

Lecture 6: Parametric curves and surfaces, implicit surfaces

DHBW, Computer Graphics

Lovro Bosnar

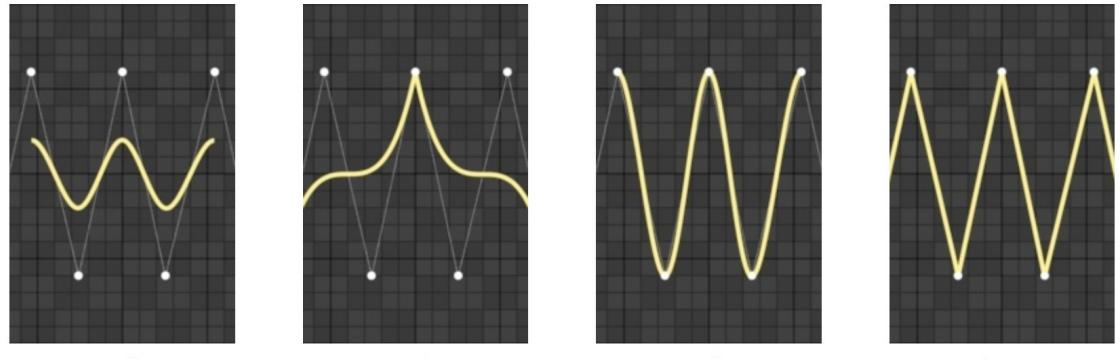
8.2.2023.

Syllabus

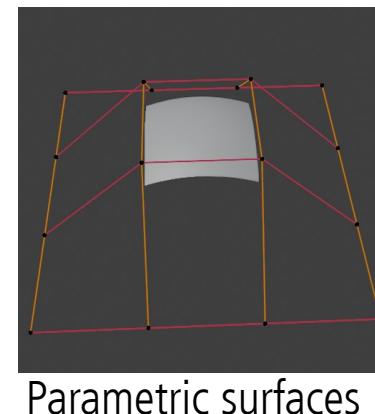
- 3D scene
 - Objects
 - Shape
 - Material
 - Lights
 - Cameras
 - Rendering
 - Image and display
-
- A blue arrow points from the 'Shape' bullet point in the first list to a rounded rectangular callout box. The callout box contains three bullet points: 'Parametric curves', 'Parametric surfaces', and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

Recap: shape representations

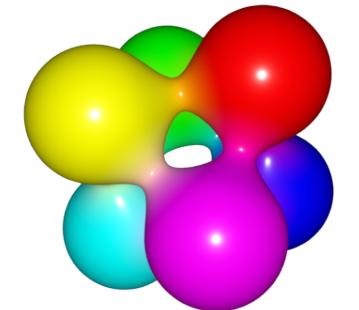
- Points
 - Point clouds
 - Particles and Particle systems
- Lines
 - Parametric curves
- Surfaces:
 - Polygonal mesh
 - Subdivision surfaces
 - **Parametric surfaces**
 - **Implicit surfaces**
- Volumetric objects/solids
 - Voxels
 - Space partitioning data-structures



Parametric curves



Parametric surfaces



Implicit surfaces

Recap: shape representations

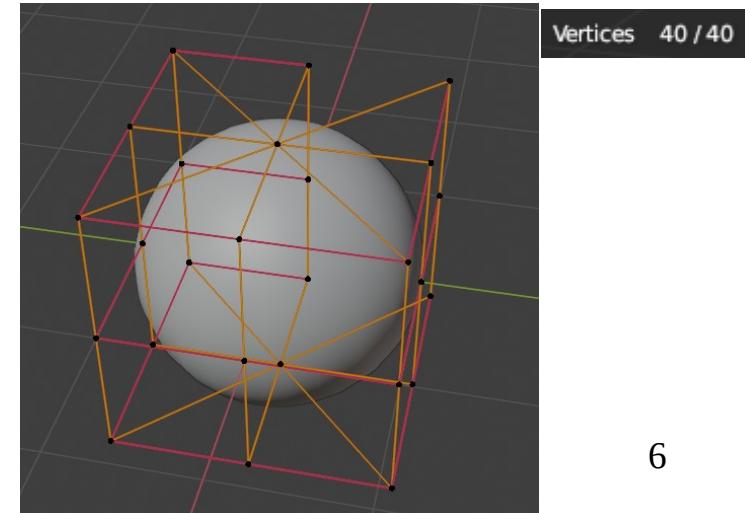
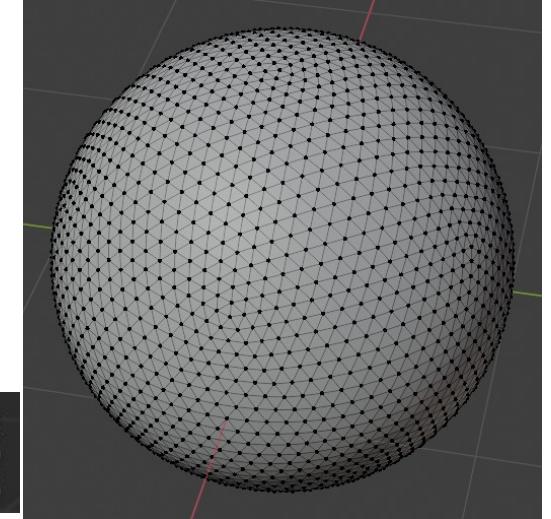
- Mesh surface representation: the most used and widespread
 - **Triangle:** basic atomic rendering primitive for GPU graphics pipelines and most ray-tracers
- Objects made in modeling systems can have many underlying geometric descriptions.
 - Easier and efficient modeling of shapes for the user
 - All higher-level geometrical descriptions are evaluated as set of triangles and then used

Parametric surfaces vs Mesh

Parametric surfaces have advantages over meshes in certain scenarios:

- More **compact representation** than meshes:
 - Described with small number of control points and parametric equation
 - 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 organic and curved objects
- Certain modeling tasks can be efficiently performed
 - Example: less operations are needed for animation and collision

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



Applications



Paul de Casteljau



1959



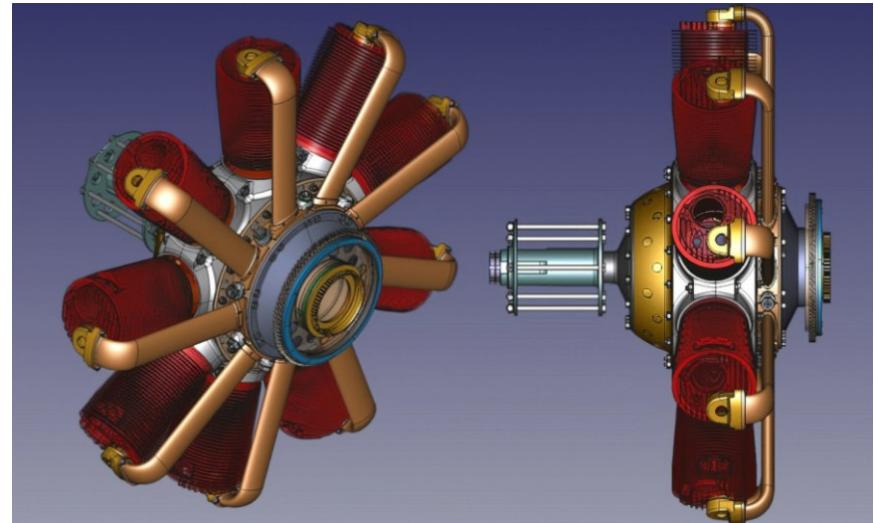
Pierre Bézier



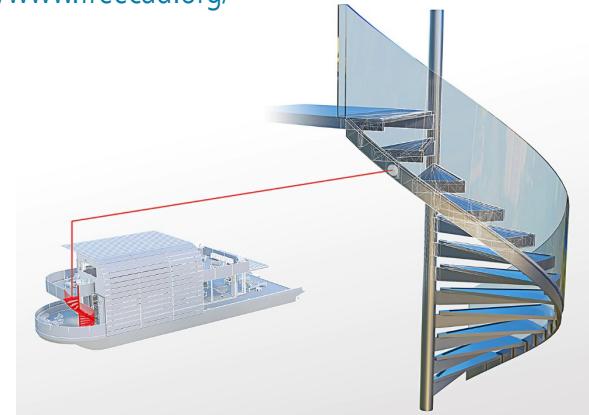
1962

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

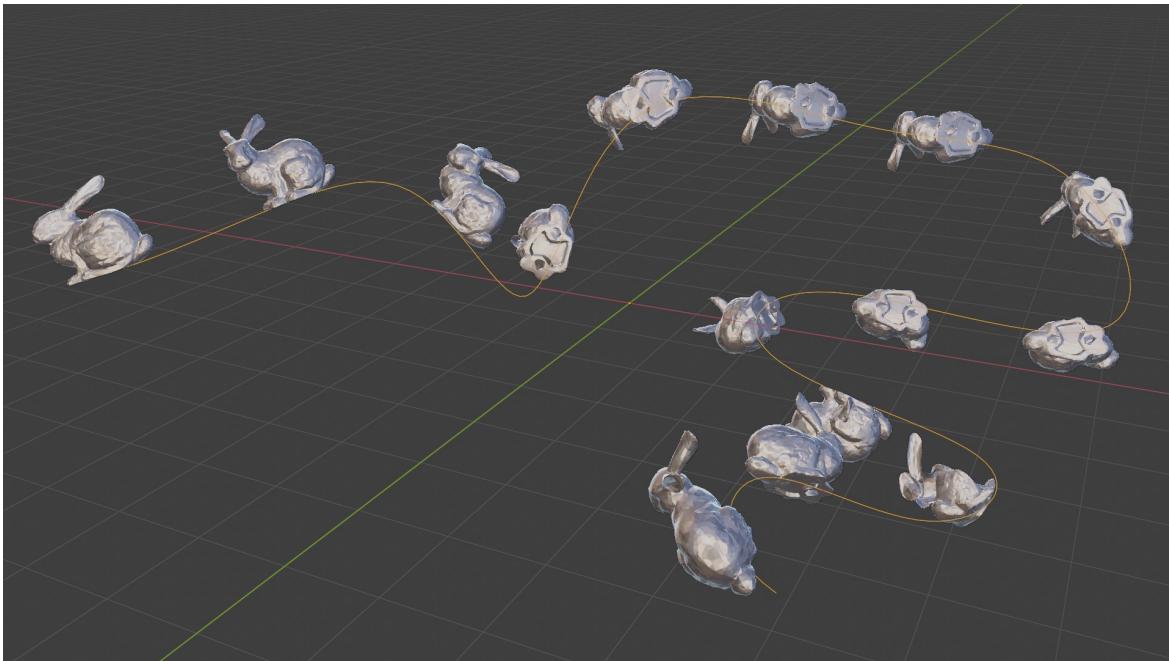
https://en.wikipedia.org/wiki/Pierre_B%C3%A9zier



Manufacturing product design using parametric surfaces
<https://www.freecad.org/>



Architecture and parametric surfaces:
<https://www.autodesk.com/products/autocad/include-toolsets/autocad-architecture>



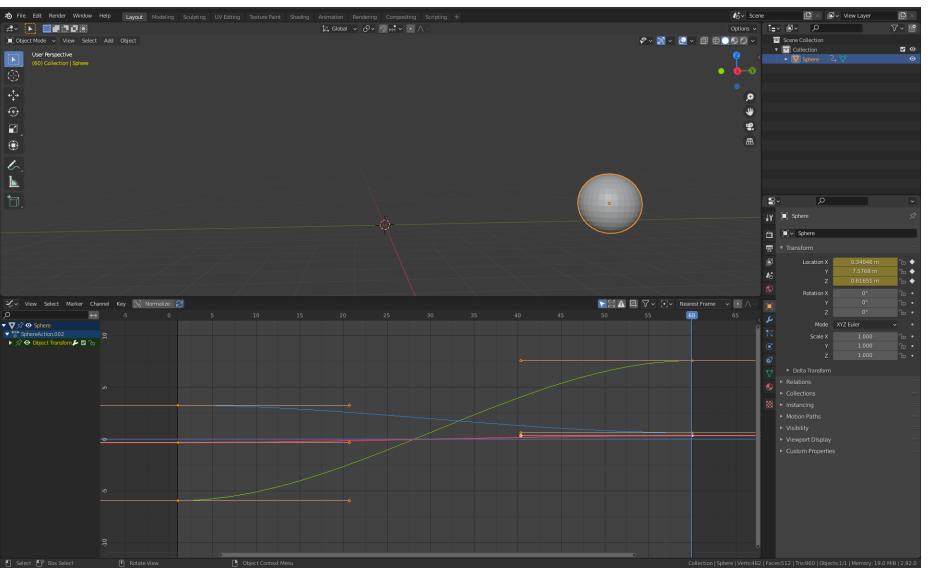
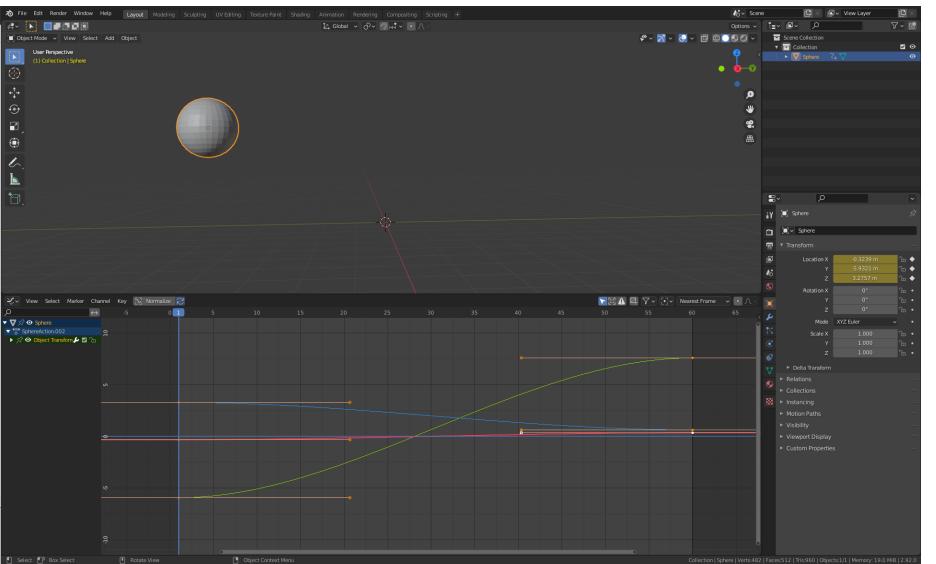
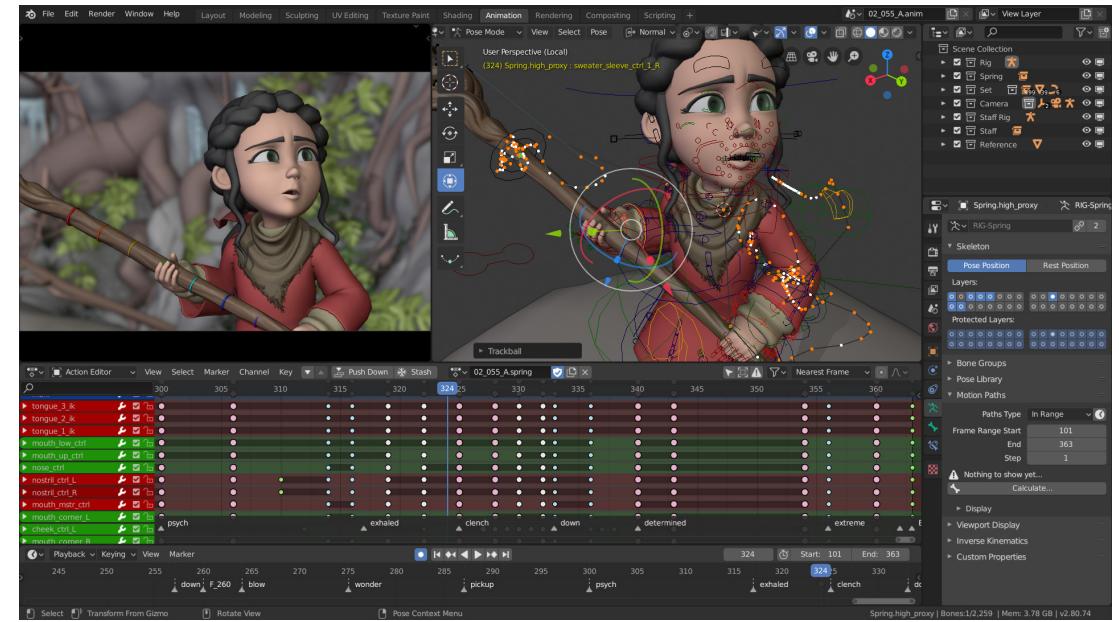
Animation paths using curves:

https://docs.blender.org/manual/en/latest/animation/constraints/relationship/follow_path.html

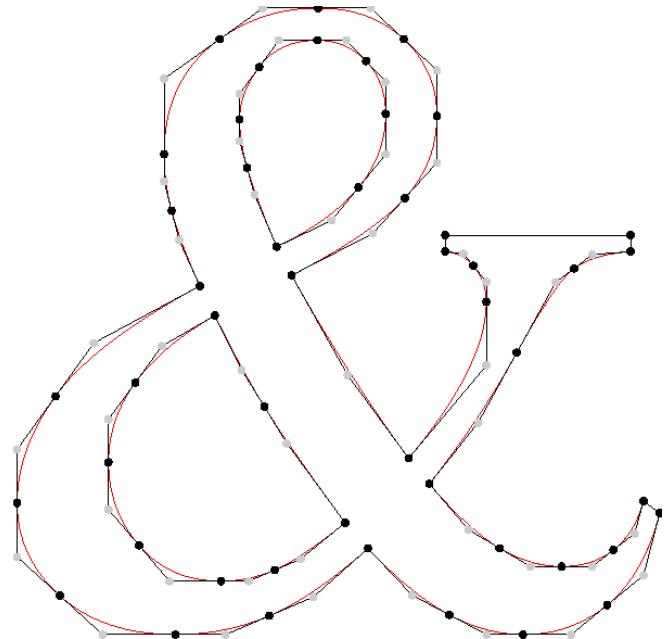


Rendering hair using curves:

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



Interpolation between key-frames
(transforms) using curves:
https://docs.blender.org/manual/en/latest/editors/graph_editor/introduction.html



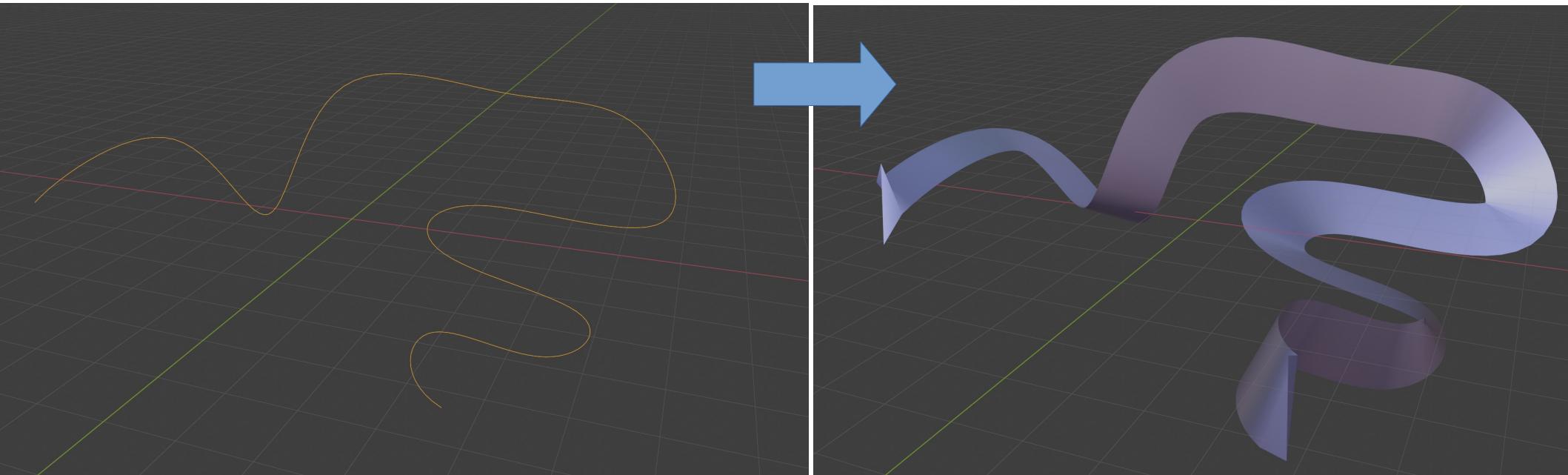
Font design (using Bezier curves):
<https://font-bakers.github.io/knead/user-guide/>



Character modeling using Bezier patches

Parametric curves and surfaces

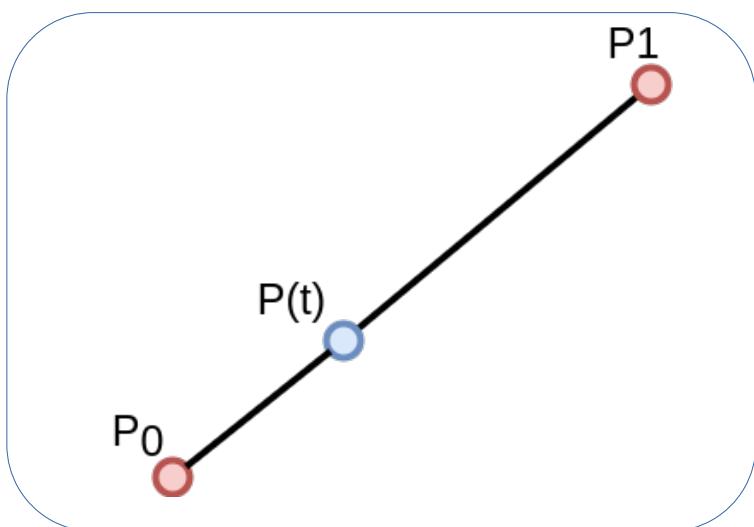
- Parametric surfaces are extension of parametric curves



Parametric curves

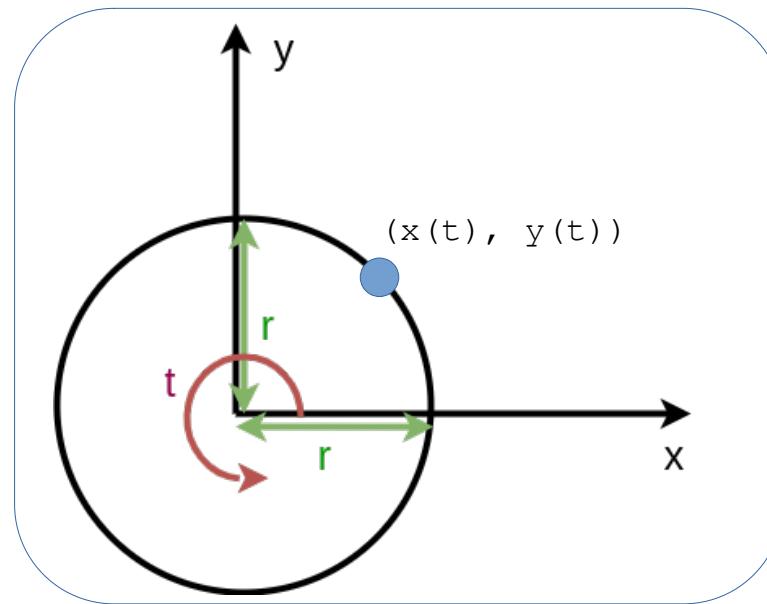
Parametric curves

- 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

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

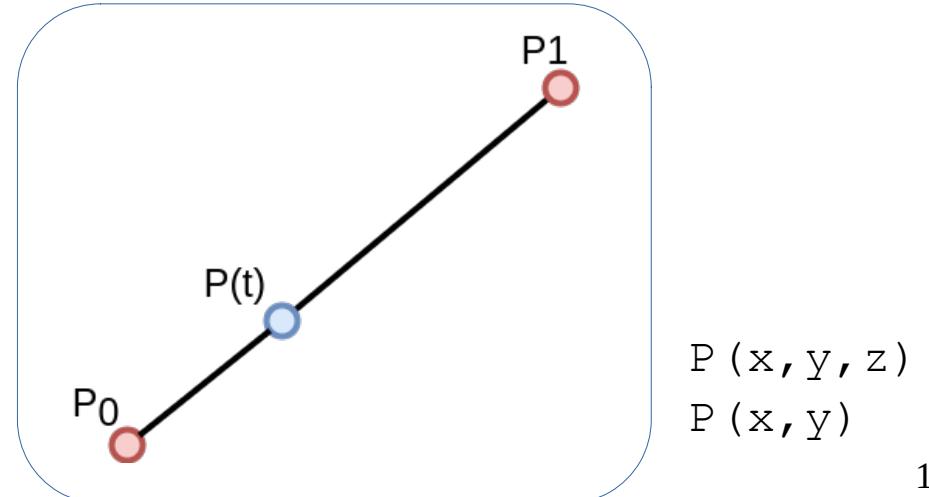
$$y(t) = r \sin(2 * \pi * t), t \text{ in } [0, 1]$$

Parametric curves

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

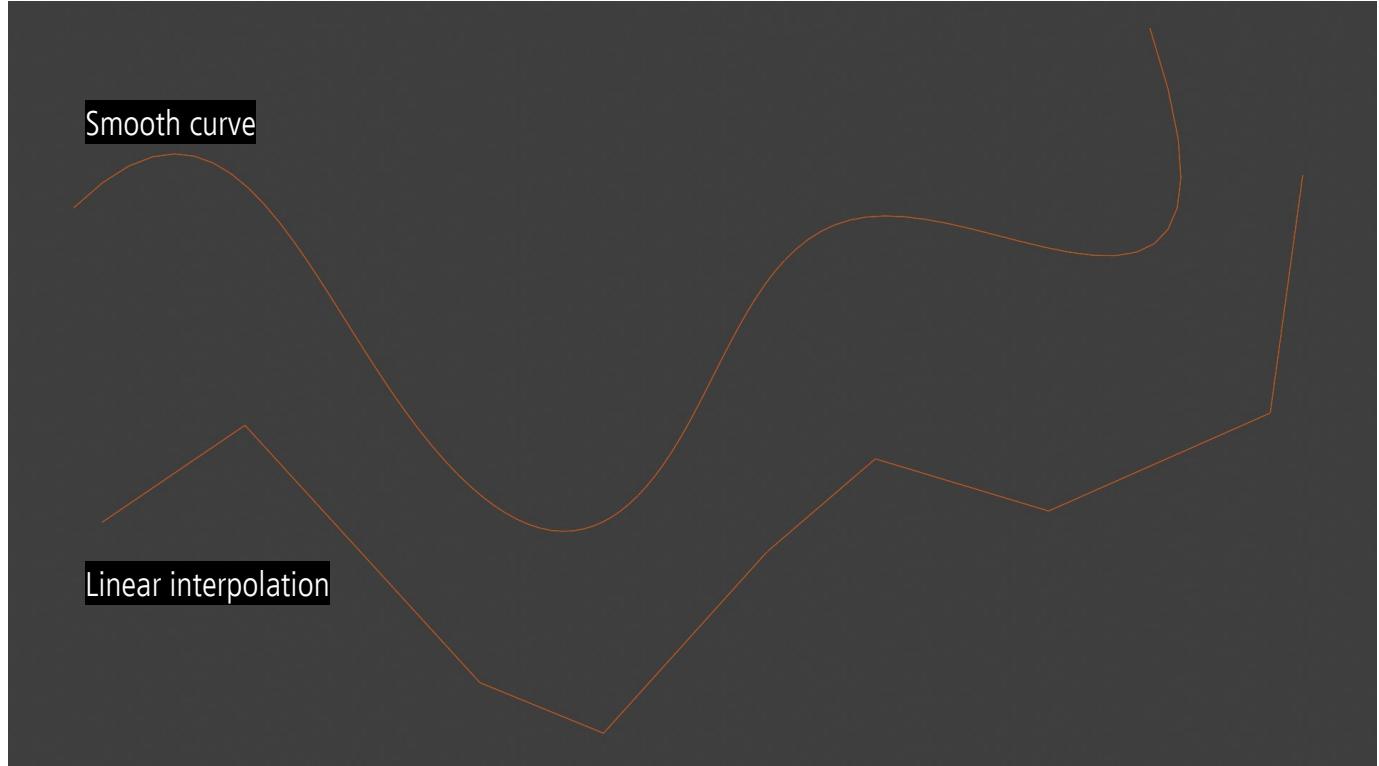
Bezier curves: linear interpolation

- 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 $P(1) = P_1$



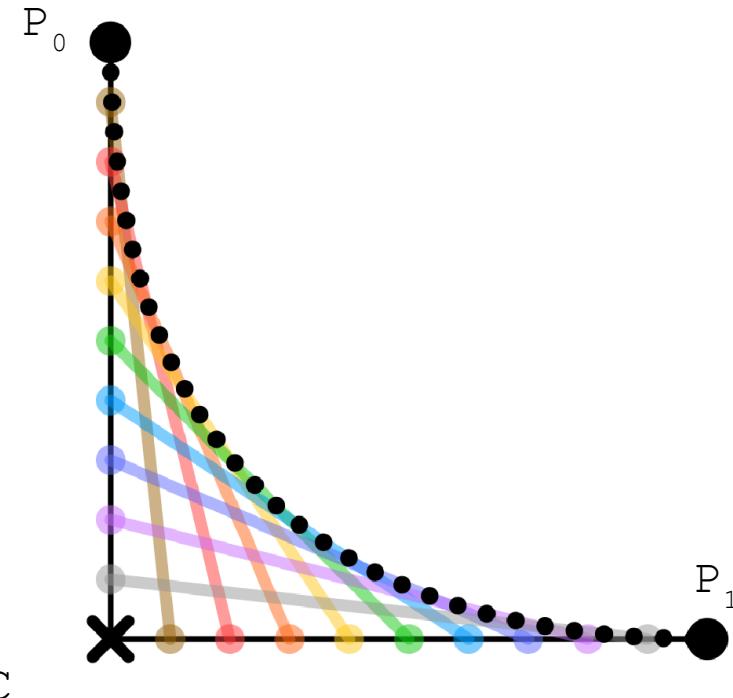
Bezier curves: linear interpolation

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



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 linear interpolation**



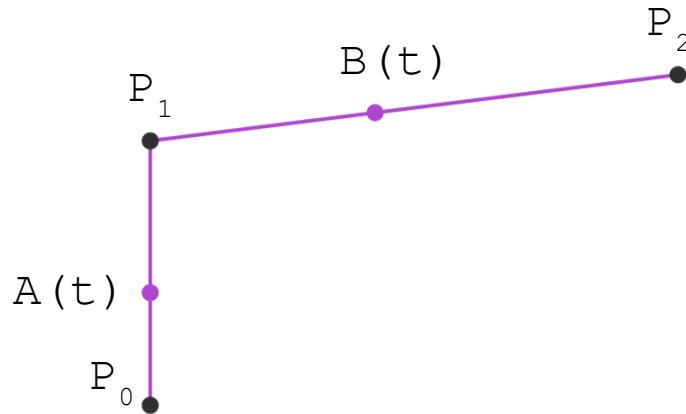
* Independently discovered by Paul de Casteljau and Pierre Bezier for use in French car industry.

Bezier curves: repeated interpolation

Linear interpolation



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

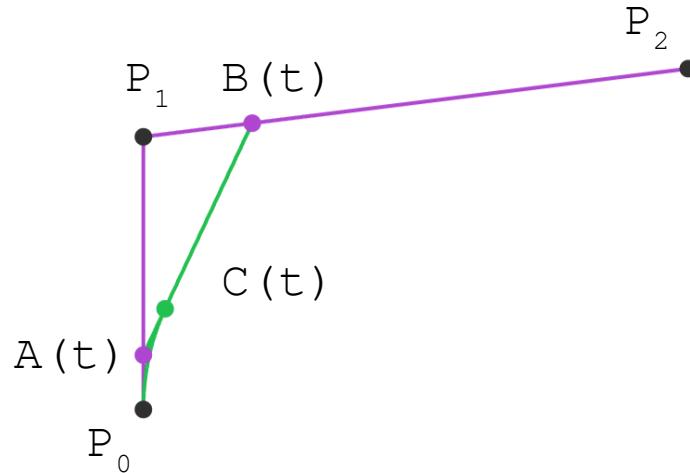


$$A(t) = \text{lerp}(P_0, P_1, t)$$

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

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 $B(t)$



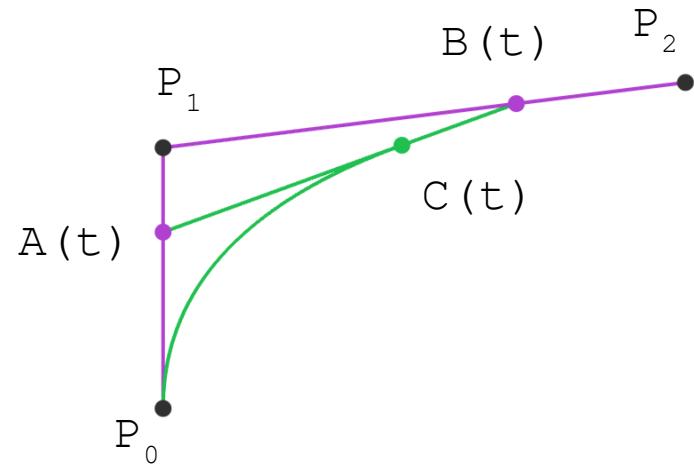
$$A(t) = \text{lerp}(P_0, P_1, t)$$

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

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

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 $B(t)$



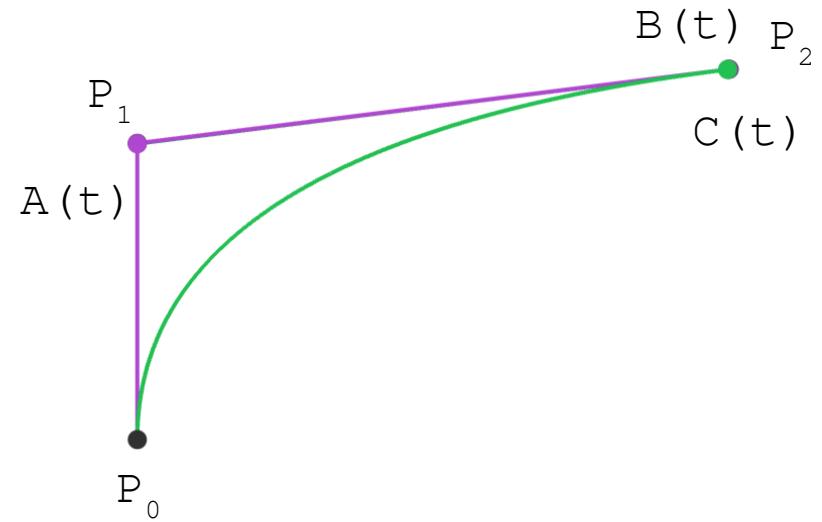
$$A(t) = \text{lerp}(P_0, P_1, t)$$

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

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

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 $B(t)$

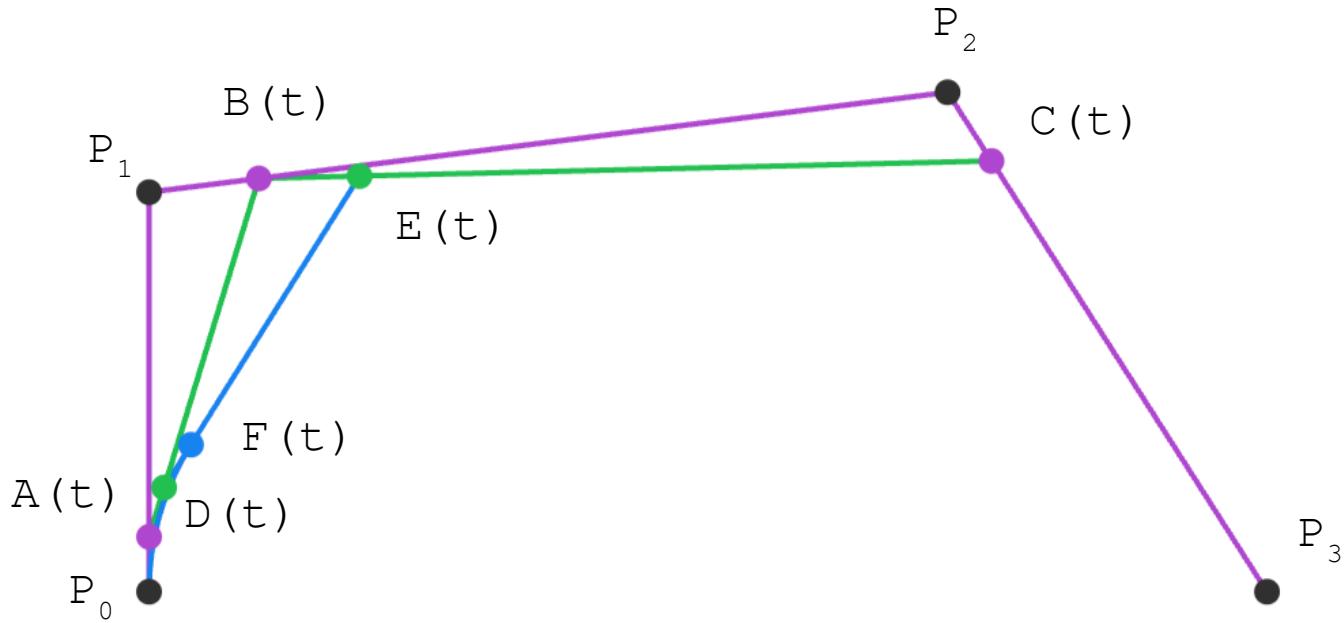


$$A(t) = \text{lerp}(P_0, P_1, t)$$

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

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

Cubic Bezier curve



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

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

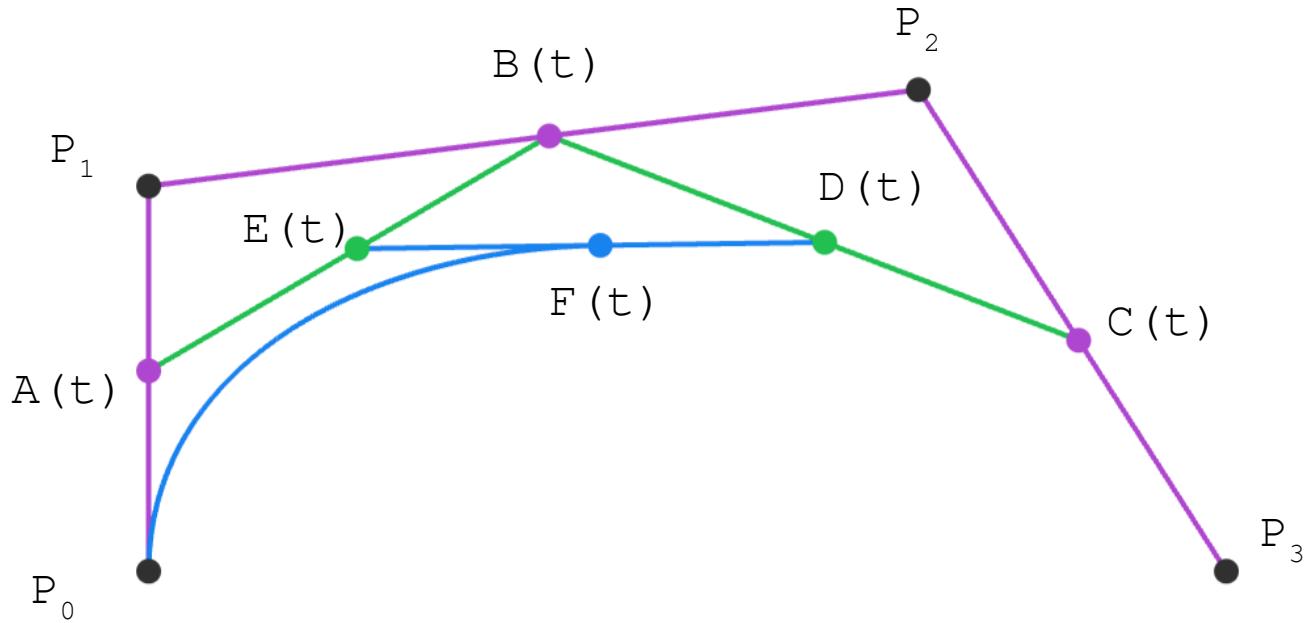
$$C(t) = \text{lerp}(P_2, P_3, t)$$

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

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

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

Cubic Bezier curve



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

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

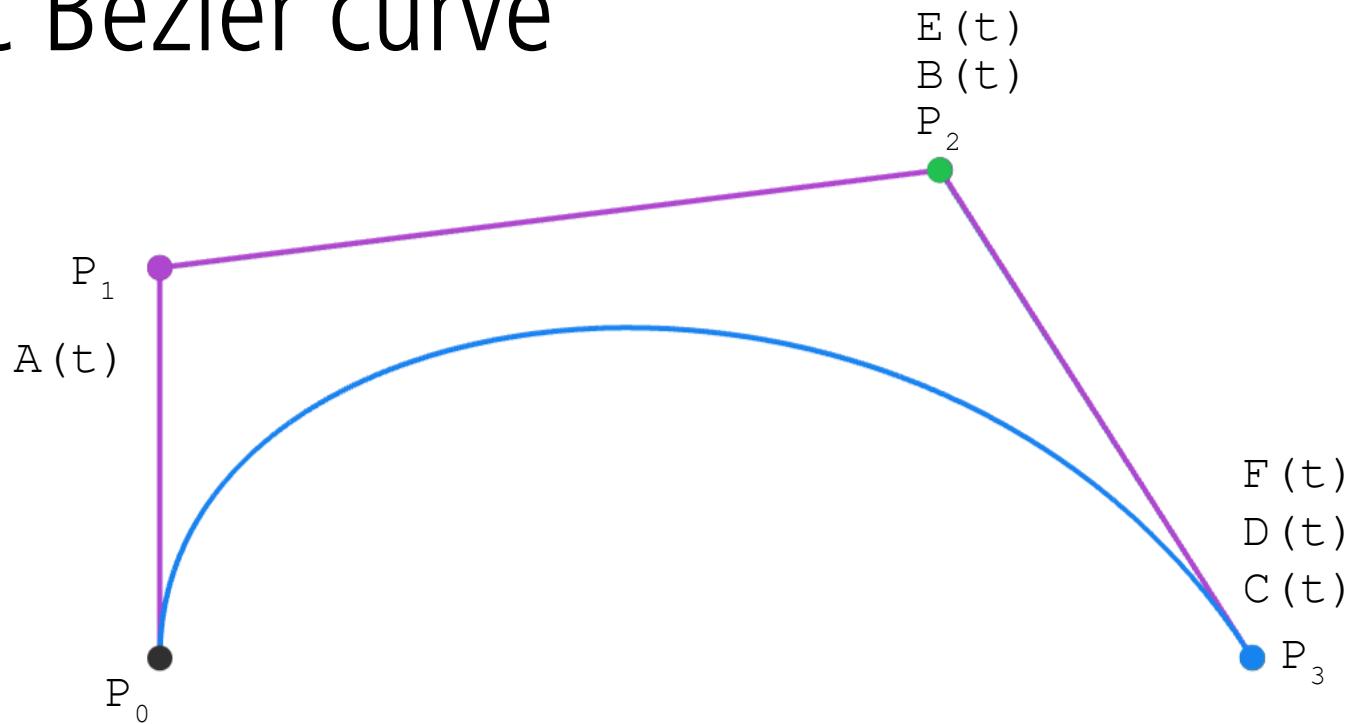
$$C(t) = \text{lerp}(P_2, P_3, t)$$

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

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

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

Cubic Bezier curve



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

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

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

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

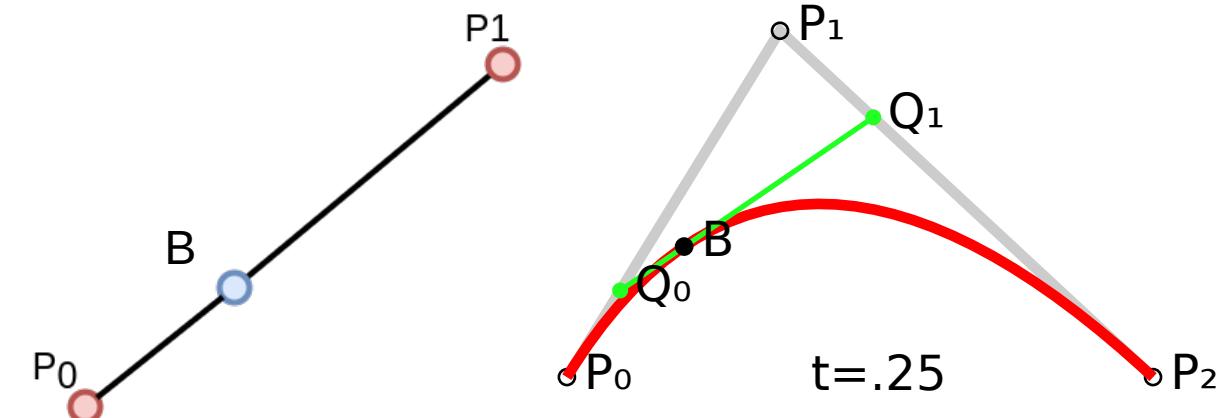
$$E(t) = \text{lerp}(B(t), C(t), t)$$

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

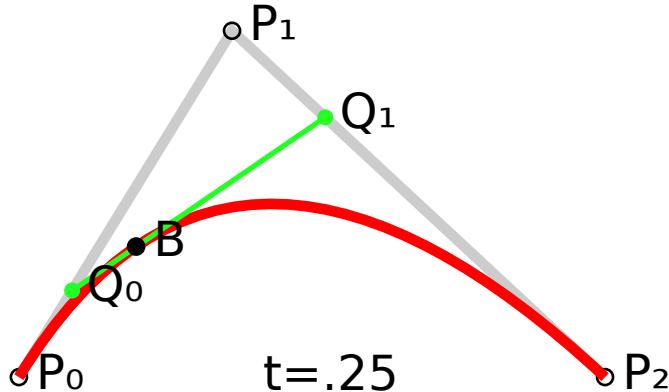
Bezier curves

- 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 and harder to control

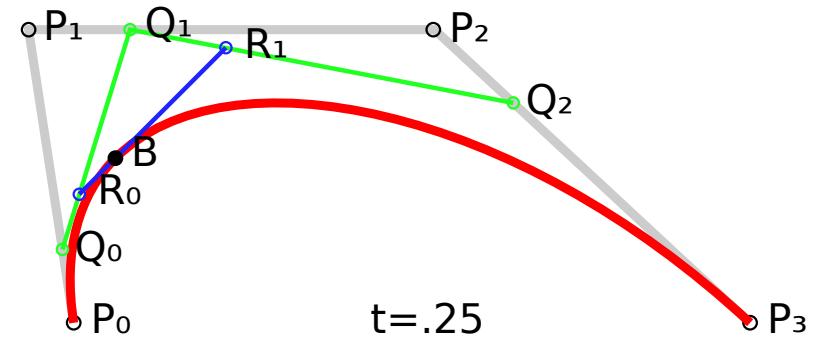
$n = 1 \rightarrow 1^{\text{st}}$ degree, linear Bezier



$n = 2 \rightarrow 2^{\text{nd}}$ 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:

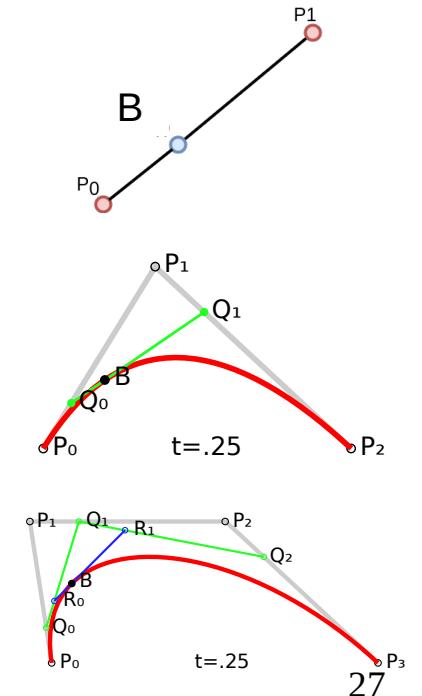
$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1, \quad 0 \leq t \leq 1$$

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:

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

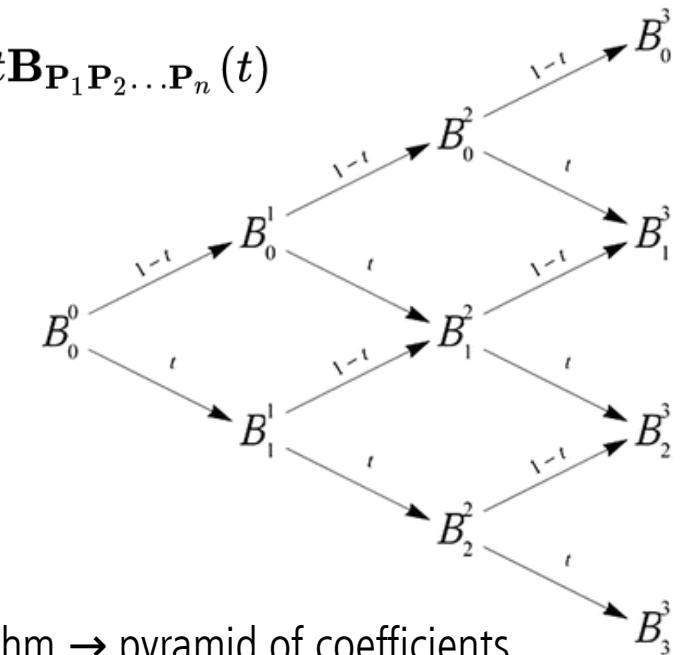


De Casteljau algorithm: recursive formula*

- 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

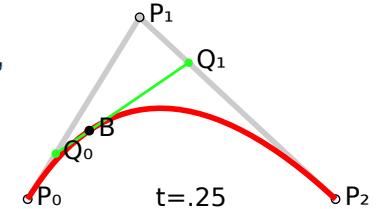
Bezier curve: Bernstein form

- 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**:

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

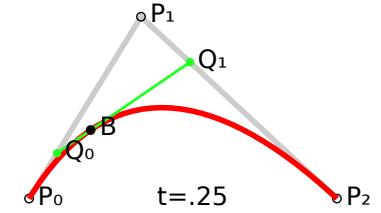


Bezier curve: Bernstein form

- Example: $n = 2$ (quadratic)

Bernstein
form

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



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

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

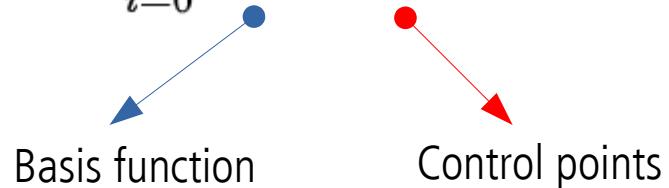
"Repeated
interpolation"

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

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:

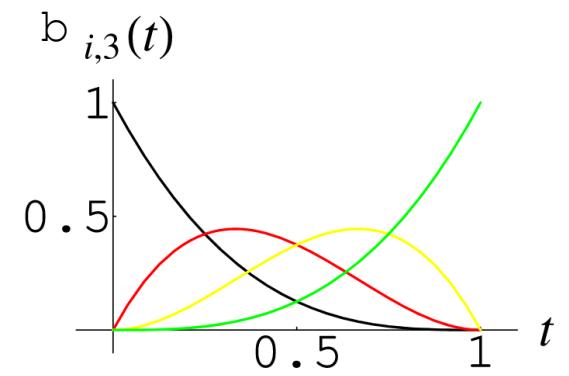
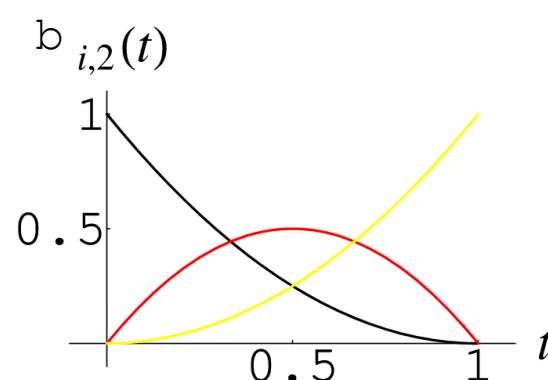
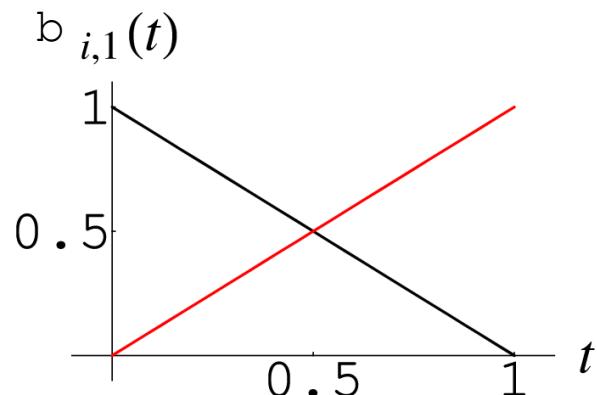
$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, \dots, n$$

Basis functions

- Defines influence (weighting) of each control point as parameter t changes.
 - Example: when t increases, blending weight for P_0 decreases and blending weight for P_1 increases, etc.

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

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



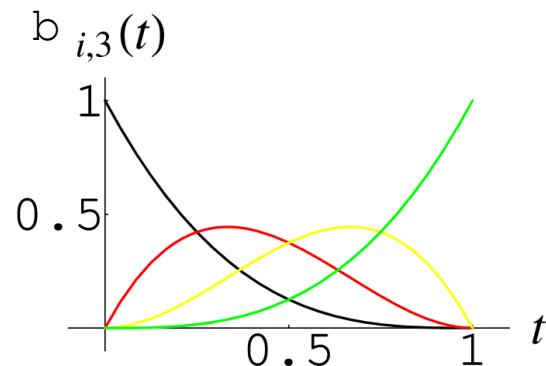
Bernstein polynomials aka blending functions: linear, quadratic, cubic

Bernstein polynomials

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

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

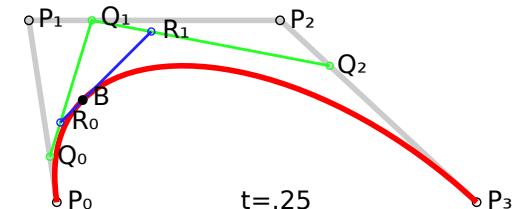
"Polynomials are in $[0, 1]$ when t in $[0, 1]$ "



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

"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.

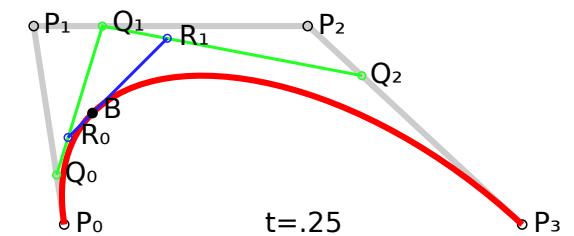


Bezier curve: matrix representation

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

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

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



Bezier curves: good properties

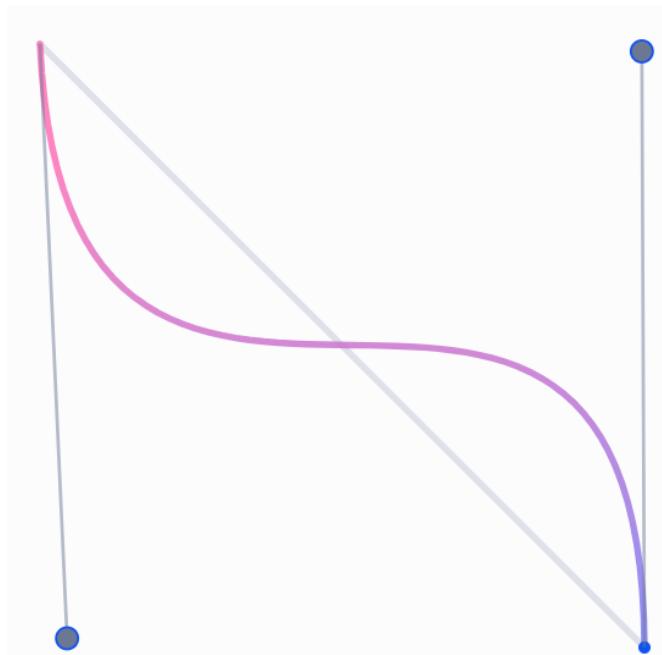
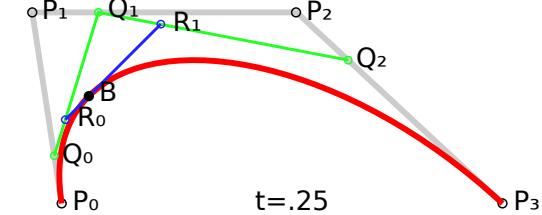
- **Intuitive theory**, good for understanding curves: repeated interpolation
- **Compact form**: Bernstein form and matrix representation
- **Tangent vector at curve point** can be calculated easily (derivation of polynomial)
- **Easy manipulation of large number of points**
 - Large number of points can be generated from parametric description
 - If rotation of those points is needed, then curve (few control points) are rotated and then points are generated

Bezier curves: problems

- **Non-interpolating:** 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

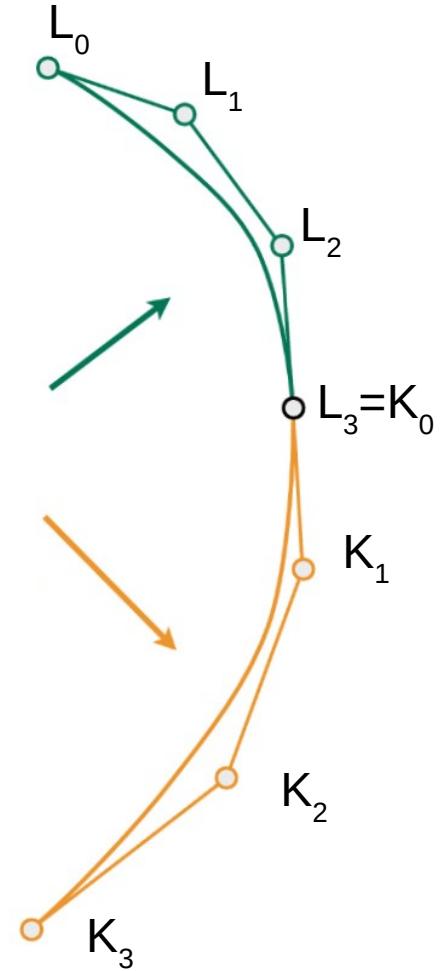
Joining cubic Bezier curves

- In practice, cubic Bezier curves are concatenated to form larger spline
 - Cubic curves are lowest degree curves that can describe complex form, e.g., S-shaped curve called **inflection**
 - Lower computation complexity
 - Easy to manipulate with
 - Resulting composite curve will pass through the set of points



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

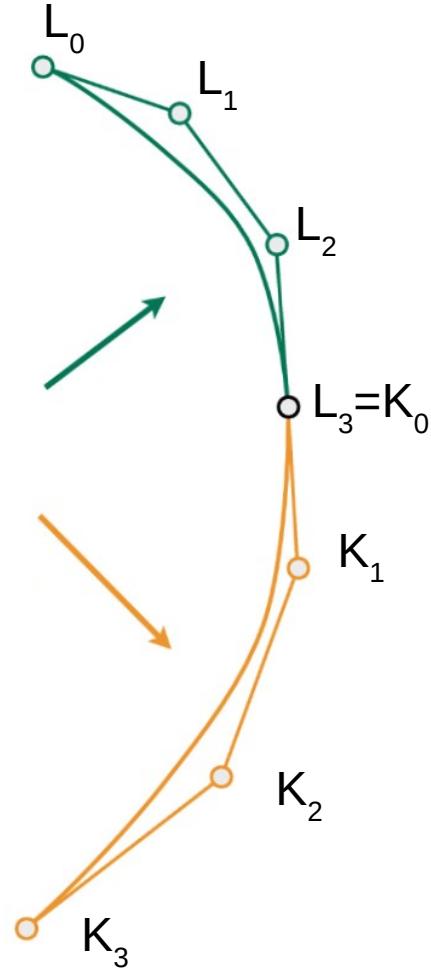


Joining cubic Bezier curves

- Composite curve is called **piecewise Bezier curve, $p(t)$**
 - First segment is $p(t)$, where t in $[0, 1]$
 - Second segment is $p(t')$, where $t' = (t - t_1) / (t_2 - t_1)$, t in $[0, 1]$
- Two curves connected just using $L_3 = K_0$ will not be smooth at joint.
 - Improved smoothness can be achieved by **joining tangentially**: tangent of first and second curve at joint must be parallel*:

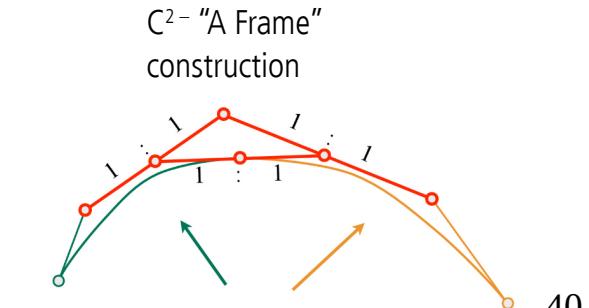
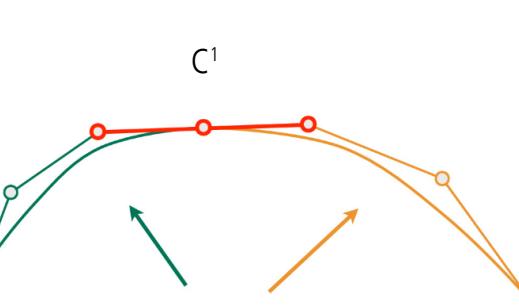
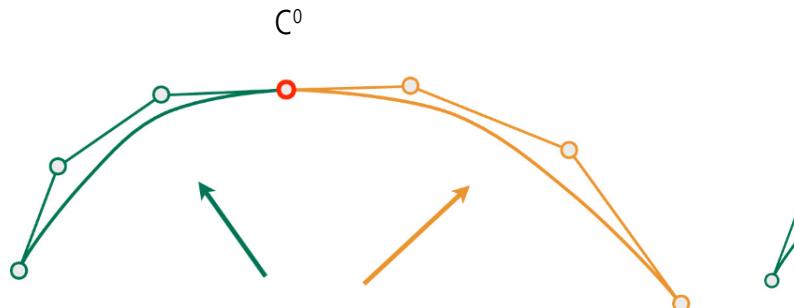
$$(K_1 - K_0) = c(L_3 - L_2), c > 0$$

$$c = \frac{t_2 - t_1}{t_1 - t_0}$$



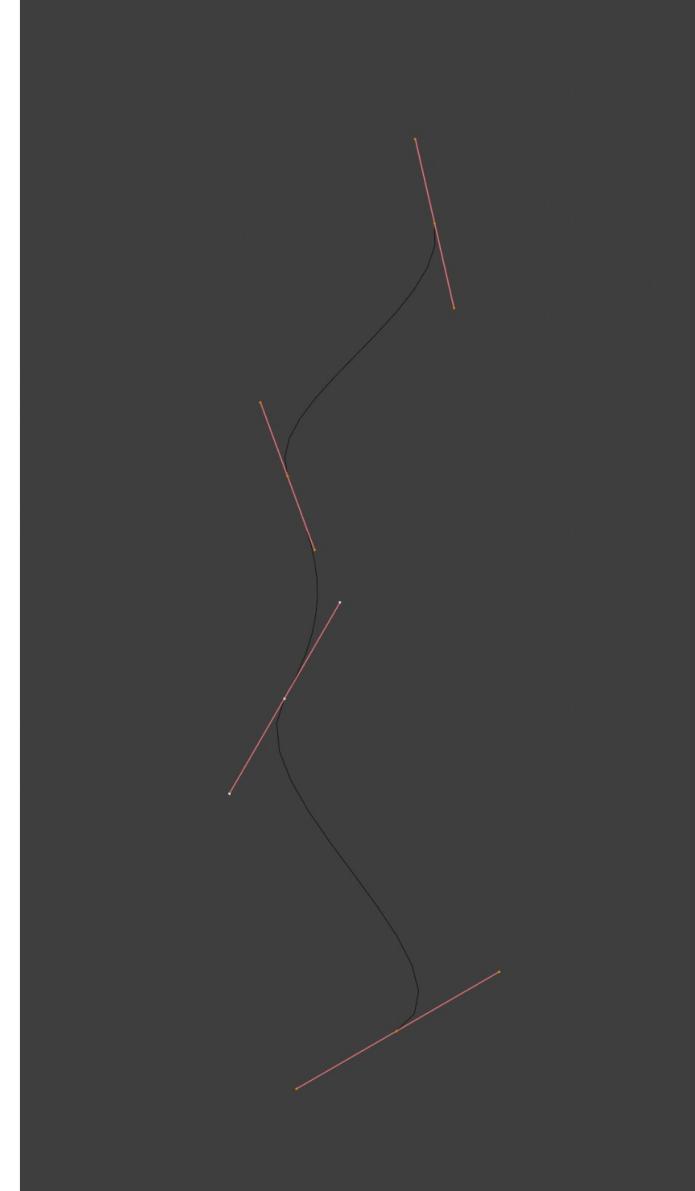
Joined curves continuity

- 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
 - G^2 – **curvature continuity** - tangent vectors from curve segments that meet at joint should be of same length and rate of length change

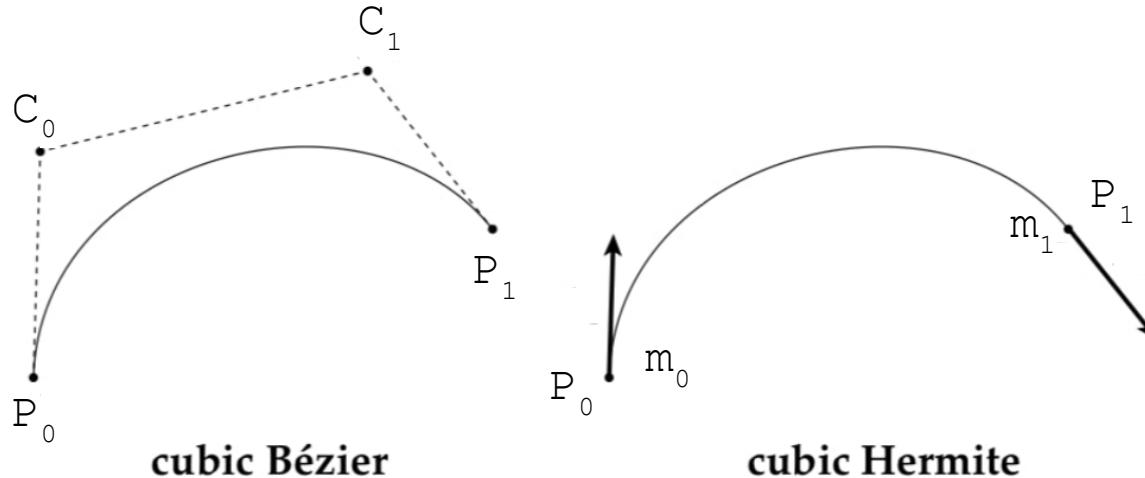


Parametric curves

- Bezier curve
 - Intuitive theory: repeated interpolation de Casteljau algorithm
 - Formalization: Bernstein and matrix form
 - Concatenating cubic curves into composite curves
 - Curve continuity
- **Hermite curve**
 - In practice, **cubic Hermite curve** is used for simpler authoring
- Catmull-Rom spline
- B-Splines

Cubic Hermite curve

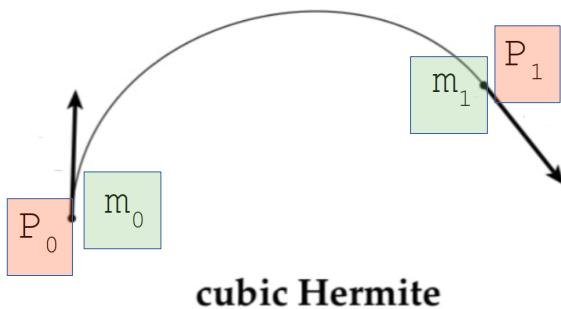
- Cubic Hermite interpolation curve:
 - Interpolates points P_0 and P_1 – **starting and ending control points**
 - Interpolation is **controlled with tangent vectors**: m_0 at P_0 and m_1 at P_1
 - Tangent vector magnitude and direction influence the curve shape



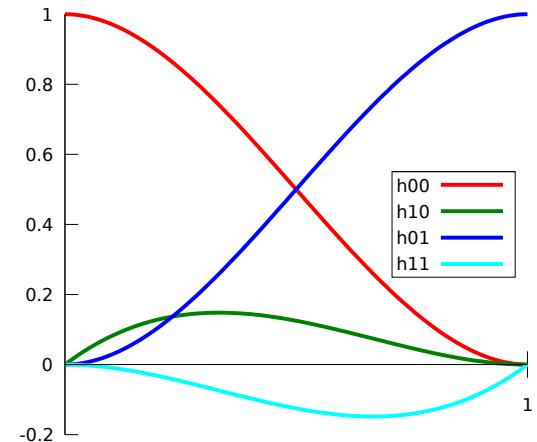
Cubic Hermite curve

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

$$p(t) = (2t^3 - 3t^2 + 1)p_0 + (t^3 - 2t^2 + t)m_0 + (-2t^3 + 3t^2)p_1 + (t^3 - t^2)m_1$$



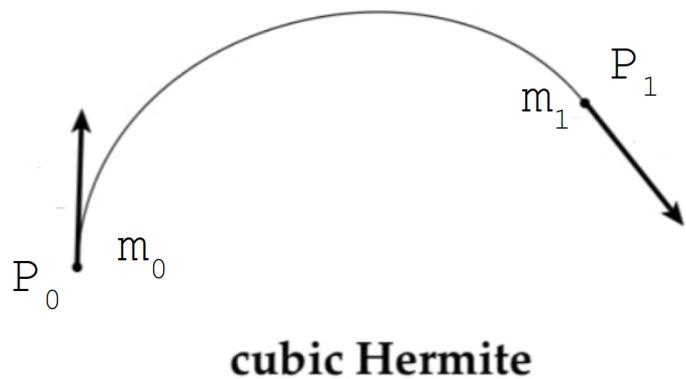
- Similarly as discussed with Bezier: Bernstein form with Hermite basis function exists



Cubic Hermite curve

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

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

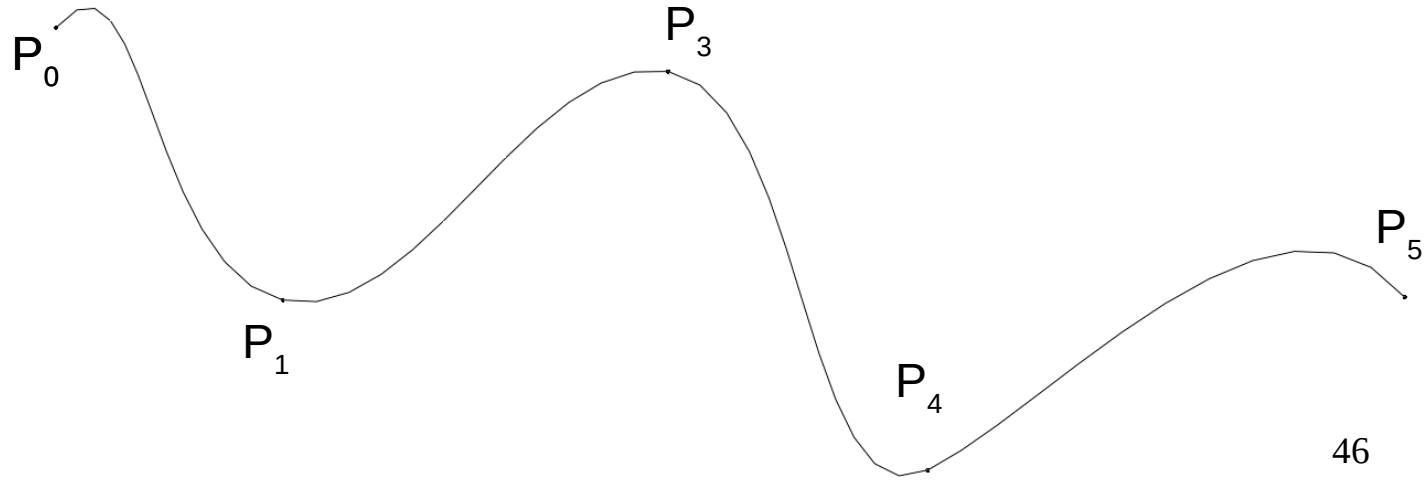


$$p(t) = (t^3 \ t^2 \ t \ 1) \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ m_0 \\ m_1 \end{pmatrix}$$

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)

How to calculate the tangent vectors in given points?
Solution: Catmull-Rom spline

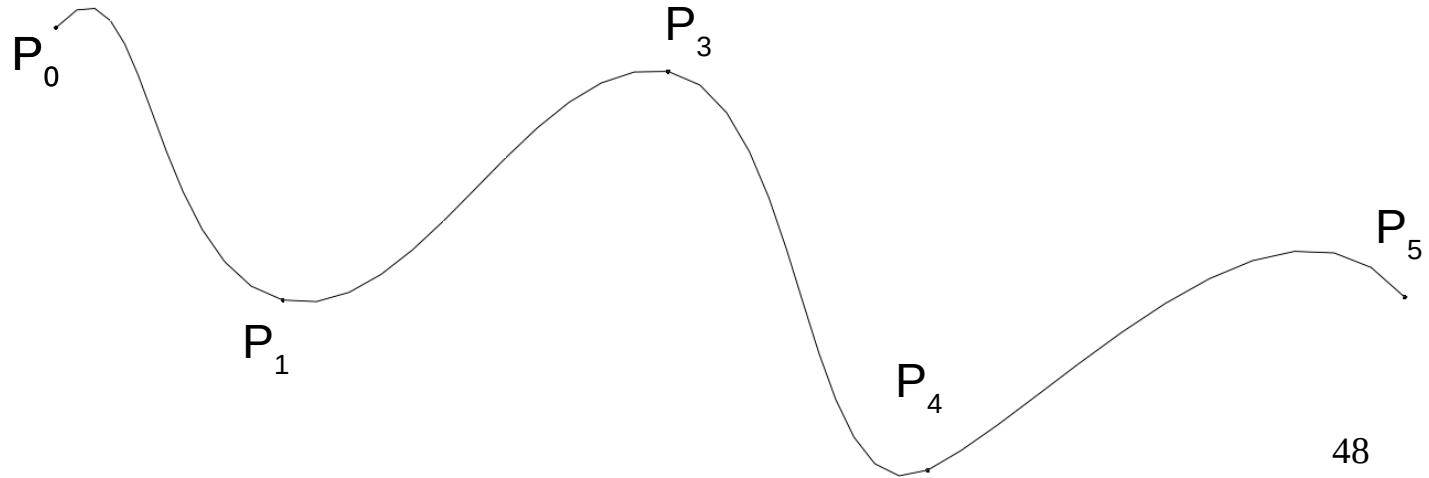


Parametric curves

- Bezier curve
- Hermite curve
- **Catmull-Rom spline**
 - Use Hermite curve to build a spline
- B-Splines

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**



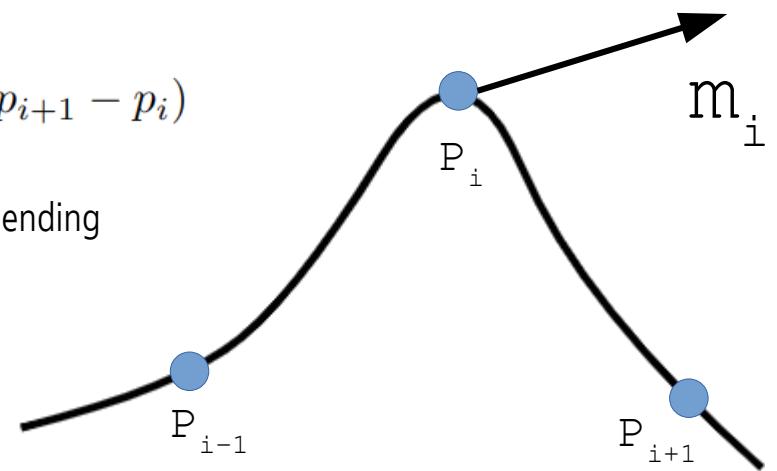
Kochanek-Bartels method

- 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

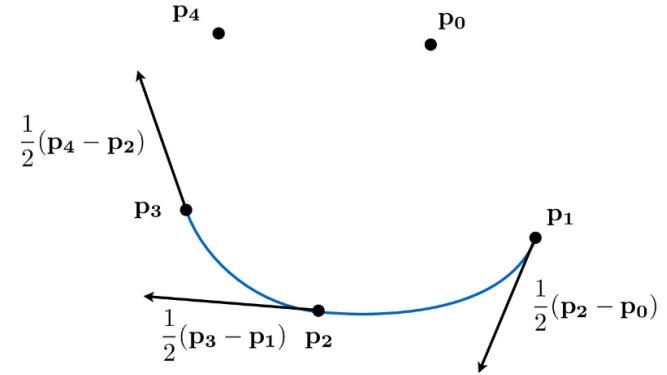
Bias parameter - b : direction of tangent



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

Parametric curves

- Bezier curve
- Hermite curve
- Catmull-Rom spline
- B-Splines
 - Next to Catmull-Rom spline, B-splines are widely used

B-spline

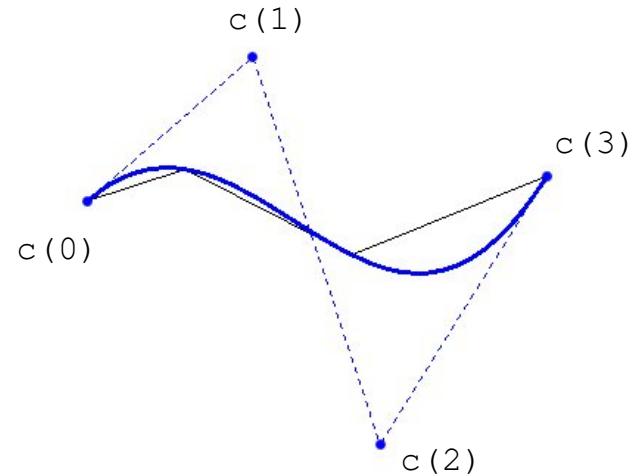
- Types:
 - Uniform: uniformly space control points
 - Non-Uniform
- Parameterized function of \mathbf{x} . (Similarly to Bezier curve)

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

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

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

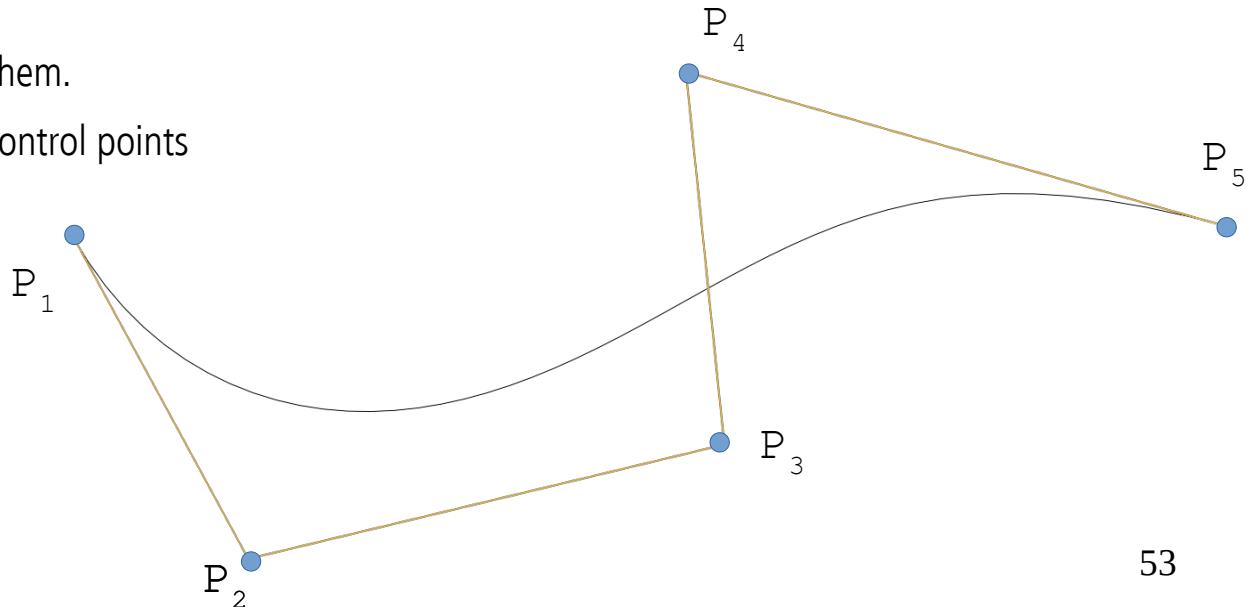
$$\beta_3(x) = \begin{cases} 0, & |x| \geq 2 \\ \frac{1}{6} \cdot (2 - |x|)^3, & 1 \leq |x| < 2 \\ \frac{2}{3} - \frac{1}{2}|x|^2 \cdot (2 - |x|), & |x| < 1 \end{cases}$$



Cubic B-spline: non-interpolating
<https://docs.bentley.com/LiveContent/web/ProStructures%20Help-v6/en/SECurvesPlaceBsplineCurve.html>

Uniform cubic B-spline

- 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
- Cubic B-spline is **non-interpolating**
 - Passing near control points, not through them.
 - No guarantee that it is interpolating the control points



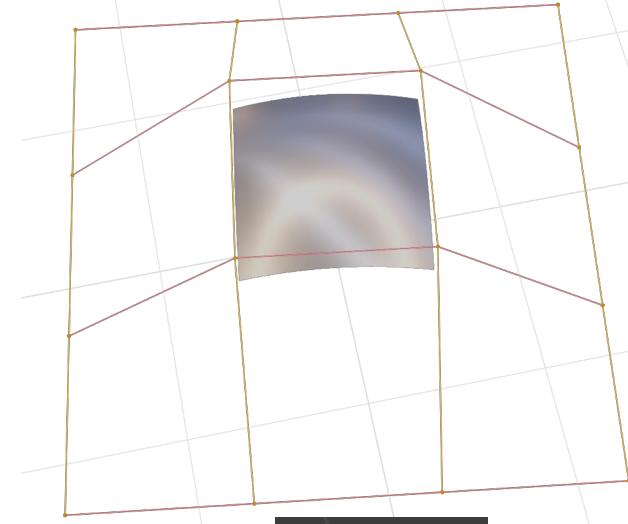
Limitations and generalizations

- Representing circle is not possible
 - Solution, a generalization is to introduce additional coordinate to B-spline → **rational B-spline**
- 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

Parametric surfaces

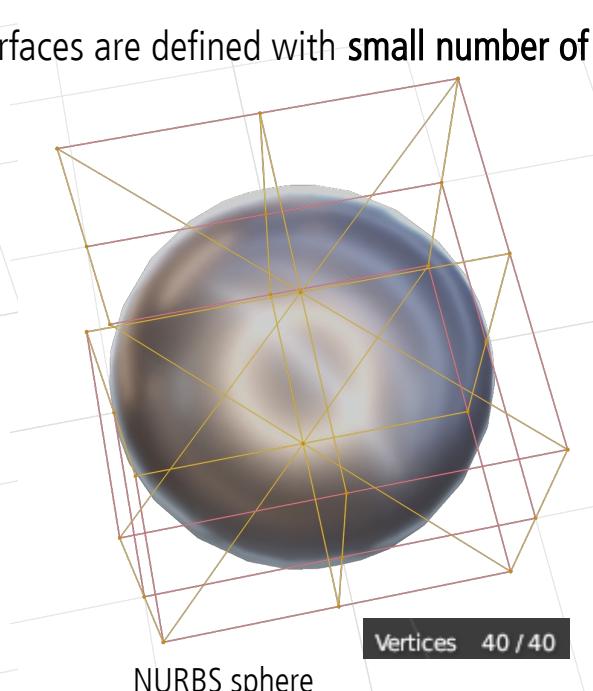
Parametric surfaces

- 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**
- Similarly as parametric curves, parametric surfaces are defined with **small number of control points and parametric equation**



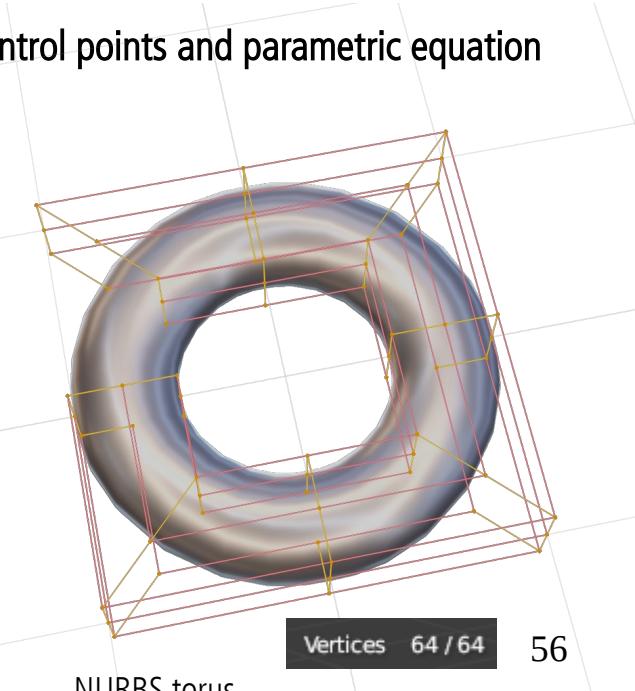
NURBS surface

Vertices 16 / 16



NURBS sphere

Vertices 40 / 40



NURBS torus

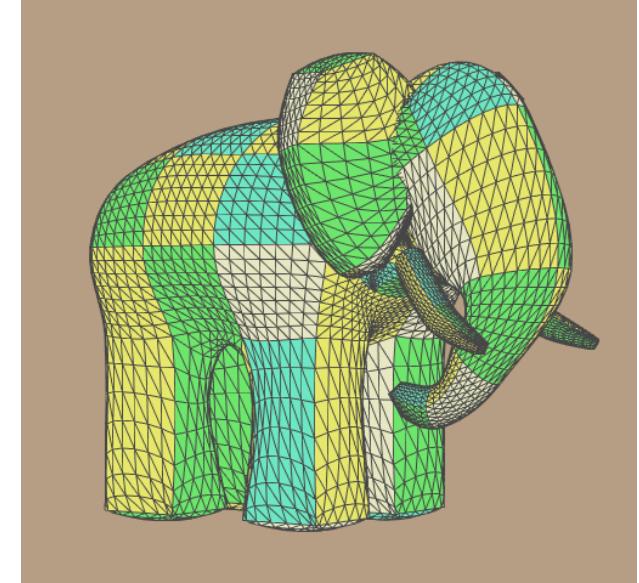
Vertices 64 / 64

Parametric surfaces

- Common types
 - Bezier surfaces
 - B-spline surfaces



NURBS Renderman: <https://rmanwiki.pixar.com/display/REN24/NURBS>

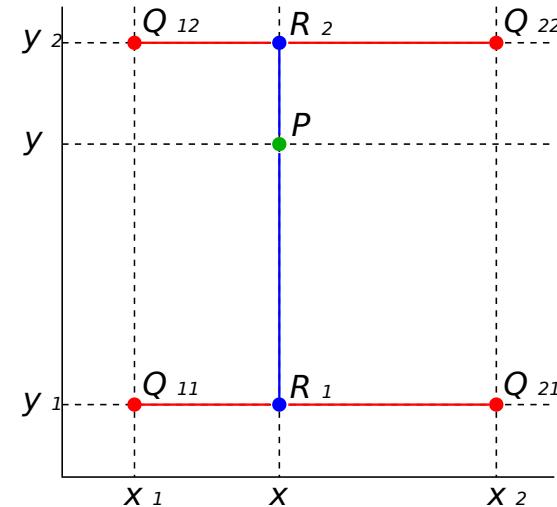


Ed Catmull's "Gumbo" model, composed from Bezier patches
https://en.wikipedia.org/wiki/B%C3%A9zier_surface

Bezier surface: bilinear interpolation

- Bezier curve is extended so it has **two parameters (u,v)** which define surface
- As linear Bezier curve is defined with linear interpolation, **planar Bezier patch** is defined with **bilinear interpolation**.

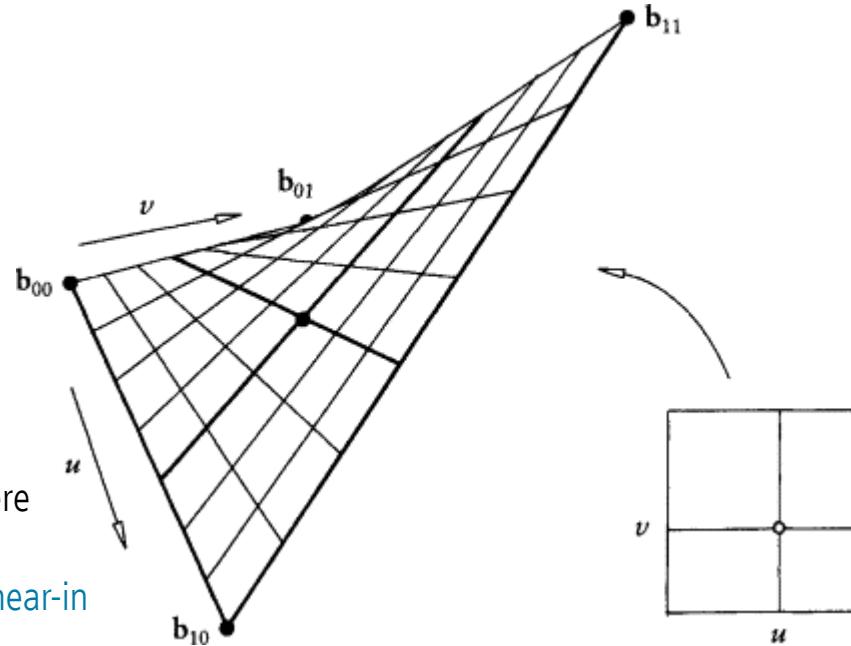
$$\begin{aligned} R_1(u) &= \text{lerp}(Q_{11}, Q_{12}, u) \\ R_2(u) &= \text{lerp}(Q_{12}, Q_{22}, u) \\ P(u, v) &= \text{lerp}(R_1(u), R_2(u), v) \end{aligned}$$



Bilinear interpolation

Bezier surface: patch

- Planar Bezier patch $p(u, v)$ is simplest, non-planar parametric surface
- It has **rectangular domain** and thus resulting surface is called a **patch**

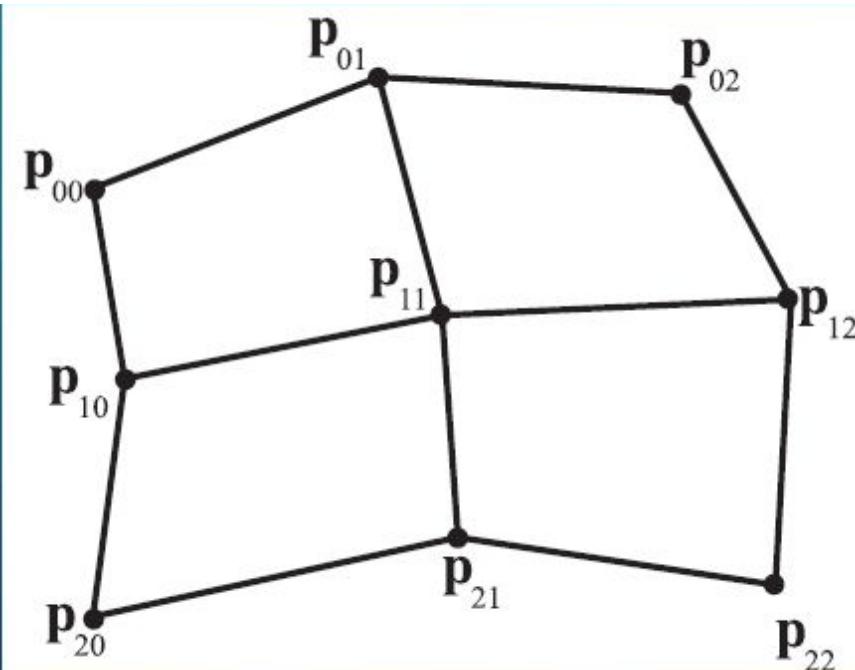


Bilinear interpolation defines planar Bezier patch $p(u,v)$ where (u,v) are in $[0,1]$.

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

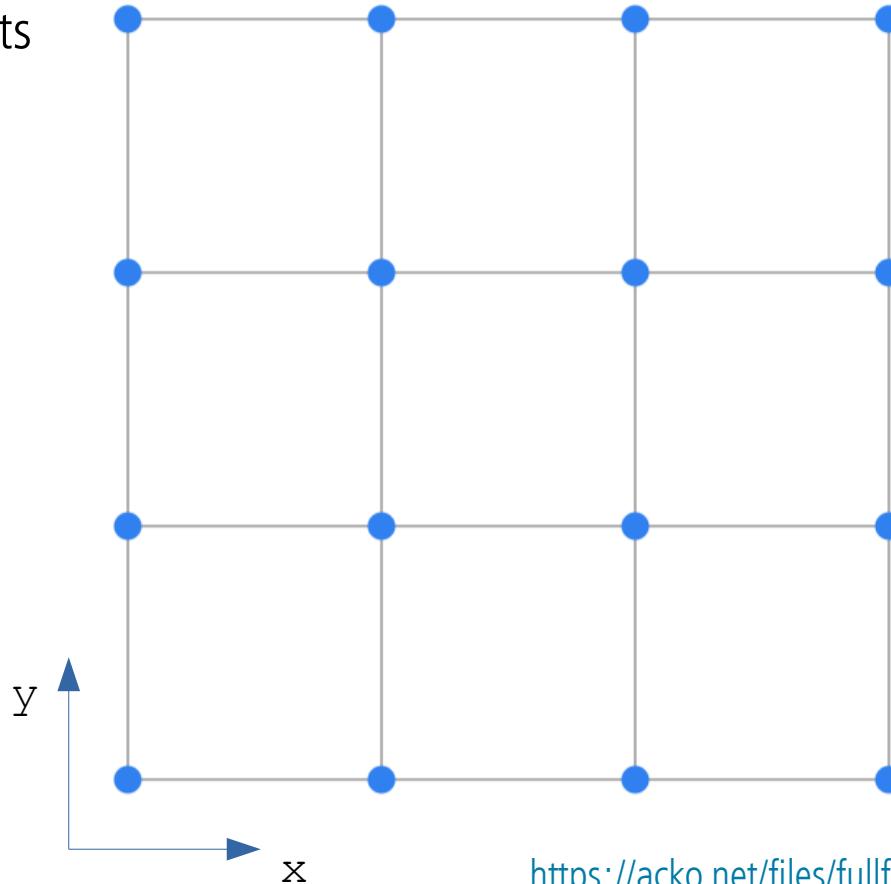
Bezier surface

- Bezier surface is achieved by adding more points to bilinear interpolation (similarly as we added more points to linear interpolation to obtain Bezier curve)
- Repeated bilinear interpolation is extension of de Casteljau's algorithm to Bezier patches.
- Example: biquadratic Bezier surface
 - 9 control points



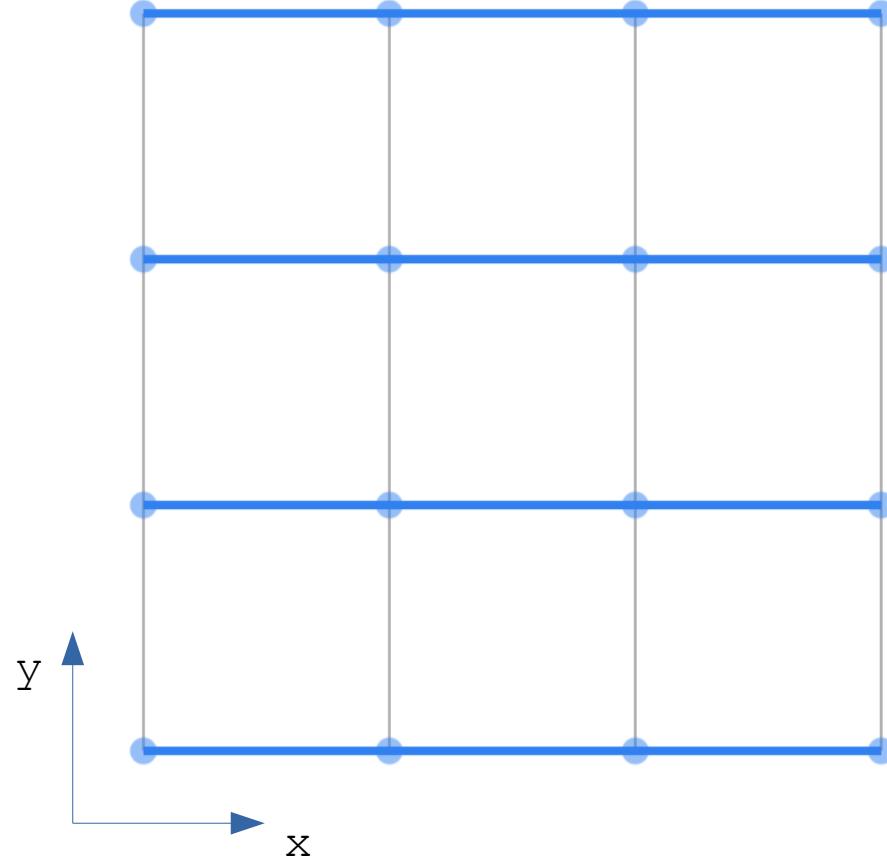
Bezier surface: repeated bilinear interpolation

- Starting control points



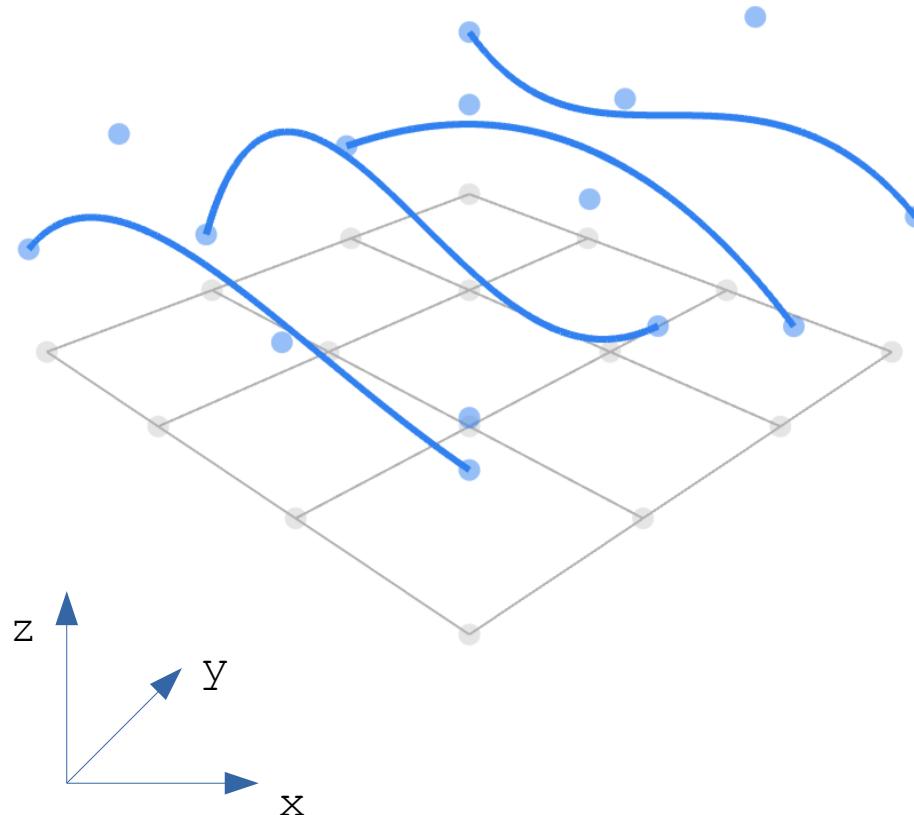
Bezier surface: repeated bilinear interpolation

- Linear interpolation



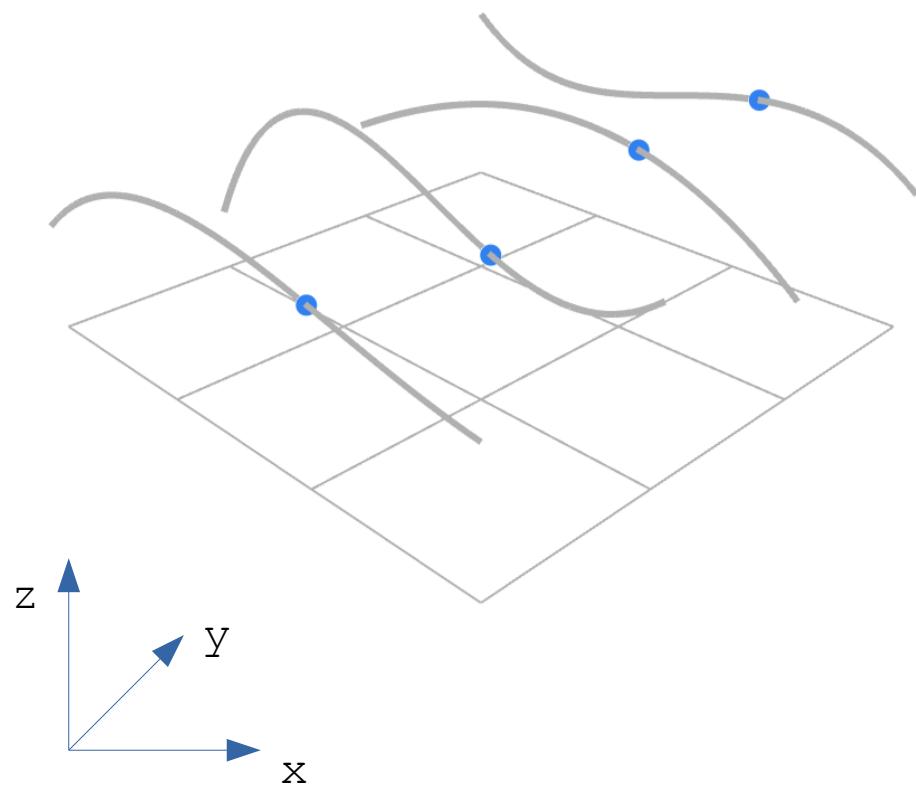
Bezier surface: repeated bilinear interpolation

- Repeated linear interpolation
 - Bezier cubic curves



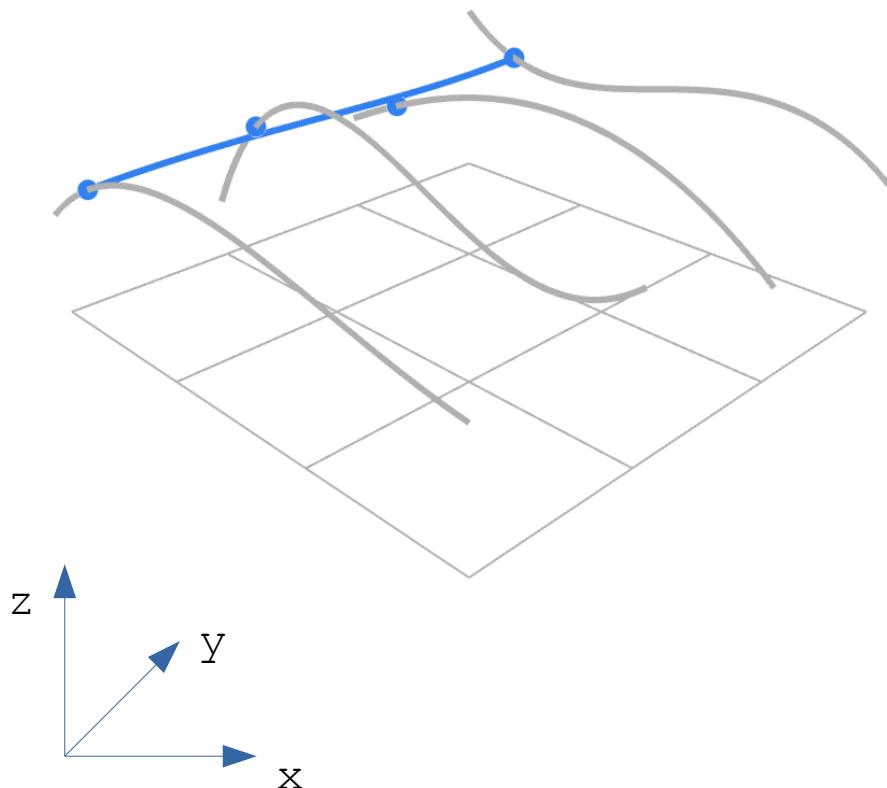
Bezier surface: repeated bilinear interpolation

- Repeated linear interpolation



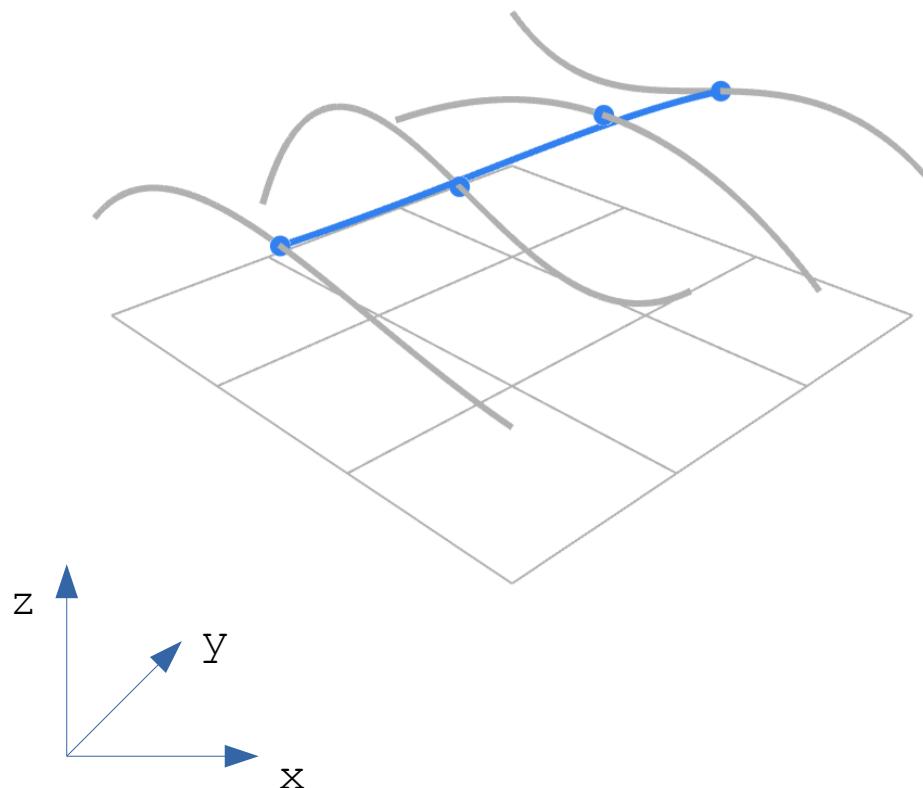
Bezier surface: repeated bilinear interpolation

- Repeated bilinear interpolation



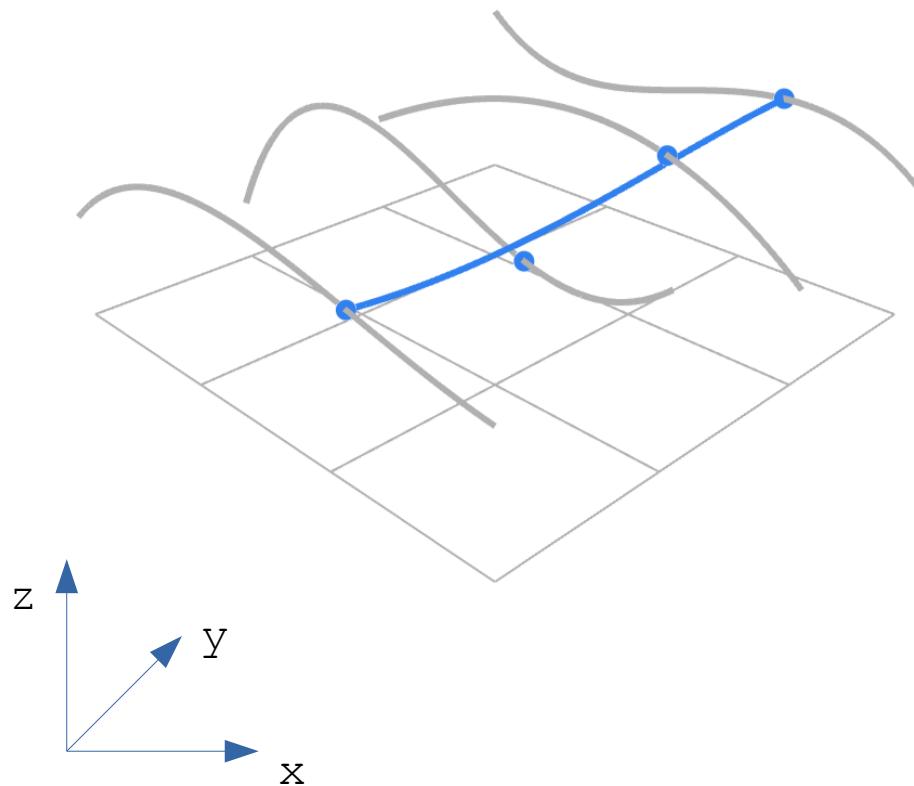
Bezier surface: repeated bilinear interpolation

- Repeated bilinear interpolation



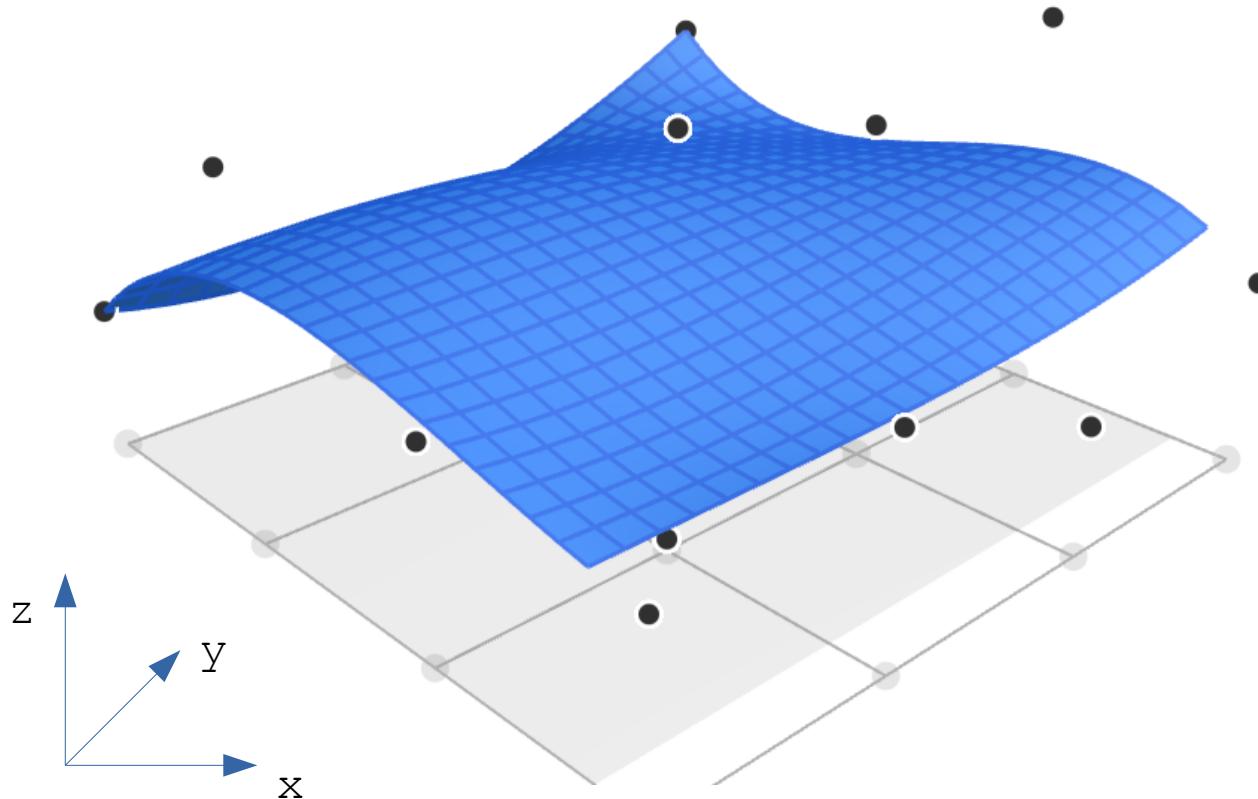
Bezier surface: repeated bilinear interpolation

- Repeated bilinear interpolation



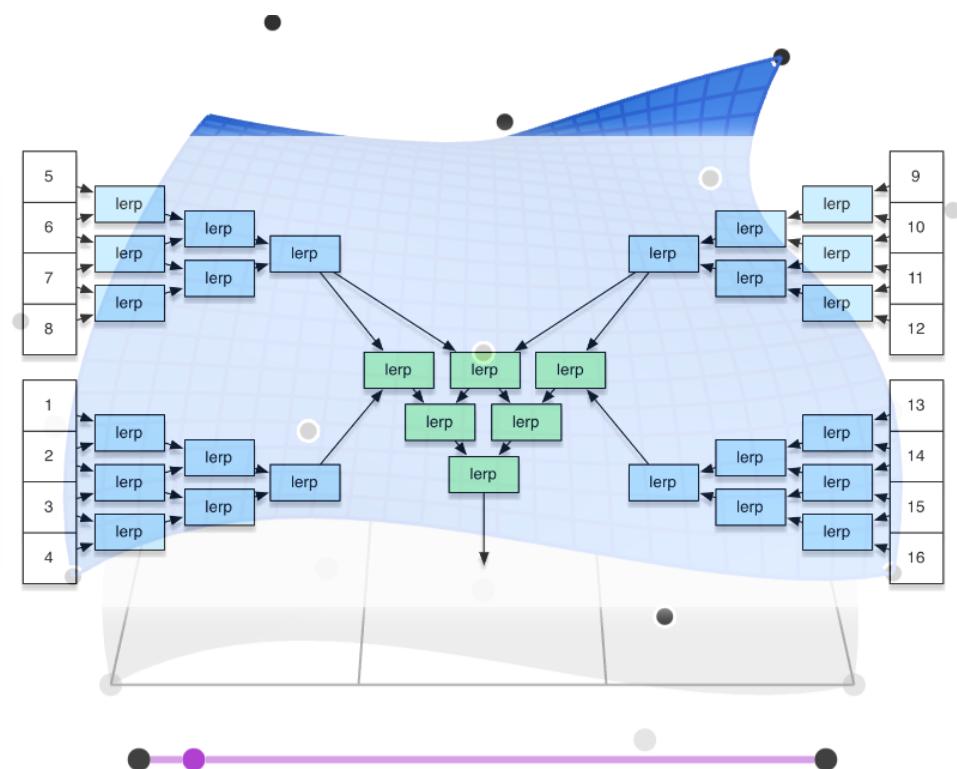
Bezier surface: repeated bilinear interpolation

- Bicubic Bezier surface



Bezier surface

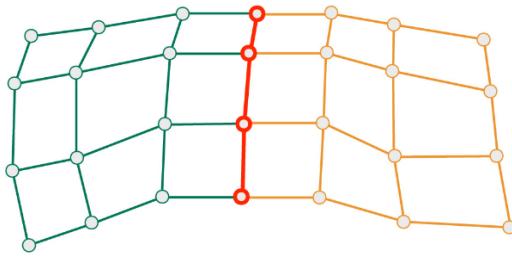
- Repeated bilinear interpolation is extension of **De Casteljau algorithm** to patches
- Point on Bezier Patch can be described in **Bernstein form** using Bernstein polynomials



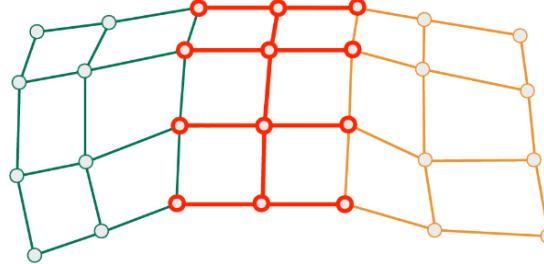
- Bicubic Bezier surface defined with 16 control points

Bezier surface: continuity

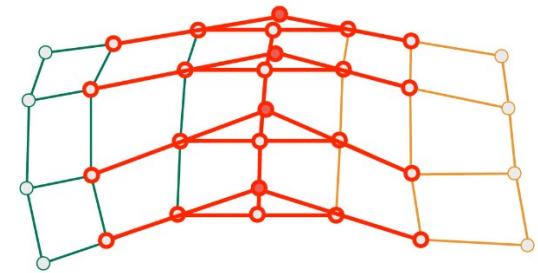
Surface continuity



C^0 – positional continuity –
joined at the same point



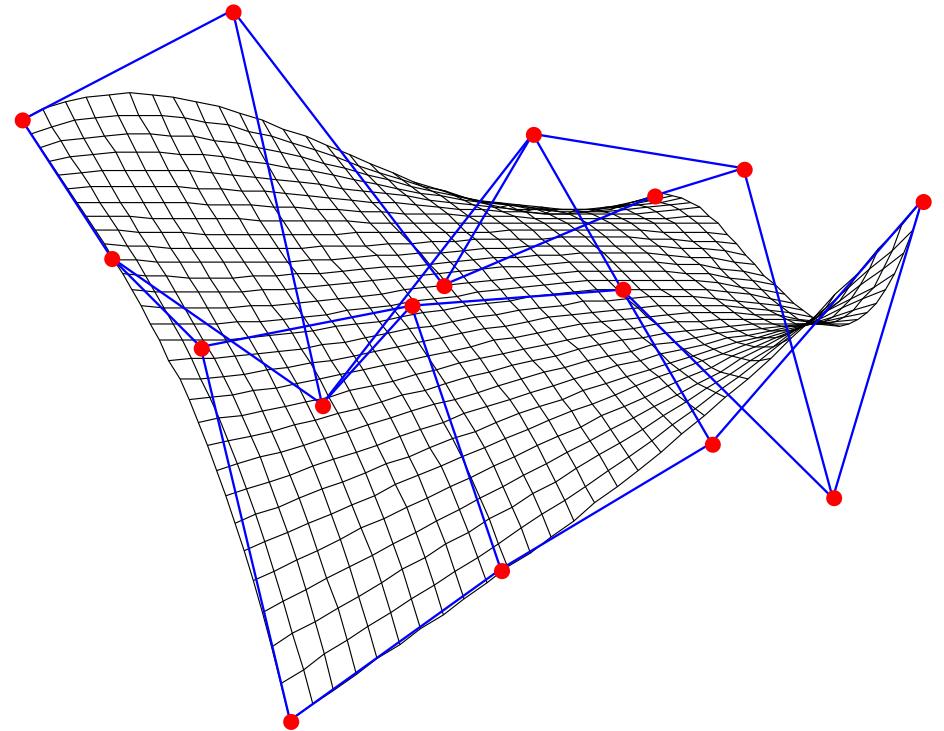
C^1 – velocity continuity -
derivation of any point
(including joints) must be
continuous



C^2 - acceleration continuity -
first and second derivatives are
continuous functions

Bezier surface properties

- **Non-interpolating:** passes through only corner control points
- **Boundary of the patch:** Bezier curve of degree n
- **Tangents at border points:** described with Bezier curve at border points – two tangents: for u and v direction
 - Derivative is straightforward (derivation of polynomials)
- **Patch lies within convex hull of its control points**
- Arbitrary number of points on patch can be generated
 - Prior generation, transformation on patch can be done which is efficient than transformation on generated points
- Extension: **rational Bezier patches**
 - <https://theses.hal.science/tel-01064604/document/>

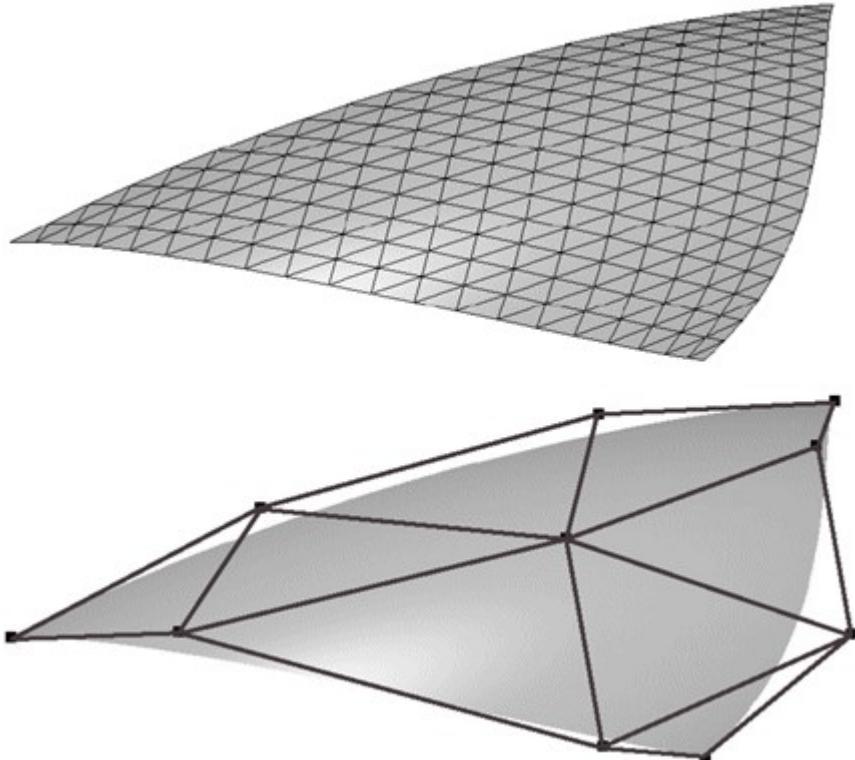


Parametrized surfaces based on Bezier surface

- Bezier triangles
- Point-Normal (PN) Triangles
- Phong tessellation

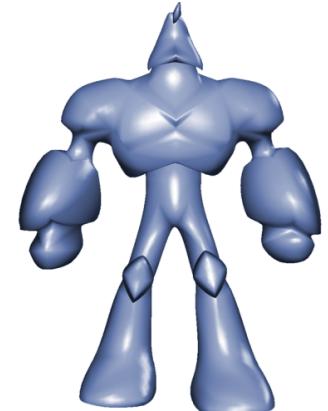
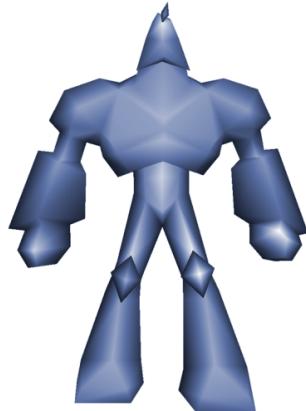
Bezier triangles

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



Point-Normal (PN) Triangles

- Given triangle mesh with normals at each vertex, the goal is to construct smoother surface using Bezier triangles
- Properties:
 - Improves mesh shading and silhouettes by creating **curved surface to replace each triangle**
 - Creases in PN triangles are hard to control
 - Continuity between Bezier triangles is C_0 but looks acceptable for certain applications

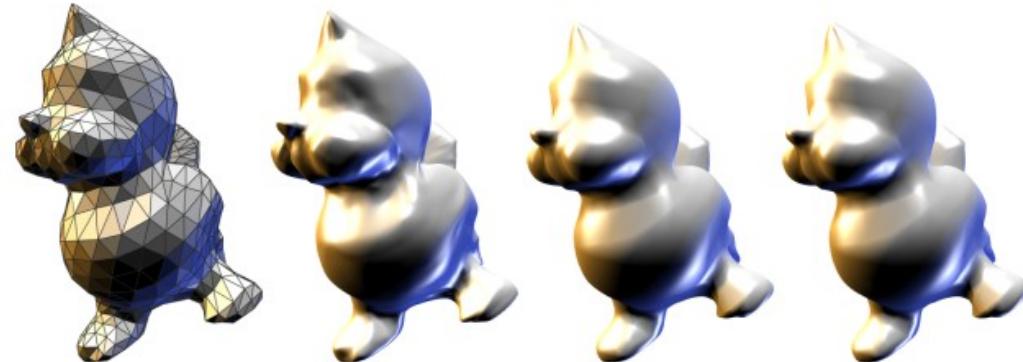


Comparison of original mesh and PN triangle counterpart
<https://www.gamedeveloper.com/programming/b-zier-triangles-and-n-patches>
<https://alex.vlachos.com/graphics/CurvedPNTriangles.pdf>

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

Phong tessellation

- Similar as PN triangles, given the triangle points with normals, construct smoother 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

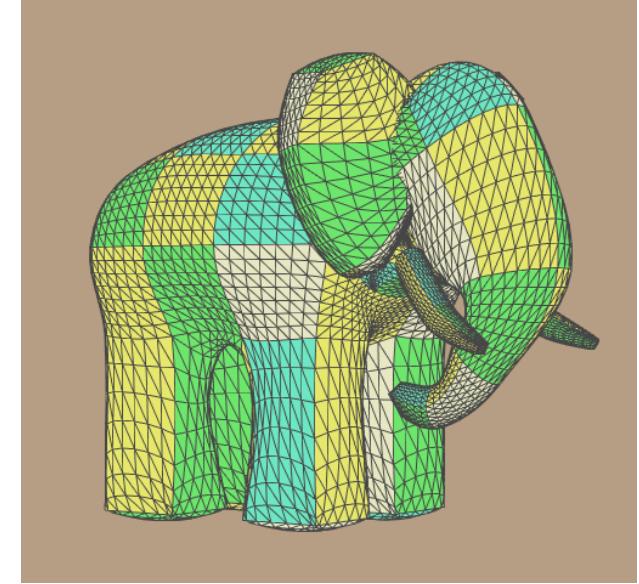
<https://perso.telecom-paristech.fr/boubek/papers/PhongTessellation/>

Parametric surfaces

- Common types
 - Bezier surfaces
 - B-spline surfaces



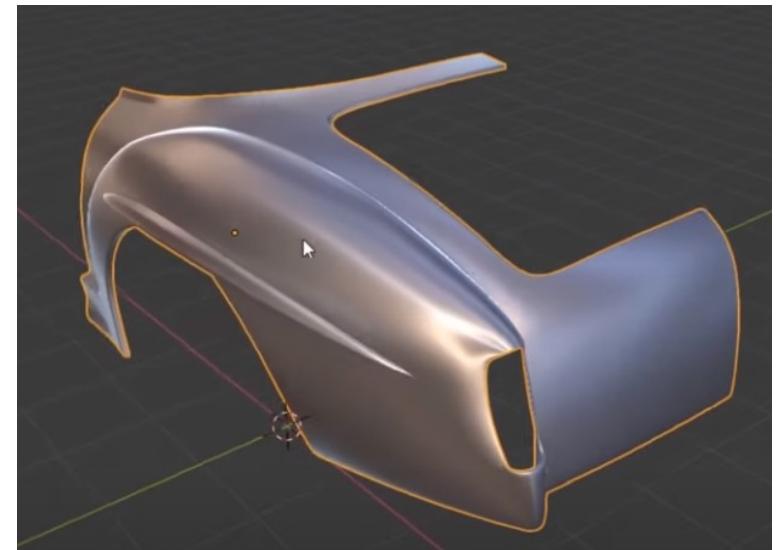
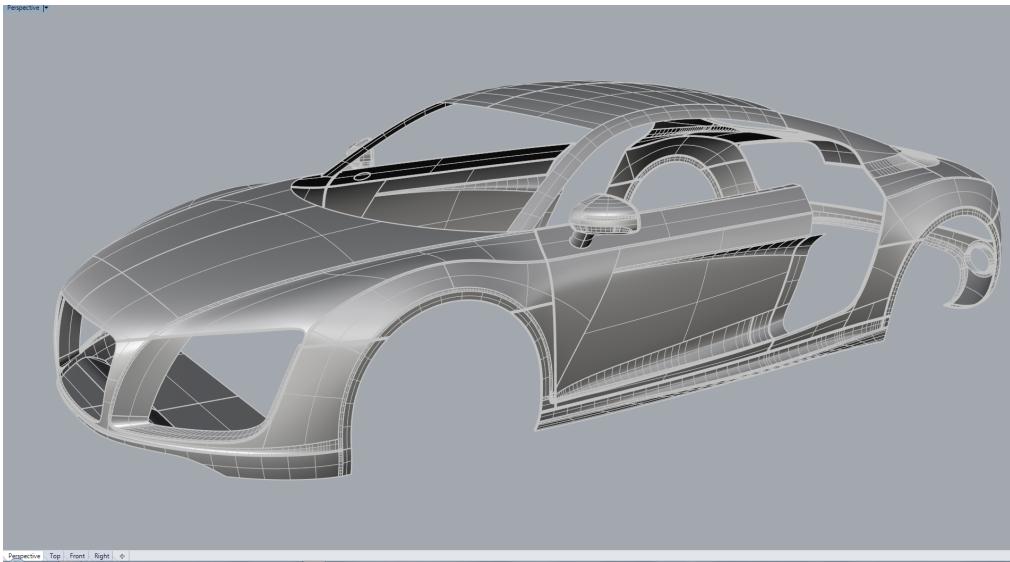
NURBS Renderman: <https://rmanwiki.pixar.com/display/REN24/NURBS>



Ed Catmull's "Gumbo" model, composed from Bezier patches
https://en.wikipedia.org/wiki/B%C3%A9zier_surface

B-Spline surfaces

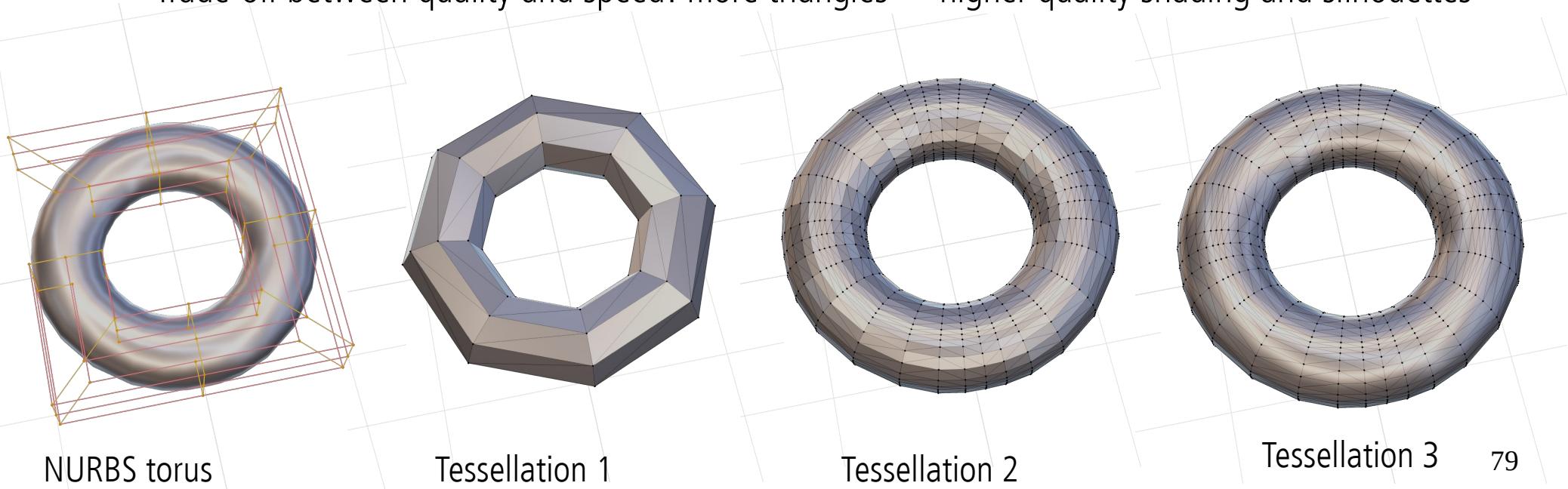
- B-Spline curves can be extended to **B-Spline surfaces** which are similar to Bezier surface
- Often **bicubic B-Spline surface is used to form composite surface**
 - Basis for Catmull-Clark subdivision surfaces
- **Non-uniform rational B-Spline surface (NURBS)** is often used in 3D modeling software



77

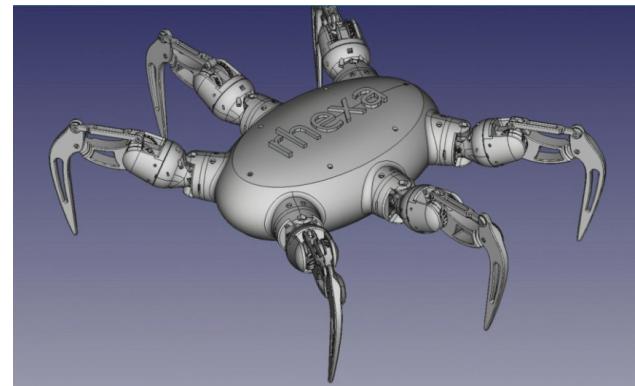
Rendering of parametric surfaces

- Ray-parametric-surface intersection can be defined
- Model made with parametric surfaces is **tessellated** for efficient rendering process.
 - Trade-off between quality and speed: more triangles → higher quality shading and silhouettes



Modeling and transferring parametric surfaces

- Parametric curves and surfaces are extensively used in CAD and modeling software:
 - <https://www.freecad.org/>
 - <https://docs.blender.org/manual/en/latest/modeling/surfaces/index.html>
 - <https://www.autodesk.com/collections/product-design-manufacturing/overview>
- Parametric surfaces are stored using:
 - **STEP file format:** <https://www.adobe.com/creativecloud/file-types/image/vector/step-file.html>
 - **IGES file format:** <https://www.adobe.com/creativecloud/file-types/image/vector/iges-file.html>



Exploring parametric curves and surfaces

- Parametric curves and surfaces are highly used and researched method in computer graphics. We only covered foundations.
- NURBS in real-time: <https://www.gamedeveloper.com/programming/using-nurbs-surfaces-in-real-time-applications>
- 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>

Implicit surfaces

Implicit surfaces

- Described with a function. Instead of using parameters (u, v) surface is described with implicit function $f(p)$:

$f(p) = 0$, if p on surface

$f(p) < 0$, if p inside surface

$f(p) > 0$, if p outside surface

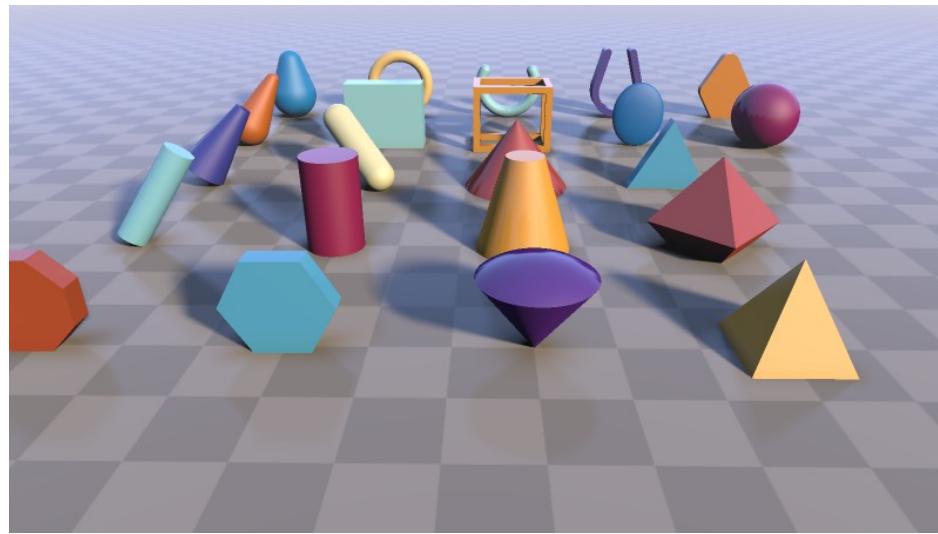
- Surface can not be computed directly
 - Distance is computed by evaluating point $p(x, y, z)$ in space → **signed distance function**

```
float sdSphere( vec3 p, float s )
{
    return length(p)-s;
}
```

```
float sdTorus( vec3 p, vec2 t )
{
    vec2 q = vec2(length(p.xz)-t.x,p.y);
    return length(q)-t.y;
}
```

```
float sdRoundBox( vec3 p, vec3 b, float r )
{
    vec3 q = abs(p) - b;
    return length(max(q,0.0)) + min(max(q.x,max(q.y,q.z)),0.0) - r;
}
```

```
float sdRoundedCylinder( vec3 p, float ra, float rb, float h )
{
    vec2 d = vec2( length(p.xz)-2.0*ra+rb, abs(p.y) - h );
    return min(max(d.x,d.y),0.0) + length(max(d,0.0)) - rb;
}
```



Implicit surfaces: normal

- Crucial information for shading is normal.
- Normal vector \mathbf{N} of implicit surface in point p
 - Normalized gradient of function f in point p : $\nabla f(p)$
 - Intuition: Gradient points in direction of steepest ascent in the function near p and thus is orthogonal to surface



Analytical expression:

- Function must be differentiable

$$\nabla f(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$$

Approximation:

- Central difference using small number ϵ

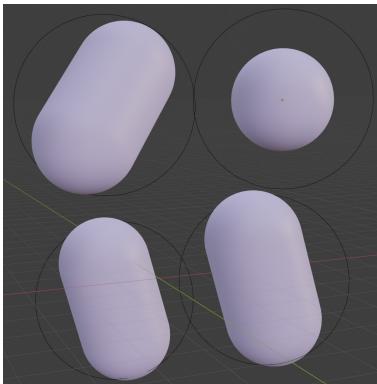
$$\nabla f_x = f(p + \epsilon e_x) - f(p - \epsilon e_x), e_x = (1, 0, 0)$$

$$\nabla f_y = f(p + \epsilon e_y) - f(p - \epsilon e_y), e_y = (0, 1, 0)$$

$$\nabla f_z = f(p + \epsilon e_z) - f(p - \epsilon e_z), e_z = (0, 0, 1)$$

Implicit surfaces: modeling

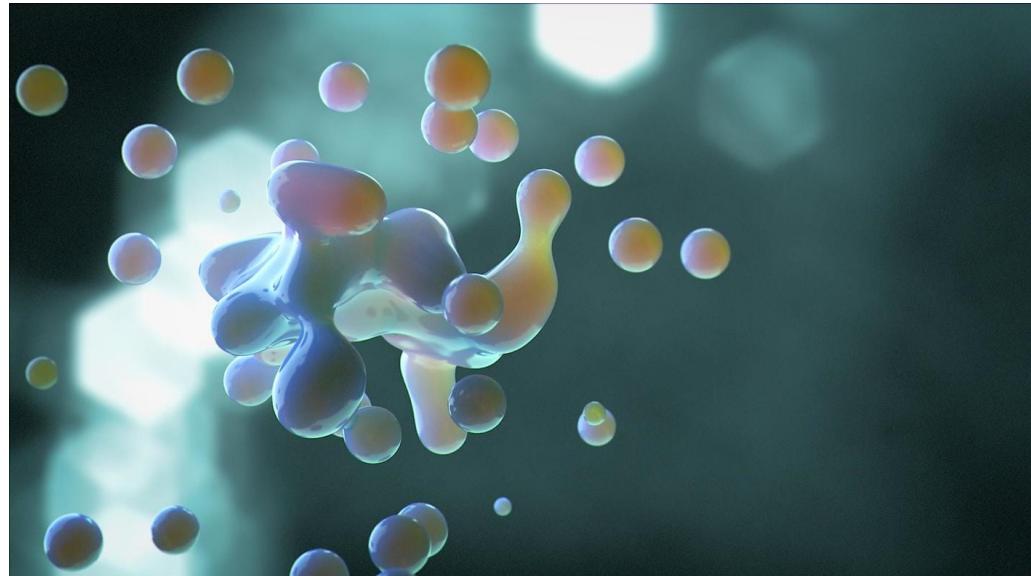
- Blending of implicit surfaces: blobby modeling, soft objects or metaballs
 - Blender: <https://docs.blender.org/manual/en/latest/modeling/metaballs/index.html>



- Example of blending: resulting distance d by blending between two distances d_1 and d_2 with a blend radius r_b

$$d = (1 - h)d_2 + hd_1 + r_b h(1 - h)$$

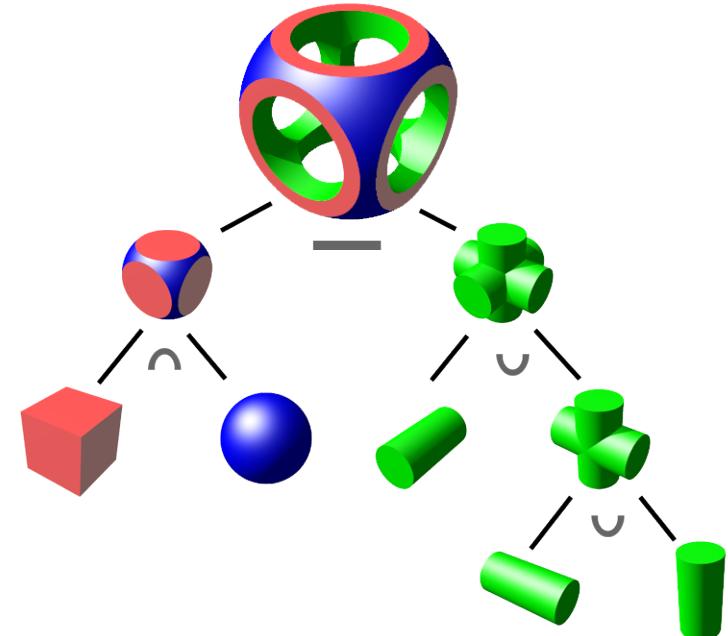
$$h = \min(\max(0.5 + 0.5(d_2 - d_1)/r_b, 0.0), 1.0)$$



<https://www.chaos.com/gallery/elmar-glaubauf-metaballs>

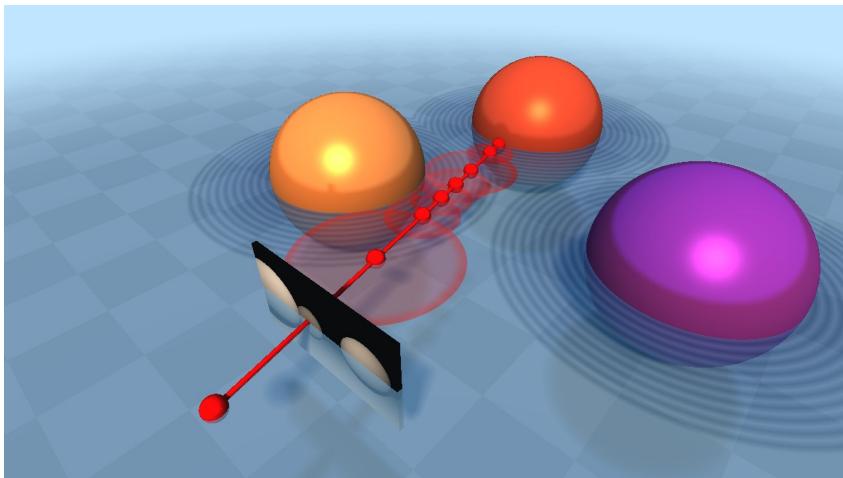
Implicit surfaces: modeling

- **Constructive solid geometry:** set of operations –
Boolean operations:
 - Intersection
 - Union
 - Difference
- Well defined for closed objects (volumetric bodies)
- Used to create complex shapes from simple, base shapes (e.g., cubes, spheres, etc.)



Implicit surfaces: rendering

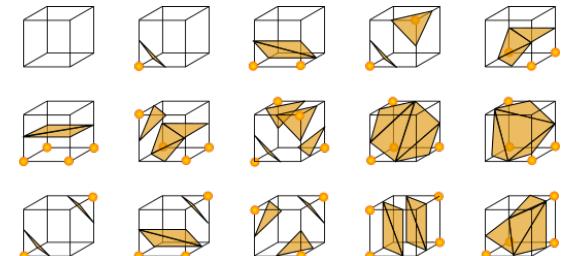
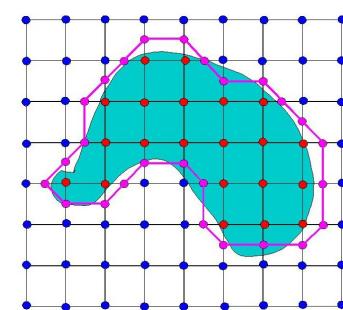
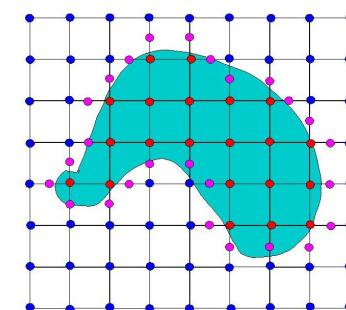
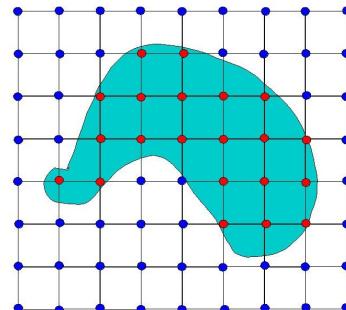
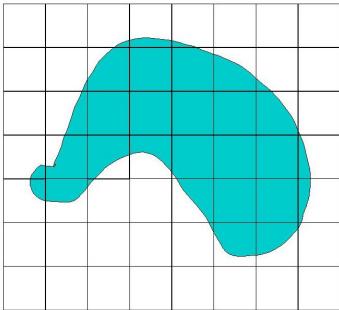
- Ray-marching
 - At first point p on the ray, shortest distance d to the surface is evaluated (sphere around the point)
 - Move in ray direction for distance d
 - Repeat until surface is reached within some distance (small threshold) or max number of steps is reached (background hit)
- Effects: shadows, reflections, ambient occlusion, etc.



<https://iquilezles.org/articles/raymarchingdf/>

Implicit surfaces: rendering

- Turn surface into triangles (triangulation) and then render
- **Marching cubes algorithm:** surface extraction, polygonization
 - First, surface is partitioned into adjacent cells at whose corners the implicit surface is evaluated
 - Negative values are considered outside surface, positive values inside surface
 - Within each cell, the intersection of cell edges with the implicit surface are connected to form one or more polygons



3D case: creating cells, labeling cell corners, calculating cell edge points, connecting edge points to form polygon

3D case

Summary questions

- https://github.com/lorentzo/IntroductionToComputerGraphics/tree/main/lectures/6_parametric_curves_surfaces

Reading material

- <https://github.com/lorentzo/IntroductionToComputerGraphics>