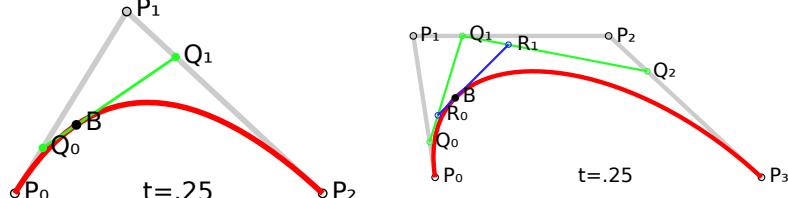
$$b_{i,n}(t) \in [0, 1] \text{ when } t \in [0, 1]$$

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

"Polynomials are in $[0, 1]$ when t in $[0, 1]$ "

"Curve will stay close to the control points P_i "

- Whole Bezier curve will be located in convex hull of control points
 - useful for computing bounding area or volume of curve.

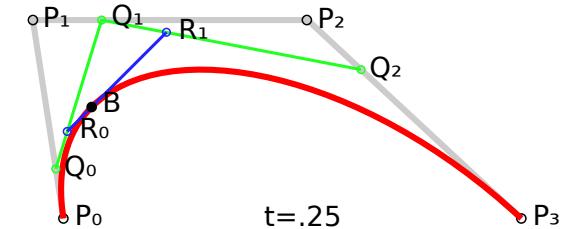


Bezier curve: matrix representation

- In practice, for efficient calculation, Bezier curve is represented in a **matrix form**
- Example: **cubic Bezier curve**

$$\mathbf{B}(t) = (1 - t)^3 \mathbf{P}_0 + 3(1 - t)^2 t \mathbf{P}_1 + 3(1 - t)t^2 \mathbf{P}_2 + t^3 \mathbf{P}_3, \quad 0 \leq t \leq 1$$

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



Check matrix

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

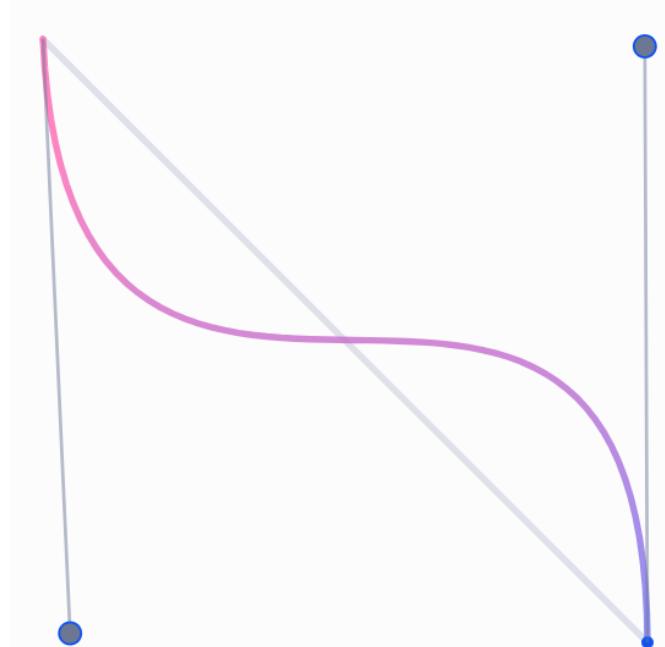
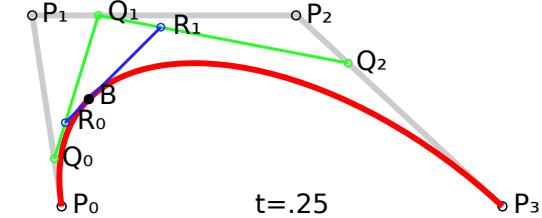
* Real-time rendering book, ch 17.1.

Bezier curves: problems

- Bezier curves **do not pass through all control points** (except endpoints)
- Not many degrees of freedom:
 - Only position of 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 and complex computation
 - Bernstein polynomials do not interpolate well
 - **Therefore, lower degree curves (often cubic) are concatenated to form larger spline**

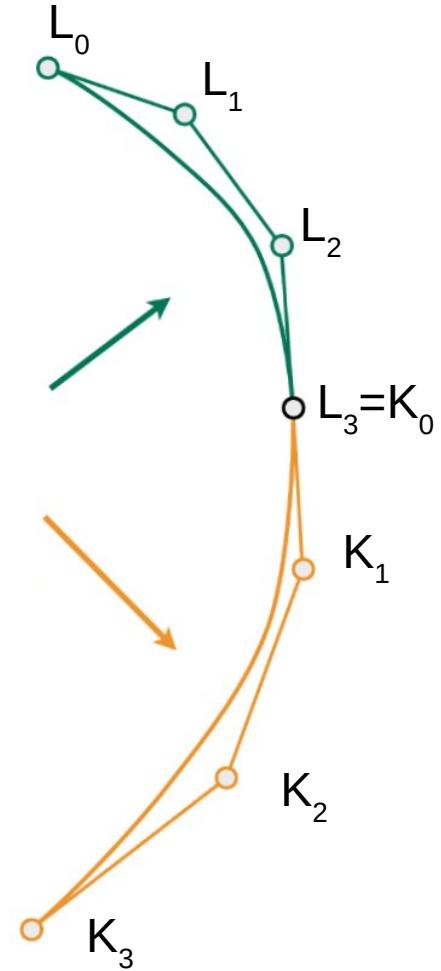
Joining Bezier curves

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



Joining cubic Bezier curves

- Example: **two cubic Bezier curves** (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



Joining cubic Bezier curves

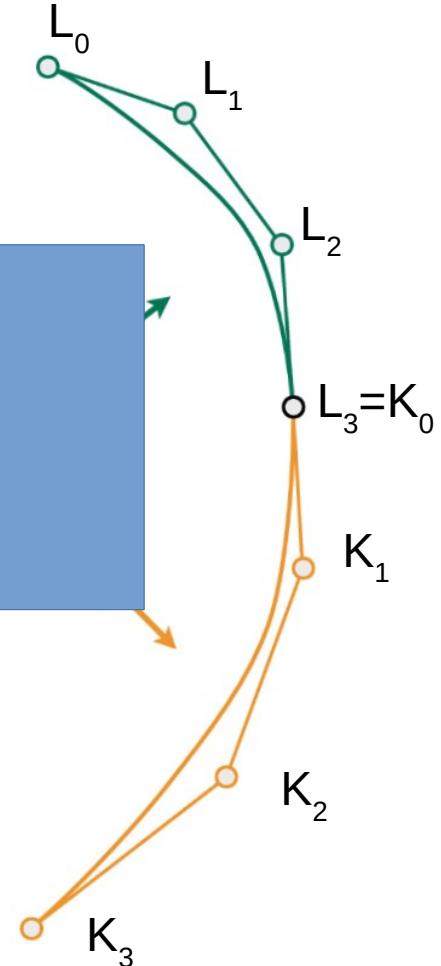
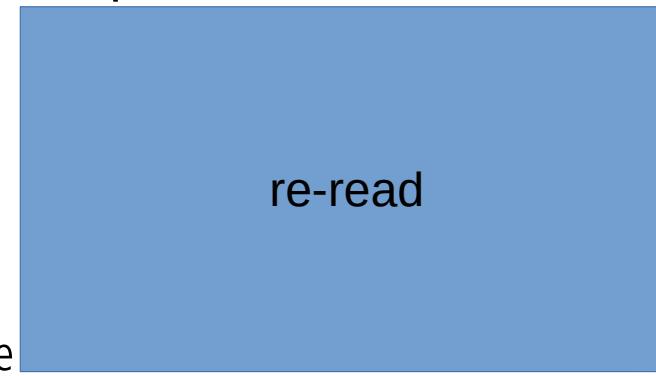
- Composite curve formed from several curve pieces is called **piecewise Bezier curve, $p(t)$**

- Now t is in $[t_1, t_2]$
 - First curve is $p(t')$, where t' in $[0, 1]$
 - Second curve is $p(t')$, where $t' = (t - t_1) /$

- Two curves connected just using $L_3 = K_0$ will not be joint.

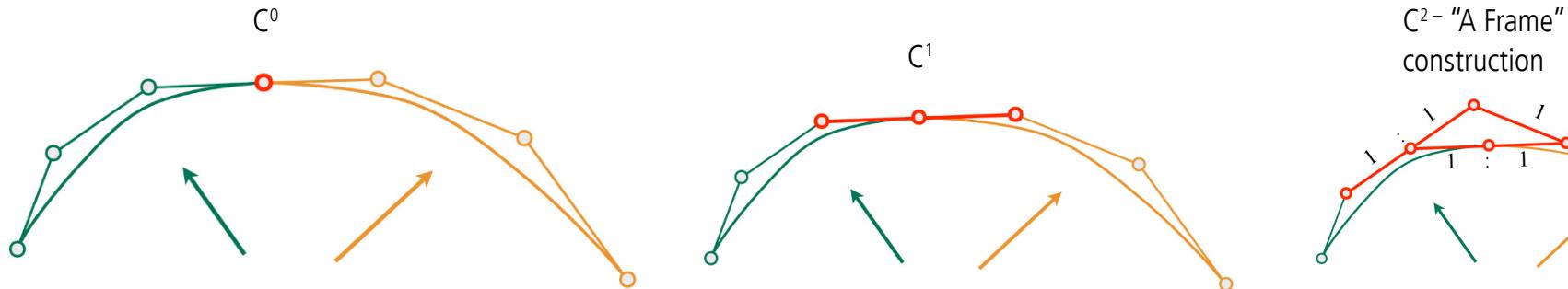
- Improved smoothness is achieved using tangent constraint:

$$(K_1 - K_0) = c(L_3 - L_2), \quad c > 0^*$$



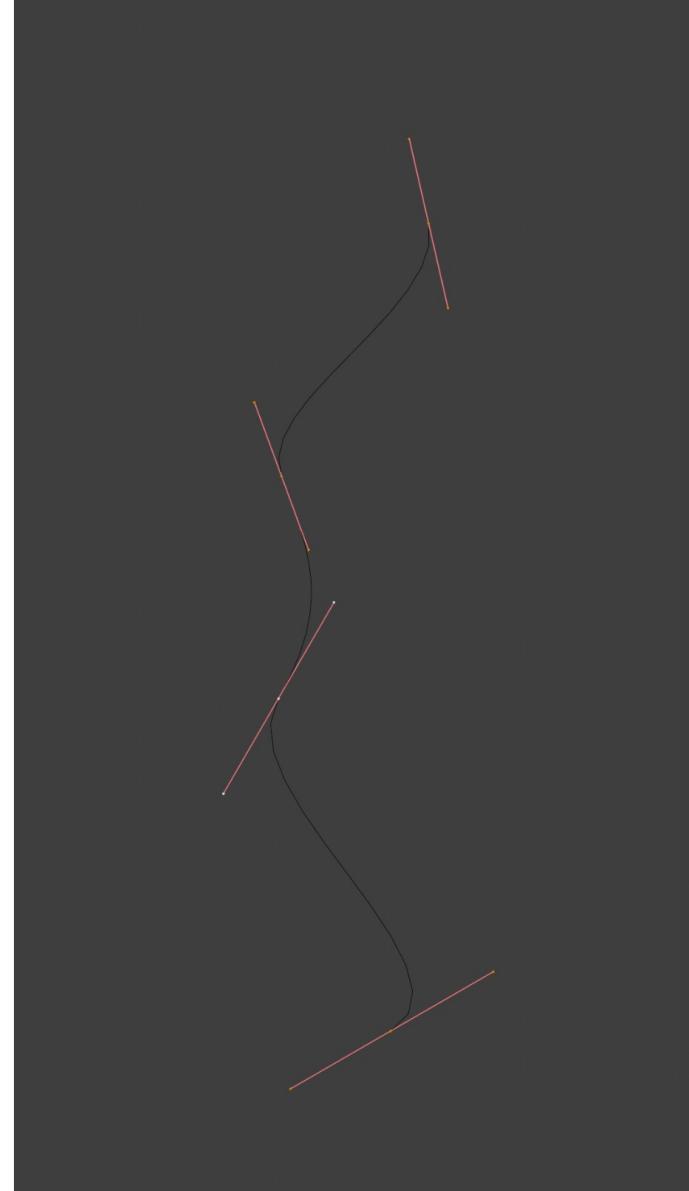
Joined curves continuity

- 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



Joined curves continuity

- Geometrical continuity:
 - G^0 – **positional continuity**: holds when the end points of two curves 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

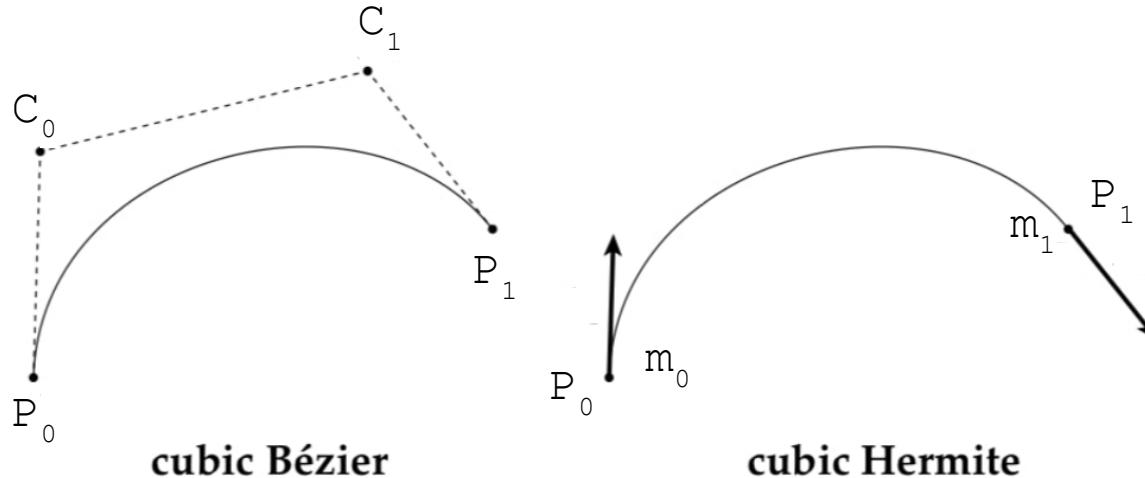


Hermite curve

- Bezier curves are good for describing theory behind constructing curves
- In practice, **Hermite curve** is used for simpler authoring
 - Most often, **cubic Hermite interpolation curve** is used

Cubic Hermite curve

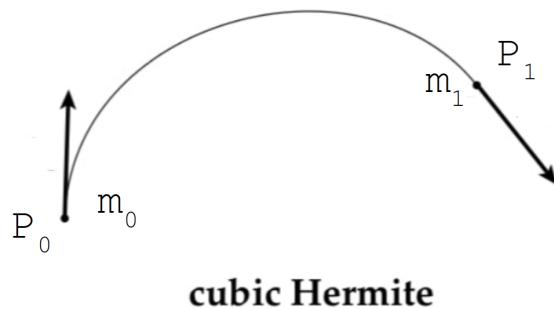
- Cubic Hermite interpolation curve:
 - Interpolates points P_0 and P_1 – **starting and ending control points**
 - Interpolation is **controlled with tangent vectors**: m_0 at P_0 and m_1 at P_1
 - Tangent vector magnitude and direction influence the curve shape



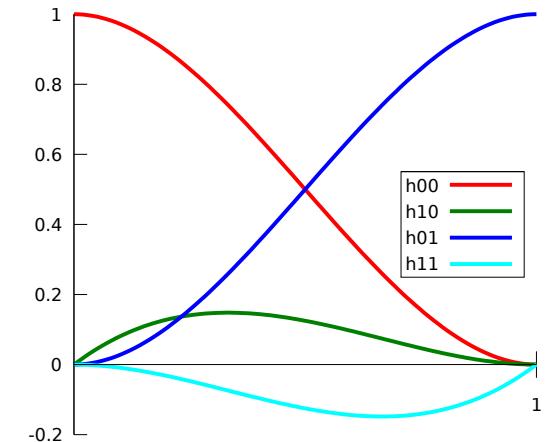
Cubic Hermite curve

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

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

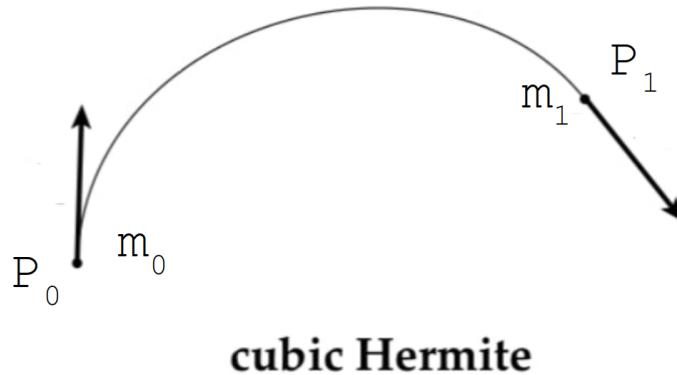


- Similarly as discussed with Bezier: Bernstein form with Hermite basis function exists



Cubic Hermite curve

- In practice, cubic Hermite interpolation curve is represented with matrix for efficient computation.

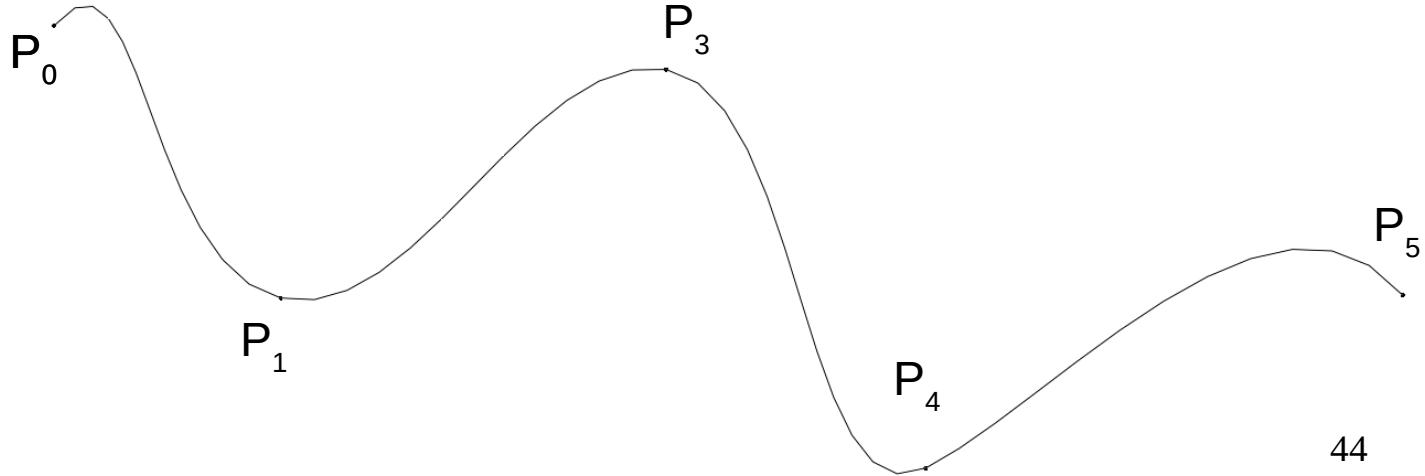


$$p(t) = (t^3 \ t^2 \ t \ 1) \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ m_0 \\ m_1 \end{pmatrix}$$

Check matrix

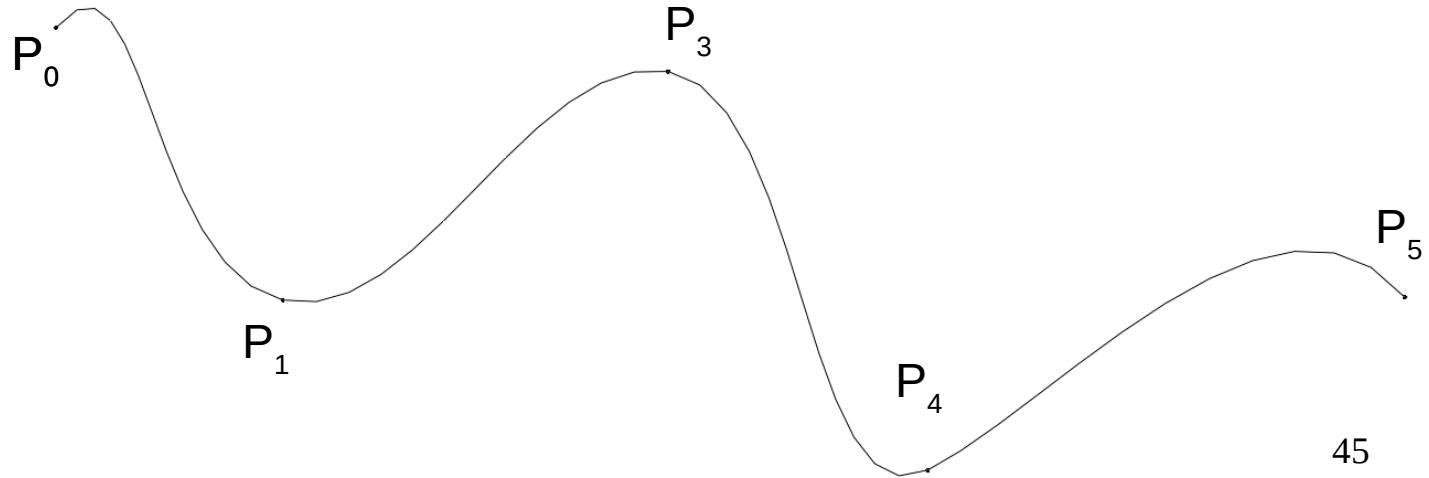
Splines: assembly of curves

- When interpolating more than two points, several Hermite curves can be connected together (similarly as done with Bezier).
- Connecting multiple points $P_0 \dots P_n$
 - $n - 1$ cubic Hermite curves must be used
 - Resultant is **assembly of curves** called spline
 - Elements of assembly are called **segments** (e.g., Hermite segments)



Catmull-Rom spline

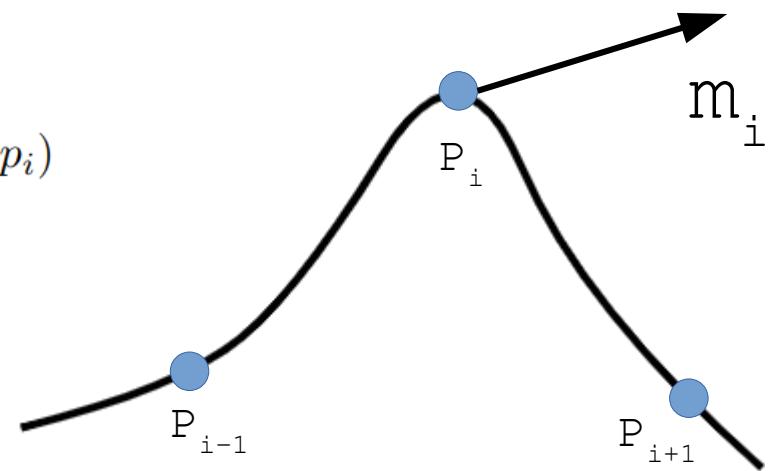
- Cubic Hermite interpolation curve tangents can be hard to control
- Therefore, **tangents are calculated from control points $P_0 \dots P_n$** specified by the user
 - Given points $P_0 \dots P_n$, the goal is to calculate tangents needed for Hermite curves which would create spline: continuous differentiable curve passing through each point → **Catmull-Rom spline**



Kochanek-Bartels method

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

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

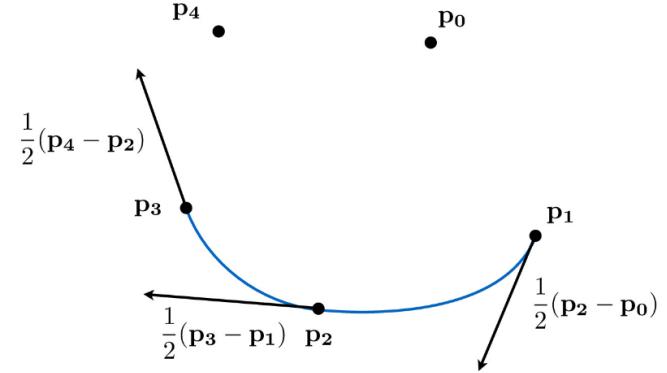


Tension parameter - a : length of tangent. Higher values → sharper curve bending
Bias parameter - b : direction of tangent

Catmull-Rom spline

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

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



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

Properties:

- Finding points on Catmull-Rom spline is fast and efficient
- Interpolating: passing through all control points but doesn't stay inside convex hull of points

B-spline

- Next to Catmull-Rom spline, B-splines are widely used

B-spline

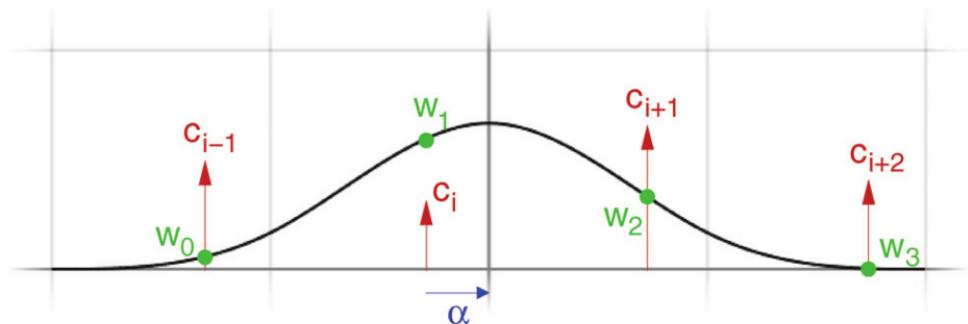
- Types:
 - Uniform: uniformly space control points
 - Non-Uniform
- Similarly to Bezier curve, it is parameterized function of x

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

$n-1$ – degree, $c(k)$ – control points, β_n - basis function

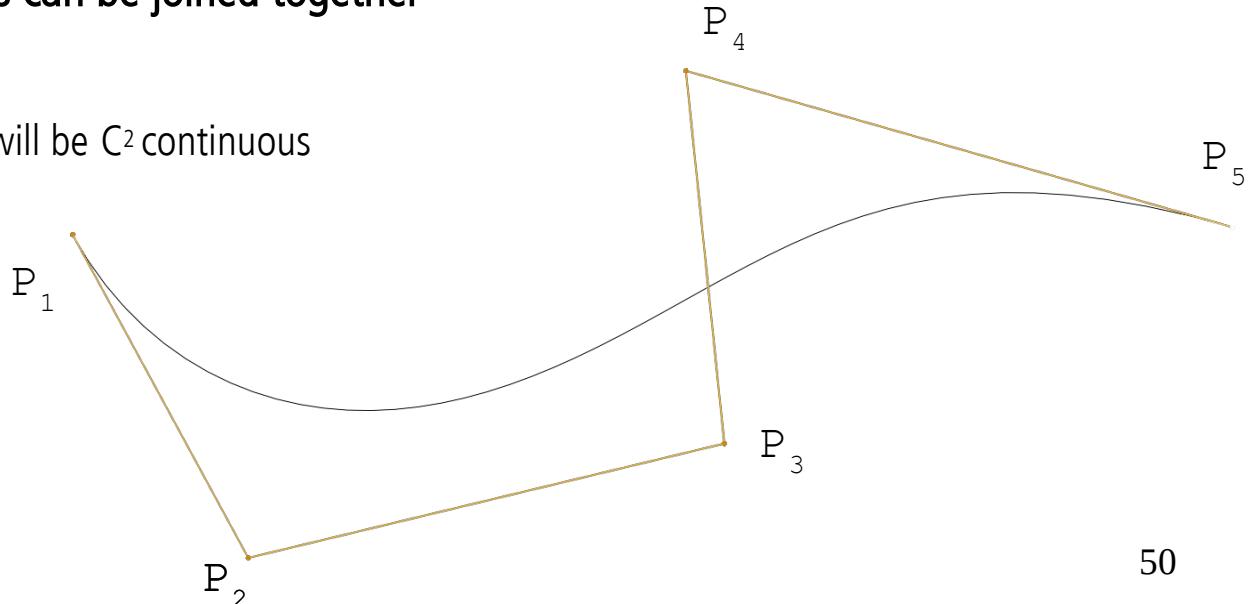
- Cubic B-spline is most widely used. Basis function:

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



Uniform cubic B-spline

- Cubic B-spline is **non-interpolating**
 - 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**
 - Cubic B-spline is C^2 continuous
 - Curve composed of multiple B-splines will be C^2 continuous



Limitations and generalizations

- Example: **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
- Requirement of **uniformly spaced control points**:
 - Example: animation when positions of object are known at $t = 1, 2, 3$ and 10 .
 - Further generalization: **Non-uniform rational B-splines (NURBS)**
 - Very often used in CAD tools

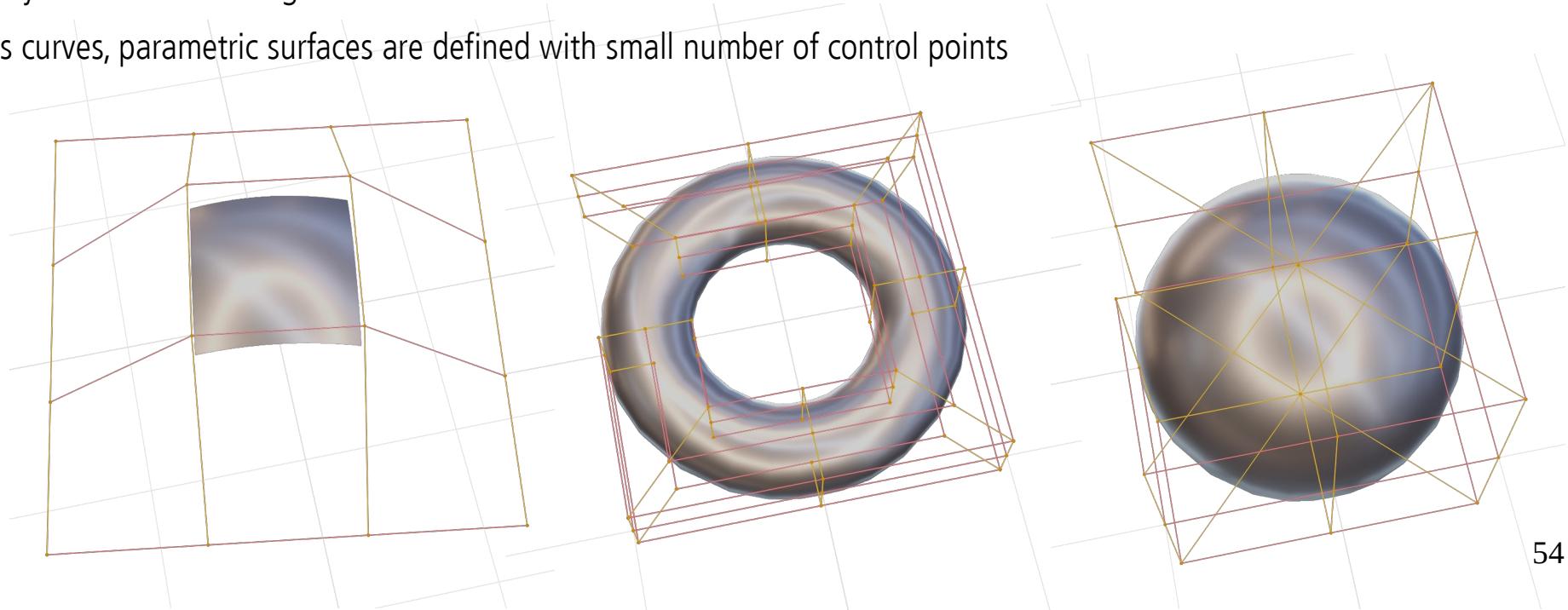
Animation via curves

<https://www.realtimerendering.com/#curves>

Parametric surfaces

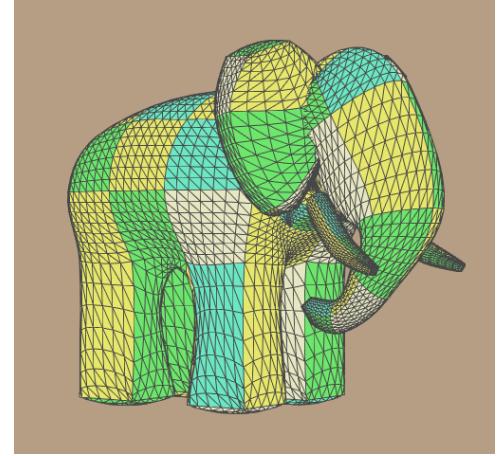
Parametric surfaces

- 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

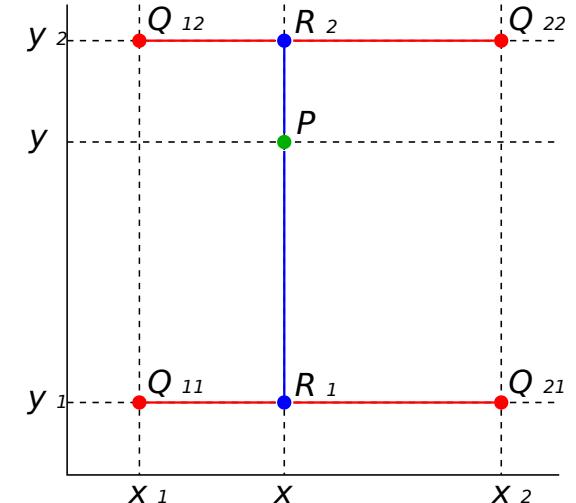
- Common types
 - Bezier surfaces
 - B-spline surfaces



[Ed Catmull](#)'s "Gumbo" model, composed from Bezier patches

Bezier surface: bilinear interpolation

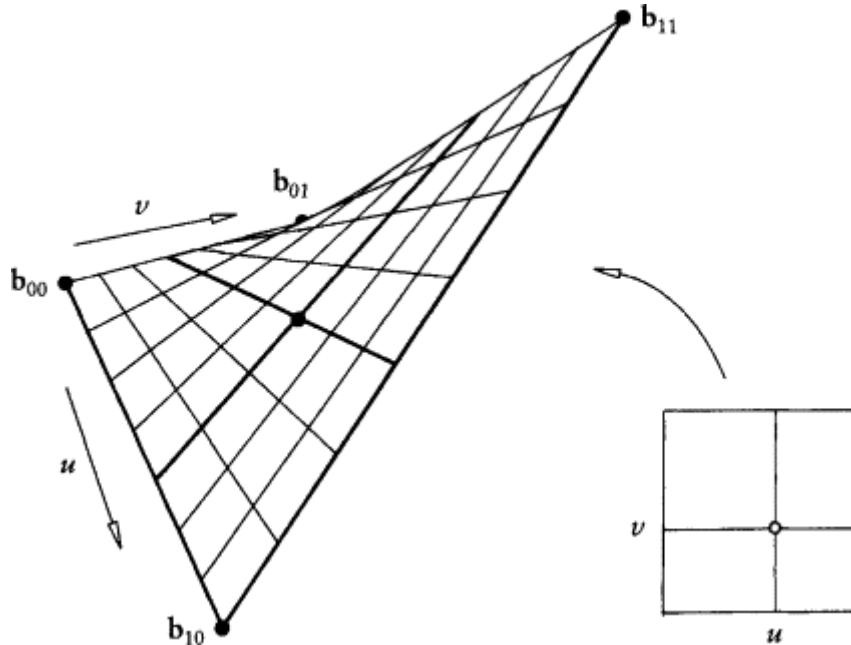
- Bezier curve is extended so it has two parameters (u, v) which define surface.
- Similarly as we started with Bezier curve by explaining linear interpolation, we explain Bezier patch by explaining **bilinear interpolation***.
 - $R_1(u) = \text{lerp}(Q_{11}, Q_{12}, u)$
 - $R_2(u) = \text{lerp}(Q_{12},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



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

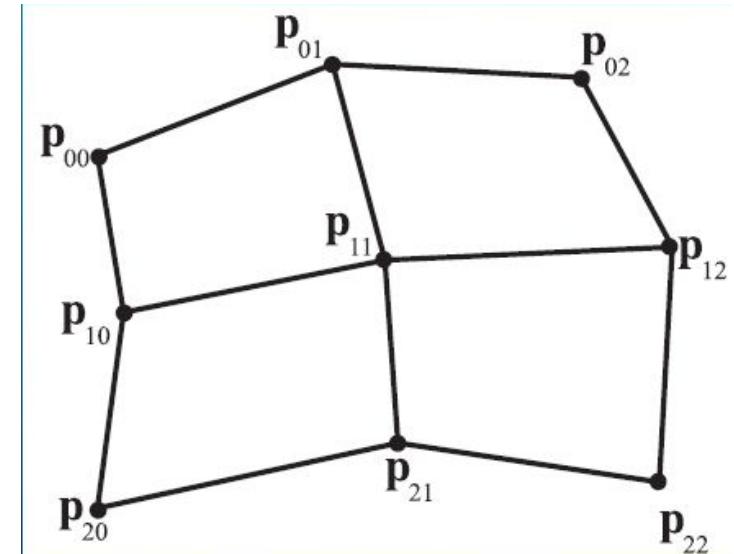
Bezier surface: patch

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



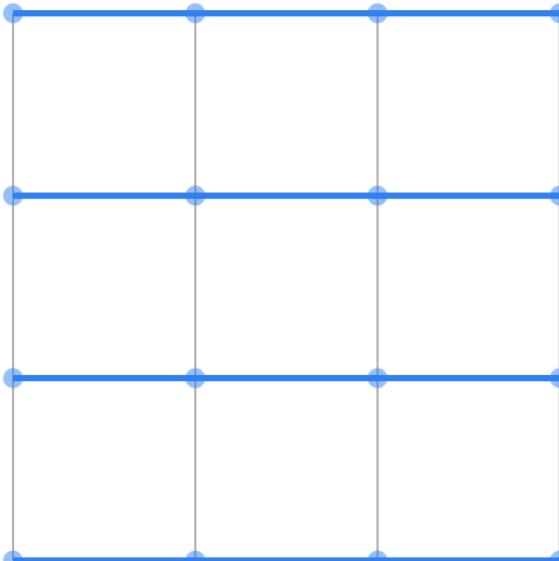
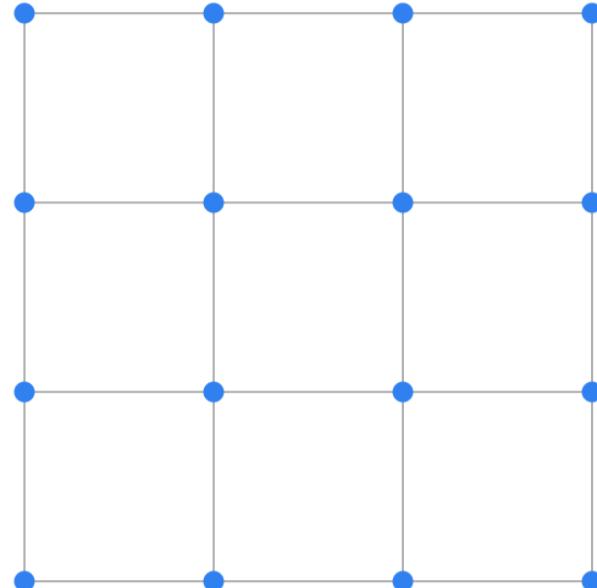
Bezier surface

- 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



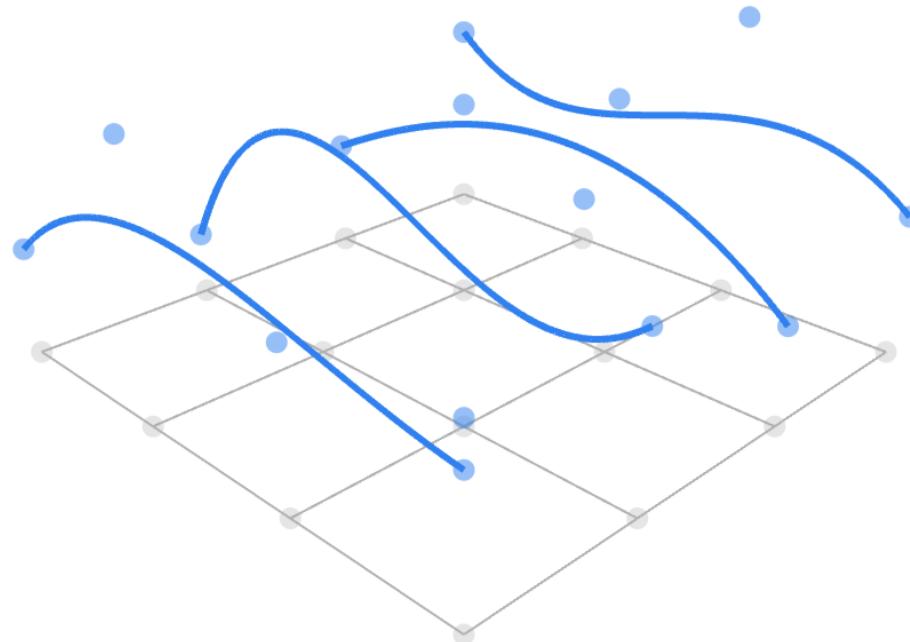
Bezier surface: repeated bilinear interpolation

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



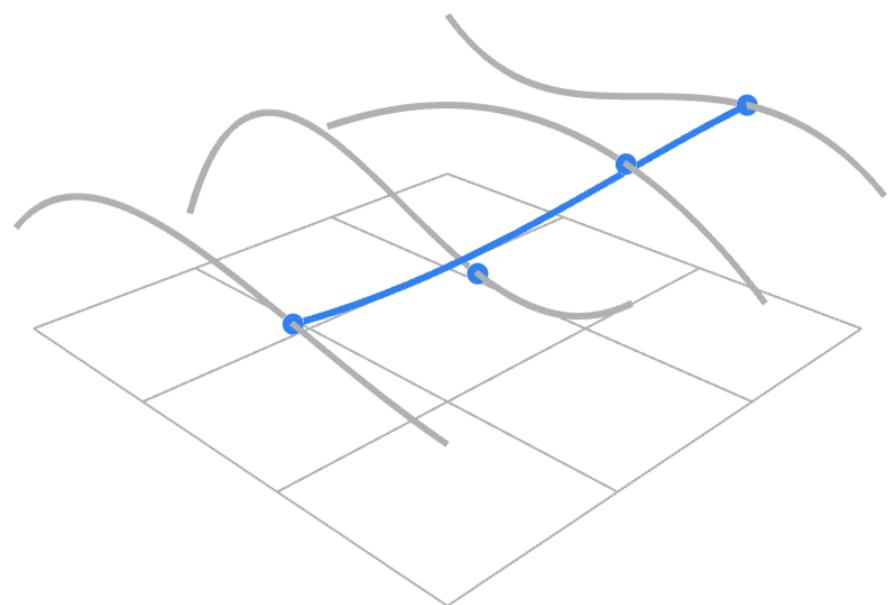
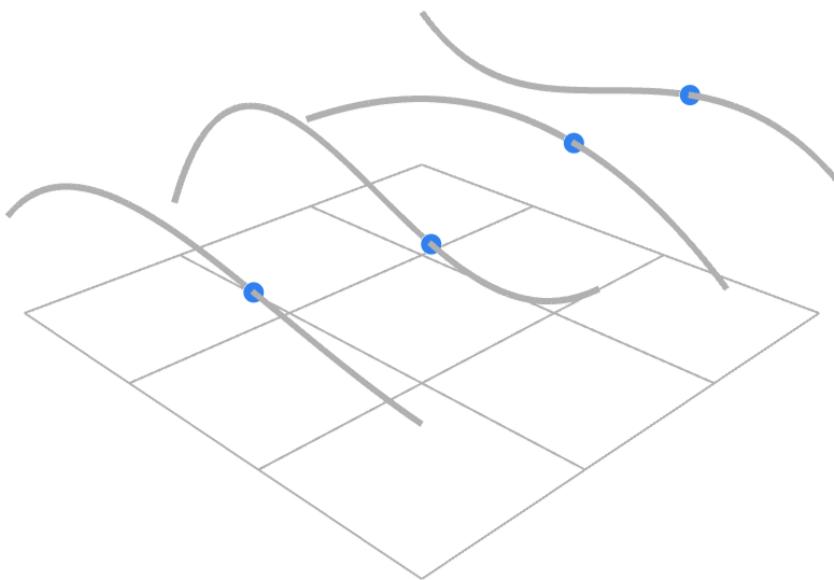
Bezier surface: repeated bilinear interpolation

- Repeated linear interpolation:



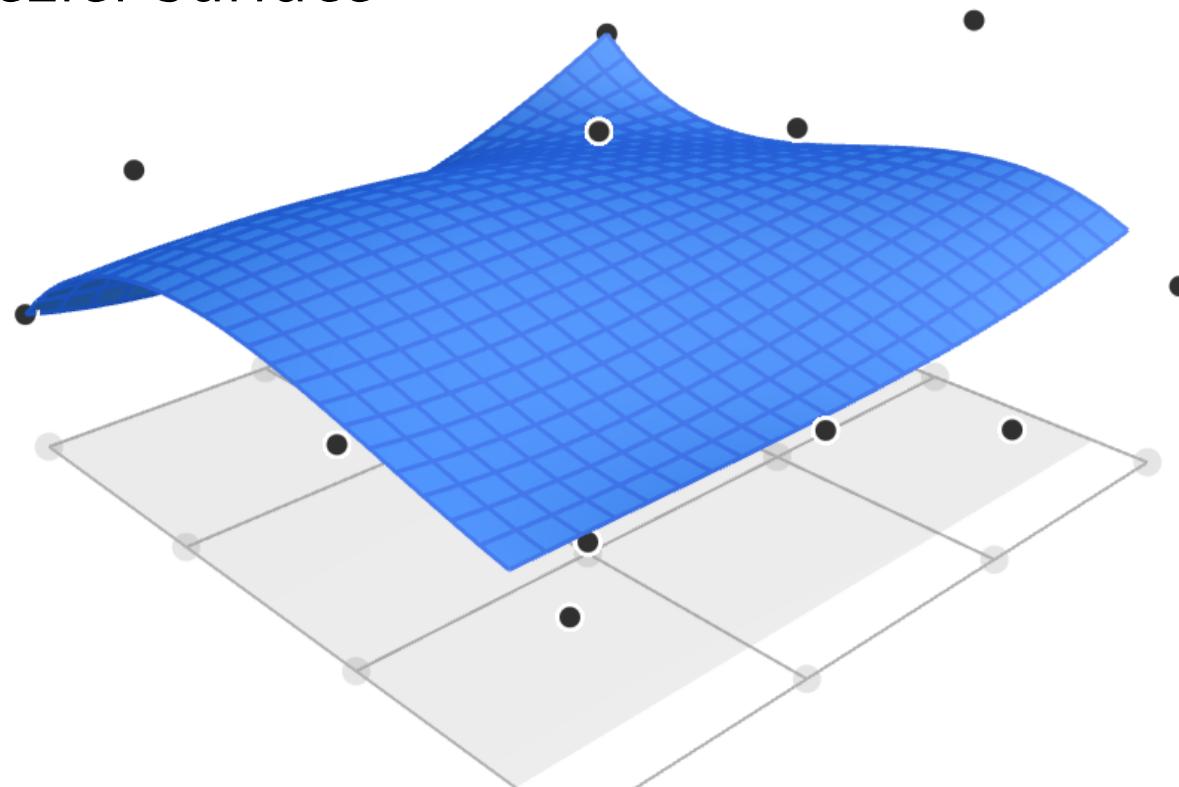
Bezier surface: repeated bilinear interpolation

- Repeated bilinear interpolation



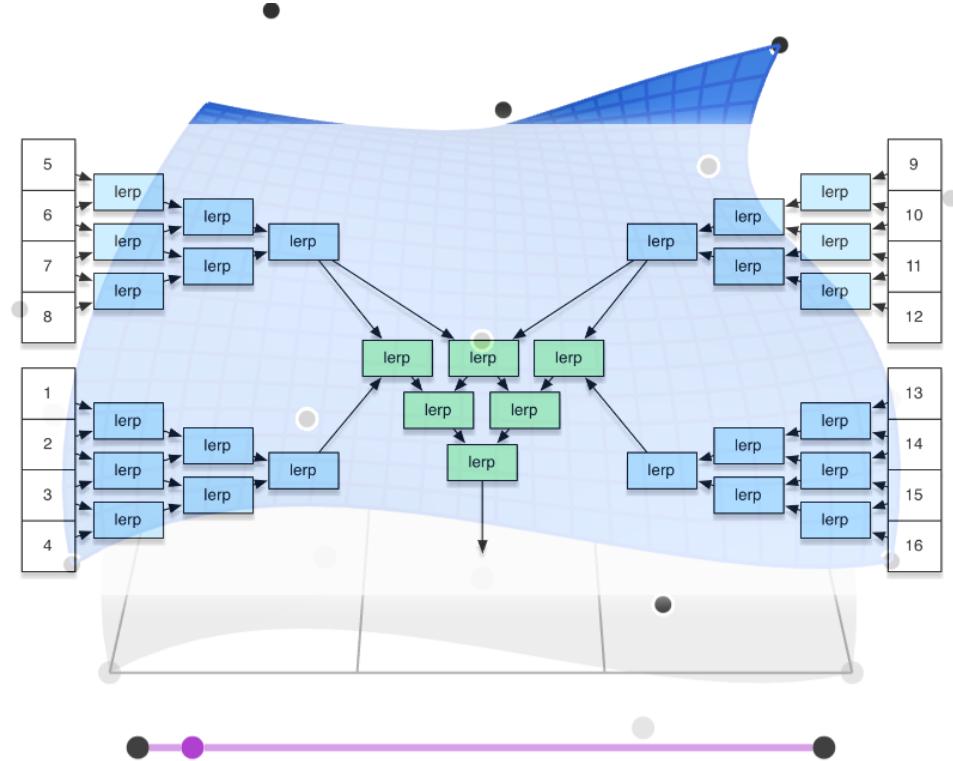
Bezier surface: repeated bilinear interpolation

- Final Bezier surface



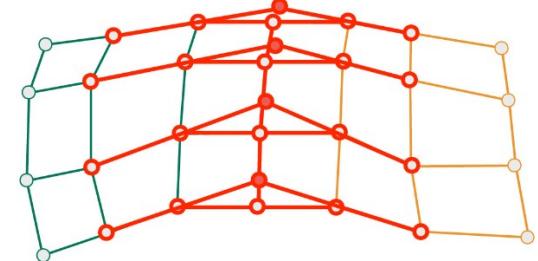
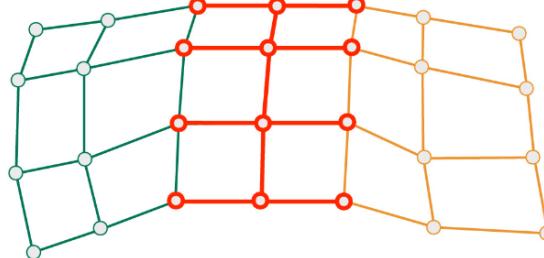
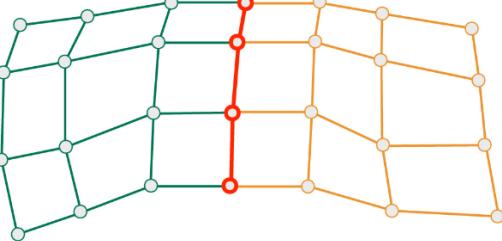
Bezier surface

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



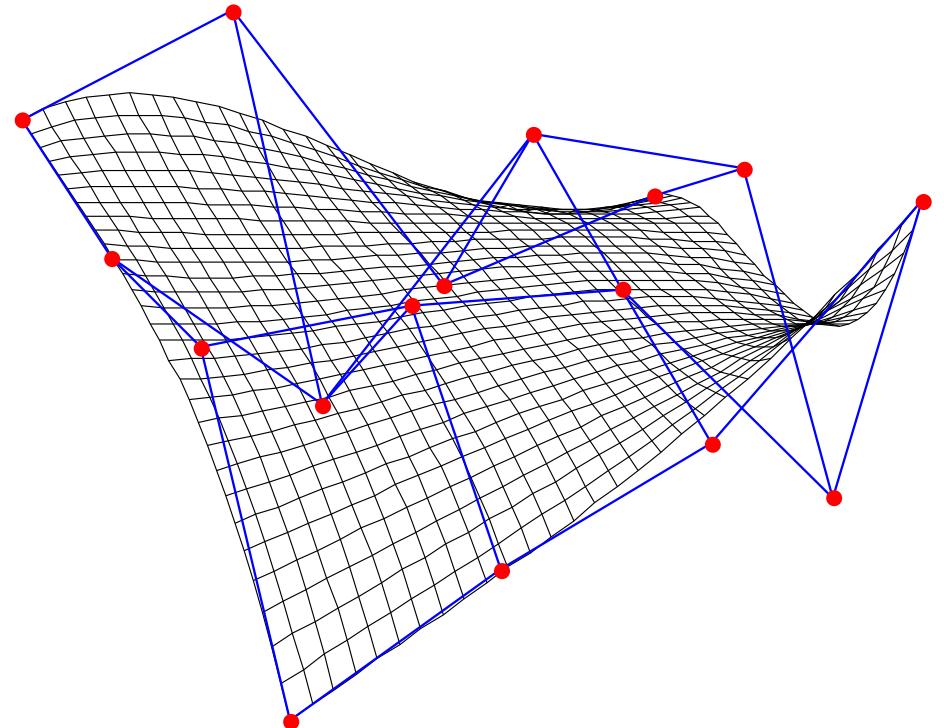
Bezier surface: continuity

- 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



Bezier surface properties

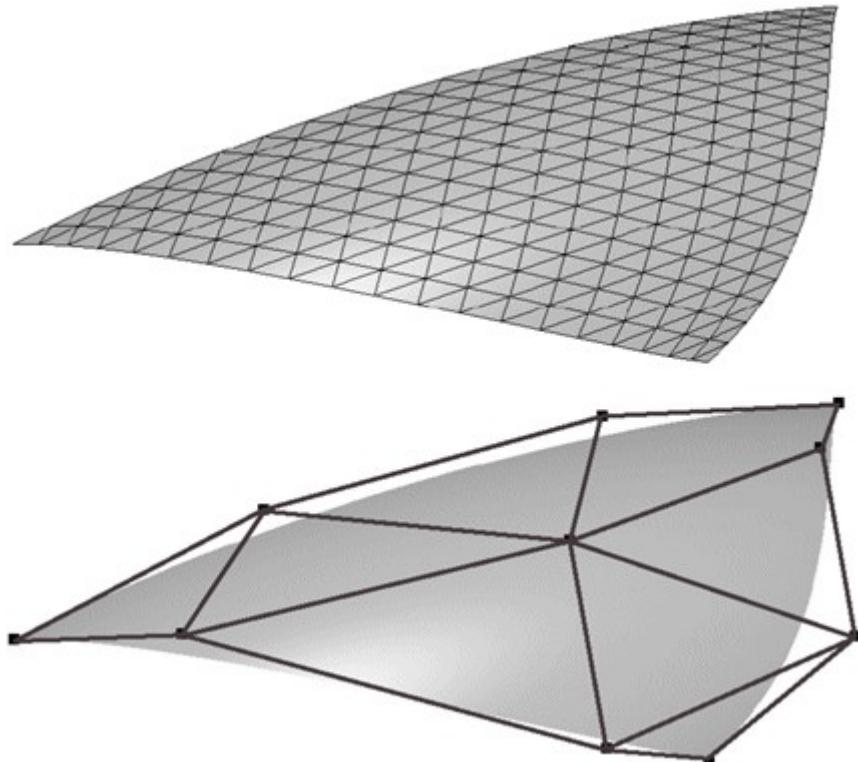
- Passes through only corner control points
- Boundary of the patch is described with Bezier curve of degree n formed by the points on the boundary
- Tangents at border points are described with Bezier curve at border points – each corner control point has two tangents: for u and v direction
- Patch 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



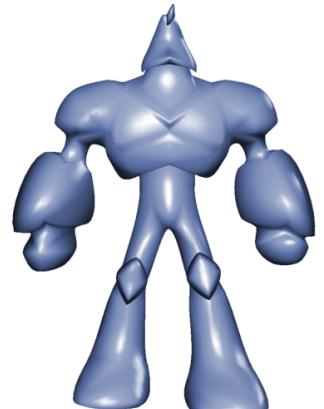
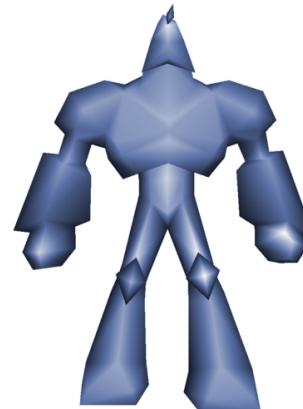
Other parametrized surfaces

- 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:

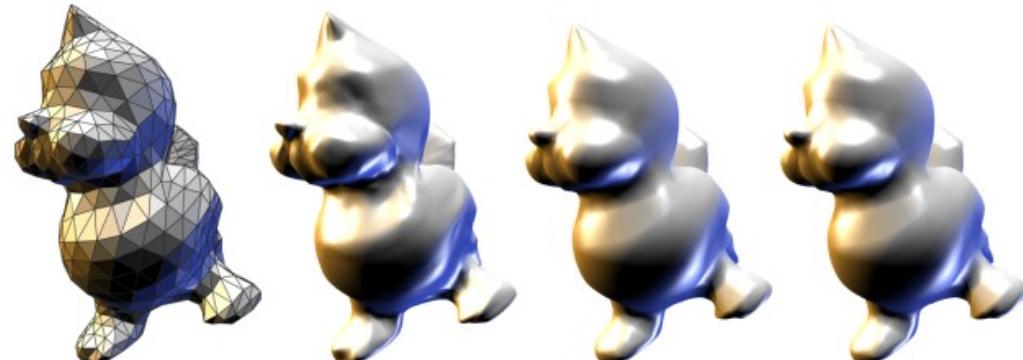
- Creases in PN triangles are hard to control
- Continuity between Bezier triangles is C0 but looks acceptable for certain applications



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

Other parametrized surfaces

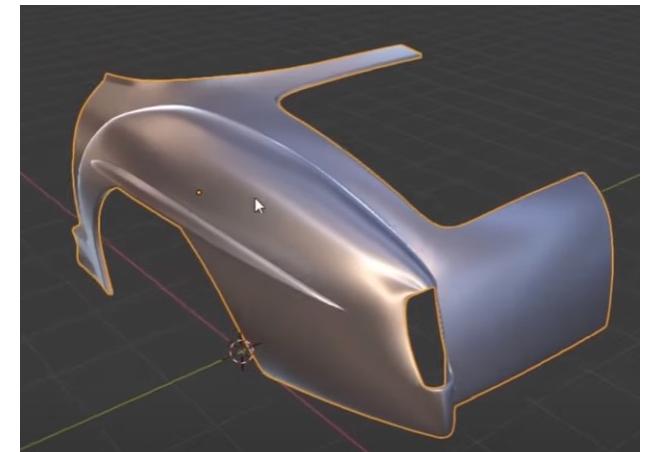
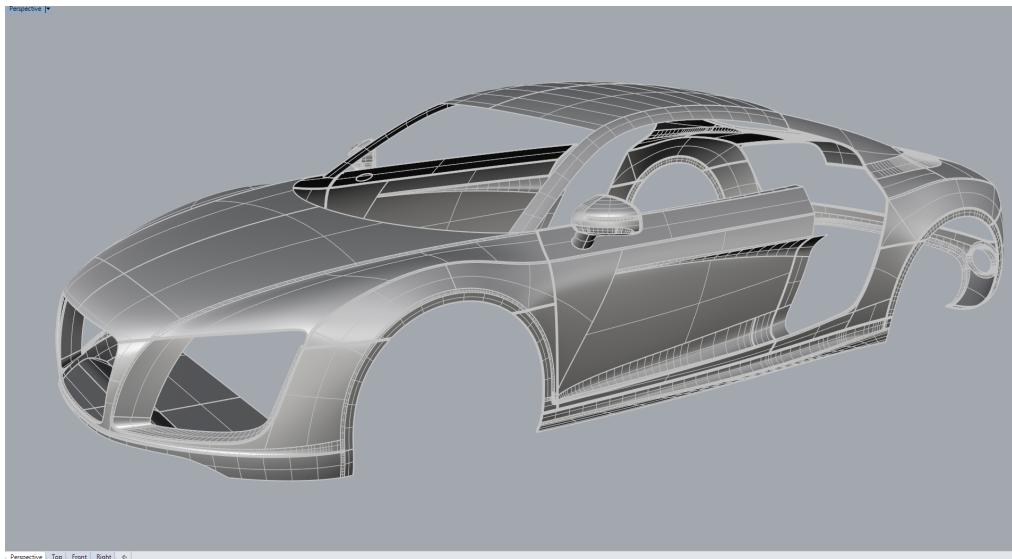
- **Phong tessellation**
 - Similar as PN triangles, given the triangle points with normals, construct “better-looking” surface
 - Phong tessellation attempts to create geometric version of Phong shading normal using repeated interpolation resulting in Bezier triangles.



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

B-Spline surfaces

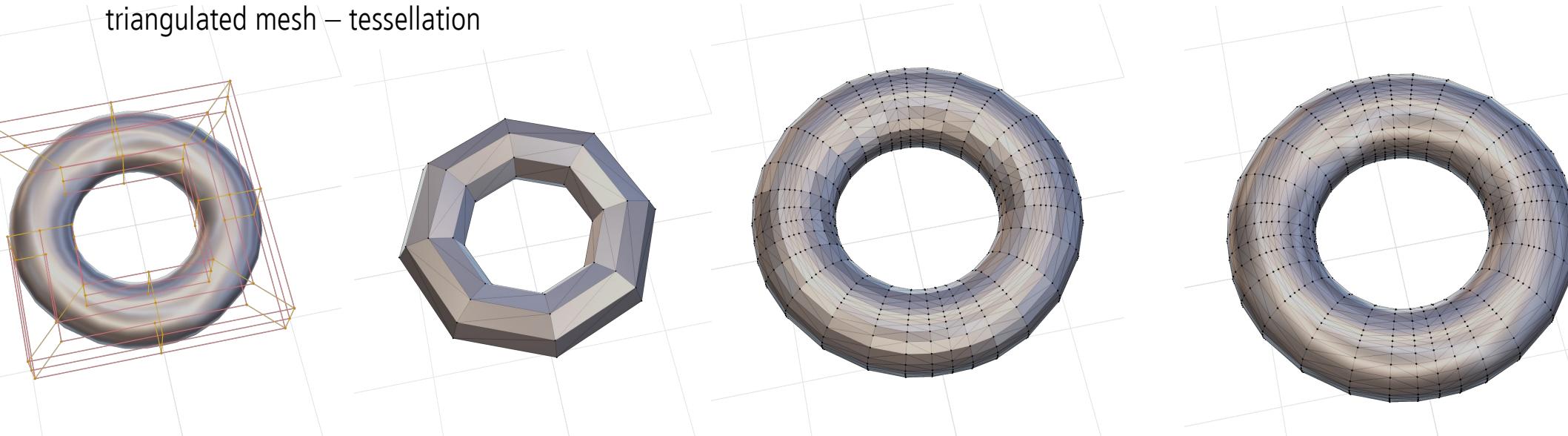
- **B-Spline curves** can be extended to B-Spline surfaces which are similar to Bezier Surface
- Often bicubic B-Spline surface is used to form composite surface
 - Essential for Catmull-Clark subdivision surfaces (will be discussed later)
- **Non-uniform rational B-Spline surface (NURBS)** is often used in 3D modeling software



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

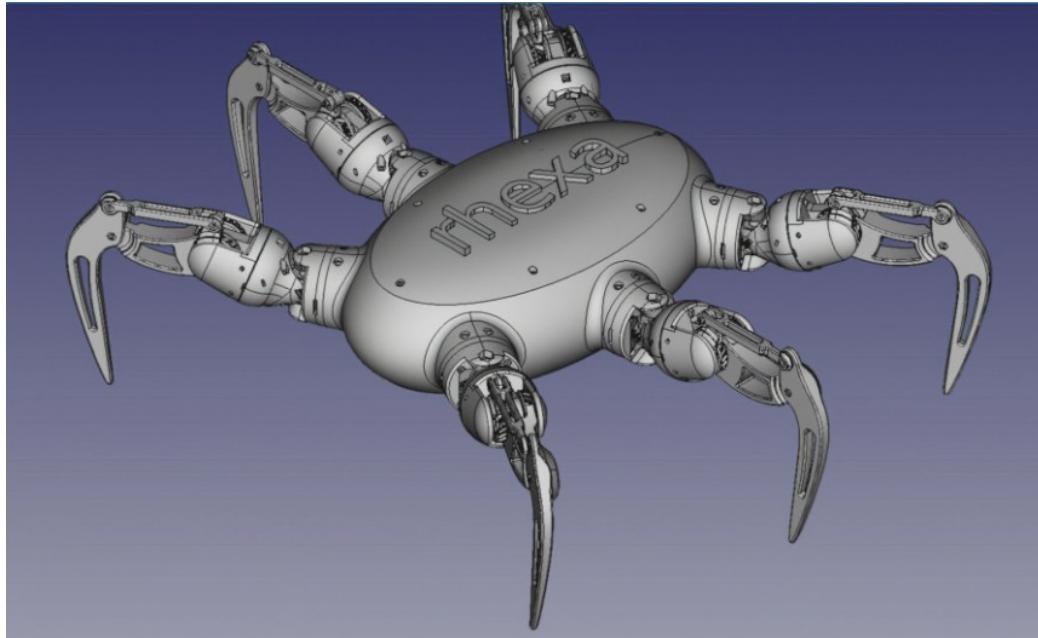
Rendering of curved surfaces

- 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.



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>

Implicit surfaces

Implicit surfaces



TODO

Literature

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