Computer Graphics and Interaction

DH2323 / Spring 2015 / P4

Bezier Curves, Splines and Surfaces

de Casteljau Algorithm · Bernstein Form

Bezier Splines

Tensor Product Surfaces · Total Degree Surfaces

Prof. Dr. Tino Weinkauf

Lab assignment

Lab help session

this Friday, May 8^{th}

in the VIC from 15:00-17:00

Bezier Curves
de Casteljau algorithm

- Paul de Casteljau (1959) @ Citroën
- Pierre Bezier (1963) @ Renault

Meine Zeit bei Citroën / My time at Citroën
see the PDF deCasteljau_de.pdf and deCasteljau_en.pdf in the download area of the webpage

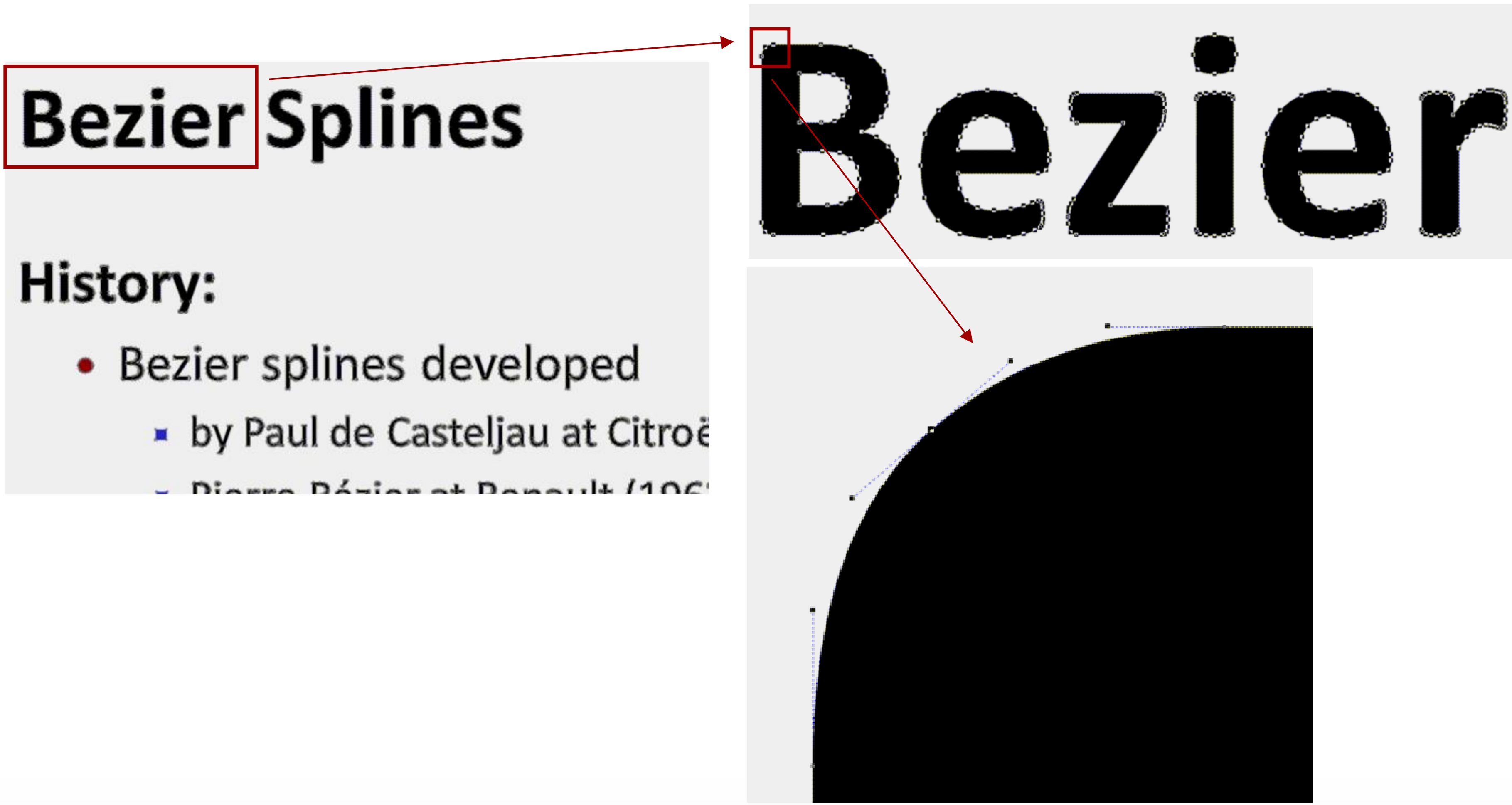
Bezier curves

History:

- Bezier curves/splines developed by
  - Paul de Casteljau at Citroën (1959)
  - Pierre Bézier at Renault (1963)

  for free-form parts in automotive design
- Today: Standard tool for 2D curve editing
- Cubic 2D Bezier curves are everywhere:
  - Postscript, PDF, Truetype (quadratic curves), Windows GDI...
  - Inkscape, Corel Draw, Adobe Illustrator, Powerpoint, ...
- Widely used in 3D curve & surface modeling as well

All You See is Bezier Curves...

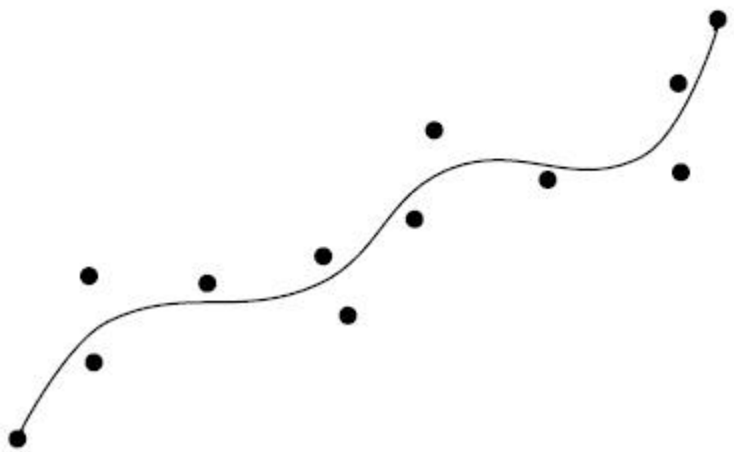


De Casteljau algorithm

Approximation setting:

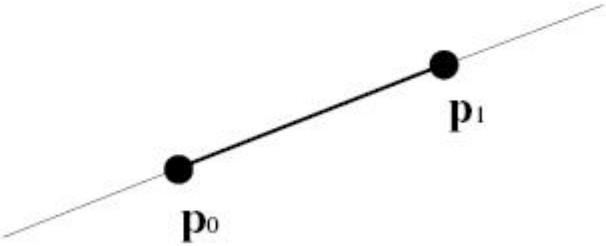
Given: $p_0, \ldots, p_n$

Wanted: smooth, approximating curve



De Casteljau algorithm

Linear interpolation



$$\mathbf{x}(t) = (1 - t) \cdot \mathbf{p}_0 + t \cdot \mathbf{p}_1$$

De Casteljau algorithm

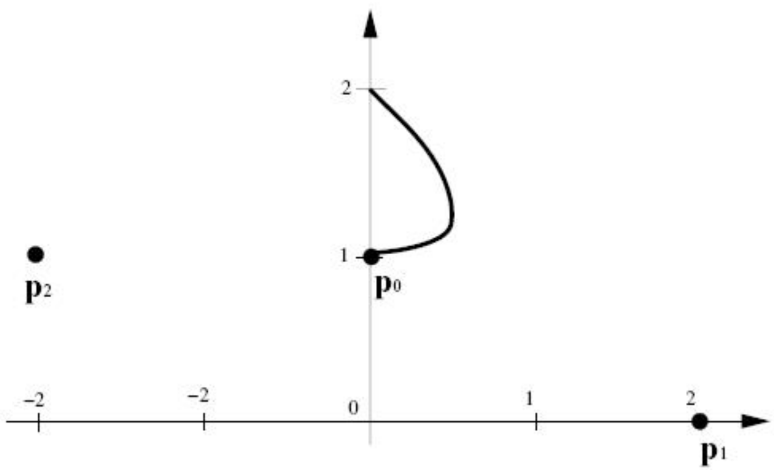
Parabolas

$$\mathbf{x}(t) = \mathbf{p}_0 + t \cdot \mathbf{p}_1 + t^2 \cdot \mathbf{p}_2$$

➔ planar curve, even if defined in R³

Example:

$$\mathbf{p}_0 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{p}_1 = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{p}_2 = \begin{pmatrix} -2 \\ 1 \\ 0 \end{pmatrix}$$



De Casteljau algorithm

Another parabola construction

given: 3 points b_0, b_1, b_2

$$\mathbf{b}_0^1 = (1-t)\cdot \mathbf{b}_0 + t\cdot \mathbf{b}_1$$
$$\mathbf{b}_1^1 = (1-t)\cdot \mathbf{b}_1 + t\cdot \mathbf{b}_2$$

$$\mathbf{b}_0^2 = (1-t)\cdot \mathbf{b}_0^1 + t\cdot \mathbf{b}_1^1$$

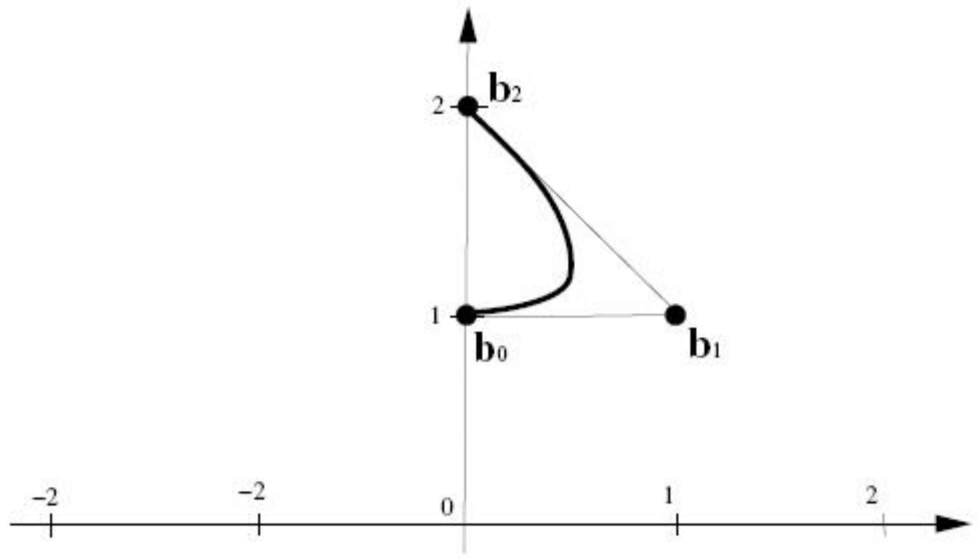
→ parabola x(t)

$$\mathbf{x}(t) = (1-t)^2\cdot \mathbf{b}_0 + 2\cdot t\cdot (1-t)\cdot \mathbf{b}_1 + t^2\cdot \mathbf{b}_2$$

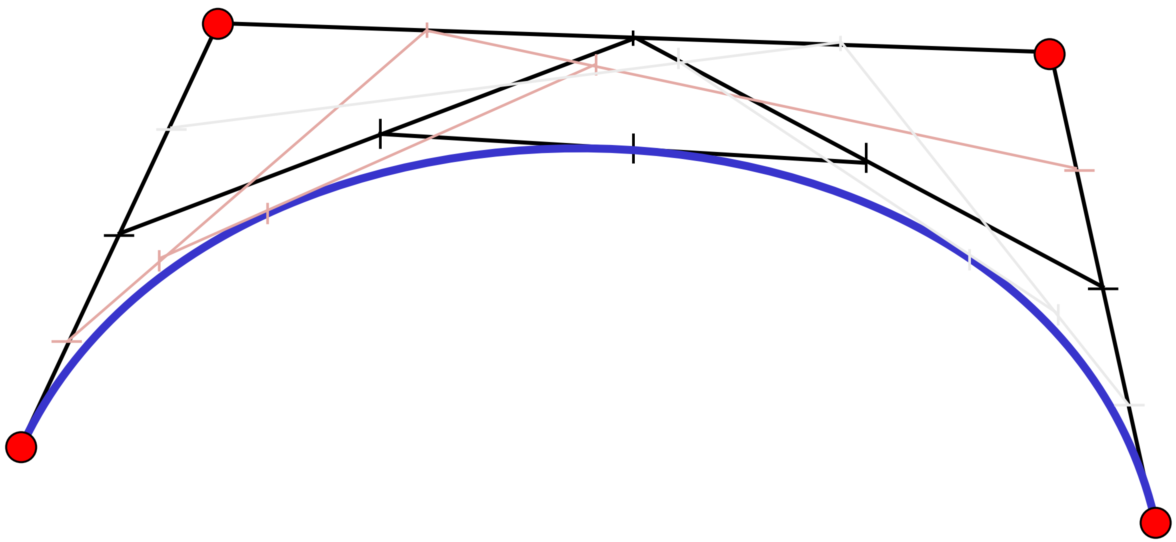
De Casteljau algorithm

Example

$$\mathbf{b}_0 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \;,\; \mathbf{b}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \;,\; \mathbf{b}_2 = \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix}$$



De Casteljau algorithm



De Casteljau Algorithm: Computes $x(t)$ for given t

- Bisect control polygon in ratio $t : (1 - t)$
- Connect the new dots with lines (adjacent segments)
- Interpolate again with the same ratio
- Iterate, until only one point is left

De Casteljau algorithm

Description of the de Casteljau algorithm

- given: points $\mathbf{b}_0, \mathbf{b}_1, ..., \mathbf{b}_n \in \mathbb{R}^3$
- wanted: curve $\mathbf{x}(t), \quad t \in [0,1]$

- geometric construction of the point x(t) for given t:

$$\mathbf{b}_i^0(t) = \mathbf{b}_i \qquad \text{für } i = 0, ..., n$$

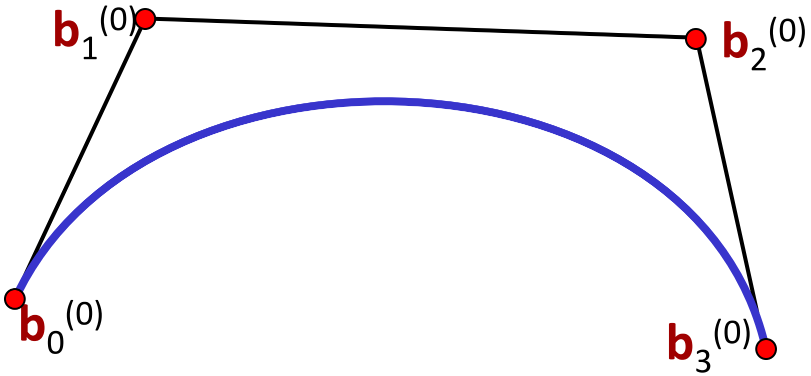
$$\mathbf{b}_i^r(t) = (1-t) \cdot \mathbf{b}_i^{r-1}(t) \; + \; t \cdot \mathbf{b}_{i+1}^{r-1}(t)$$
$$\text{für } r = 1, ..., n \quad ; \quad i = 0, ..., n-r.$$

- Then, $\mathbf{b}_0^n(t)$ is the searched curve point x(t) at the parameter value t

De Casteljau algorithm

repeated convex combination of control points

$$\mathbf{b}_i^{(r)} = (1-t)\cdot\mathbf{b}_i^{(r-1)} + t\cdot\mathbf{b}_{i+1}^{(r-1)}$$



$\mathbf{b}_0^{(0)}$

$\mathbf{b}_1^{(0)}$

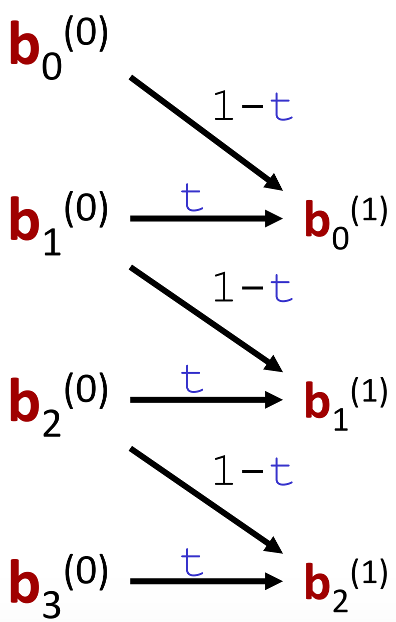
$\mathbf{b}_2^{(0)}$

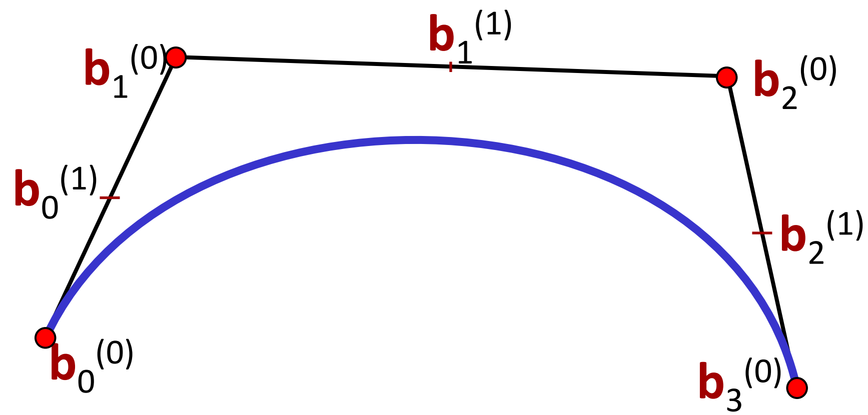
$\mathbf{b}_3^{(0)}$

De Casteljau algorithm

repeated convex combination of control points

$$\mathbf{b}_i^{(r)} = (1-t)\cdot\mathbf{b}_i^{(r-1)} + t\cdot\mathbf{b}_{i+1}^{(r-1)}$$

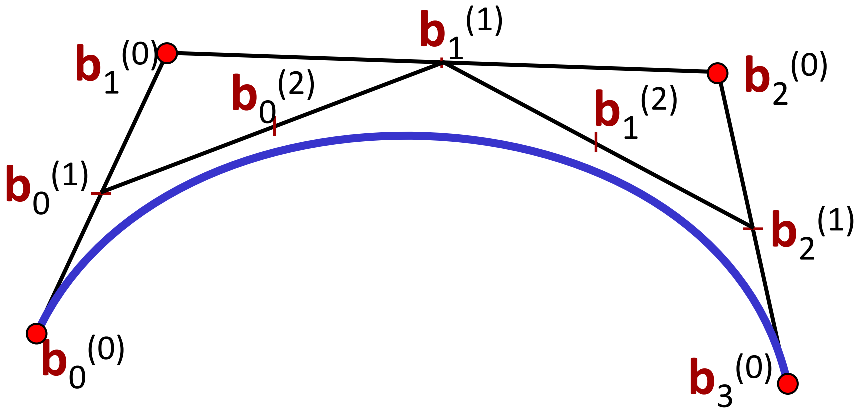


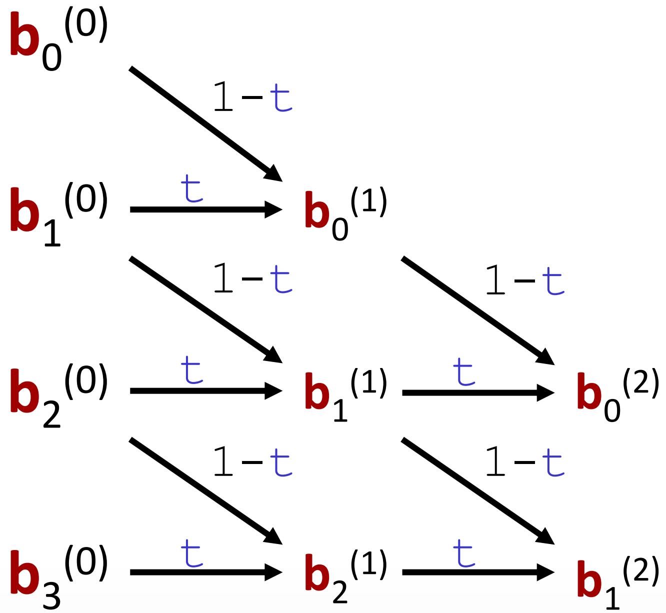


De Casteljau algorithm

repeated convex combination of control points

$$\mathbf{b}_i^{(r)} = (1-t)\cdot\mathbf{b}_i^{(r-1)} + t\cdot\mathbf{b}_{i+1}^{(r-1)}$$

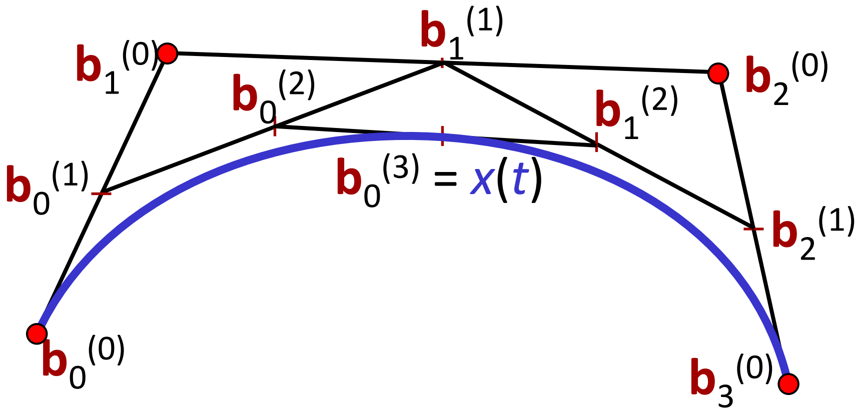


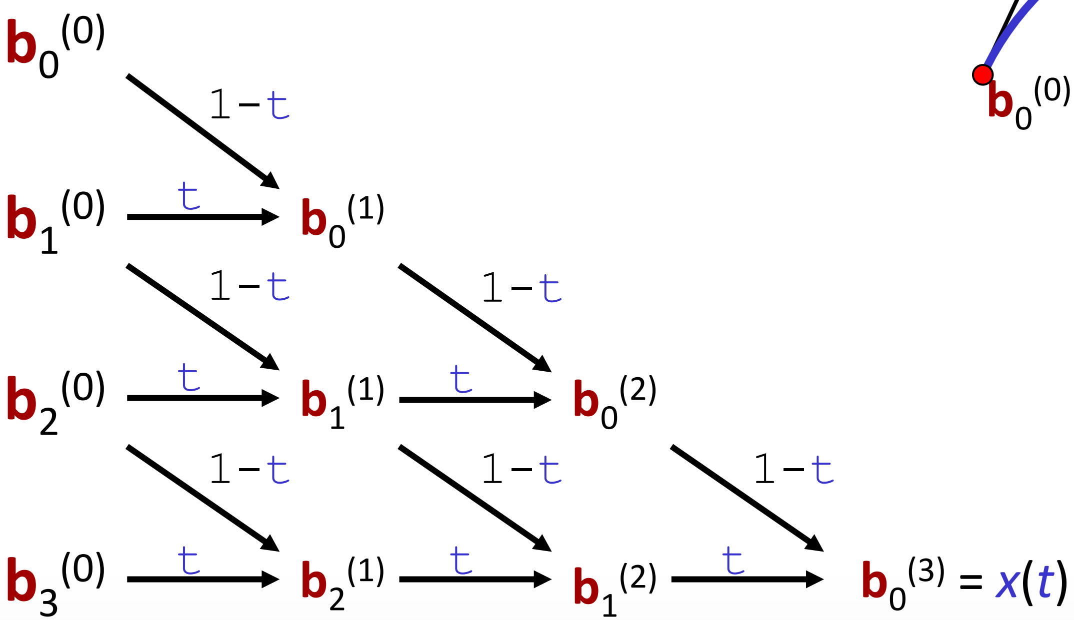


De Casteljau algorithm

repeated convex combination of control points

$$\mathbf{b}_i^{(r)} = (1-t)\cdot\mathbf{b}_i^{(r-1)} + t\cdot\mathbf{b}_{i+1}^{(r-1)}$$





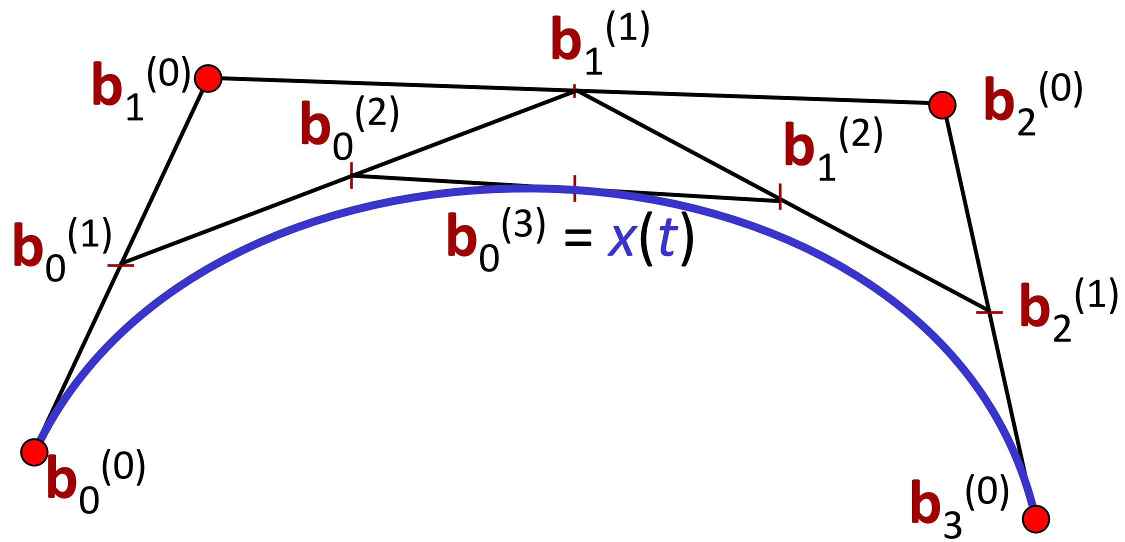
de Casteljau scheme

De Casteljau algorithm

The intermediate coefficients $b_i^r(t)$ can be written in a triangular matrix: the de Casteljau scheme:

$$
\begin{array}{lllll}
\mathbf{b}_0 = \mathbf{b}_0^0 & & & & \\
\mathbf{b}_1 = \mathbf{b}_1^0 & \mathbf{b}_0^1 & & & \\
\mathbf{b}_2 = \mathbf{b}_2^0 & \mathbf{b}_1^1 & \mathbf{b}_0^2 & & \\
\mathbf{b}_3 = \mathbf{b}_3^0 & \mathbf{b}_2^1 & \mathbf{b}_1^2 & \mathbf{b}_0^3 & \\
\quad\vdots & & & & \\
\mathbf{b}_{n-1} = \mathbf{b}_{n-1}^0 & \mathbf{b}_{n-2}^1 & \ldots & \mathbf{b}_0^{n-1} & \\
\mathbf{b}_n \quad = \mathbf{b}_n^0 & \mathbf{b}_{n-1}^1 & \ldots & \mathbf{b}_1^{n-1} & \mathbf{b}_0^n = \mathbf{x}(t)
\end{array}
$$

De Casteljau algorithm



Algorithm:

```
for r = 1..n do
    for i = 0..n-r do
        b_i^(r) = (1-t)·b_i^(r-1) + t·b_{i+1}^(r-1)
    end for
end for
return b_0^(n)
```

$$\mathbf{b}_i^{(r)} = (1-t)\cdot\mathbf{b}_i^{(r-1)} + t\cdot\mathbf{b}_{i+1}^{(r-1)}$$

The whole algorithm consists only of repeated linear interpolations.

De Casteljau algorithm

The polygon consisting of the points b_0, …, b_n is called Bezier polygon. The points b_i are called Bezier points.

The curve defined by the Bezier points b_0, …, b_n and the de Casteljau algorithm is called Bezier curve.

The de Casteljau algorithm is numerically stable, since only convex combinations are applied.

Complexity of the de Casteljau algorithm

- $O(n^2)$ time
- $O(n)$ memory
- with n being the number of Bezier points

De Casteljau algorithm

Properties of Bezier curves:

- given: Bezier points $\mathbf{b}_0, \ldots, \mathbf{b}_n$

  Bezier curve $\mathbf{x}(t)$
- Bezier curve is polynomial curve of degree n.
- End point interpolation: $\mathbf{x}(0) = \mathbf{b}_0$, $\mathbf{x}(1) = \mathbf{b}_n$. The remaining Bezier points are only generally approximated.
- Convex hull property:

  Bezier curve is completely inside the convex hull of its Bezier polygon.

De Casteljau algorithm

- **Variation diminishing**
  no line intersects the Bezier curve more often than its Bezier polygon.

- **Influence of Bezier points: global, but pseudo-local**
  - *global:* moving a Bezier point changes the whole curve progression
  - *pseudo-local:* $\mathbf{b}_i$ has its maximal influence on $\mathbf{x}(t)$ at $t = i/n$.

- **Affine invariance:**
  Bezier curve and Bezier polygon are invariant under affine transformations

- **Invariance under affine parameter transformations**

De Casteljau algorithm

- **Symmetry:**
  The following two Bezier curves coincide, they are only traversed in opposite directions:

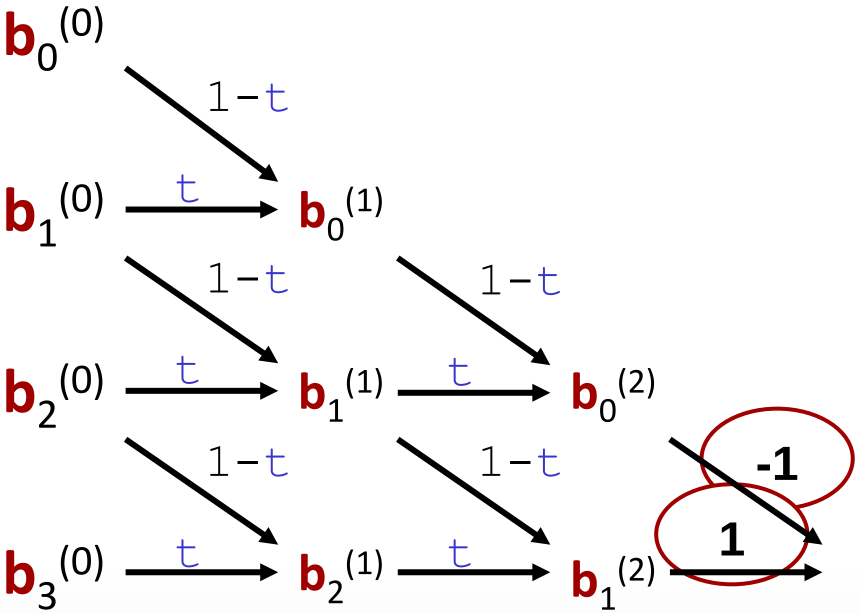
$$\mathbf{x}(t) = [\mathbf{b}_0, \ldots, \mathbf{b}_n] \qquad \mathbf{x}'(t) = [\mathbf{b}_n, \ldots, \mathbf{b}_0]$$

- **Linear precision:**
  Bezier curve is line segment, if $\mathbf{b}_0, \ldots, \mathbf{b}_n$ are collinear

- **Invariant under barycentric combinations**

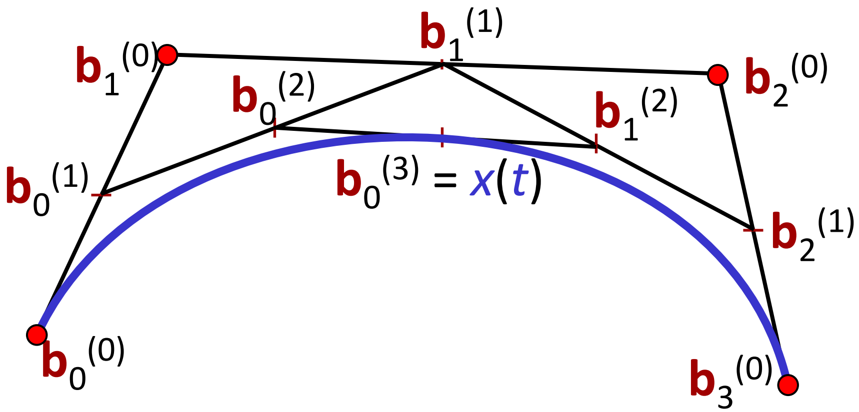
De Casteljau algorithm

- **First derivative of a Bezier curve**
  - **Endpoints:** $\dot{\mathbf{x}}(0) = n \cdot (\mathbf{b}_1 - \mathbf{b}_0)$ $\dot{\mathbf{x}}(1) = n \cdot (\mathbf{b}_n - \mathbf{b}_{n-1})$ $t = 0, t = 1:$



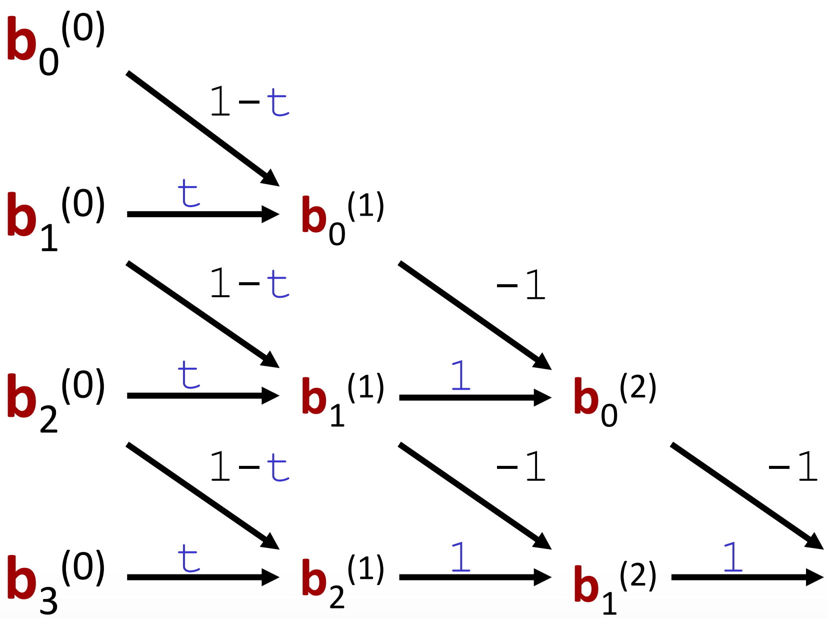
$$\dot{\mathbf{x}}(t) = n\left(\mathbf{b}_1^{(n-1)} - \mathbf{b}_0^{(n-1)}\right)$$

de Casteljau scheme

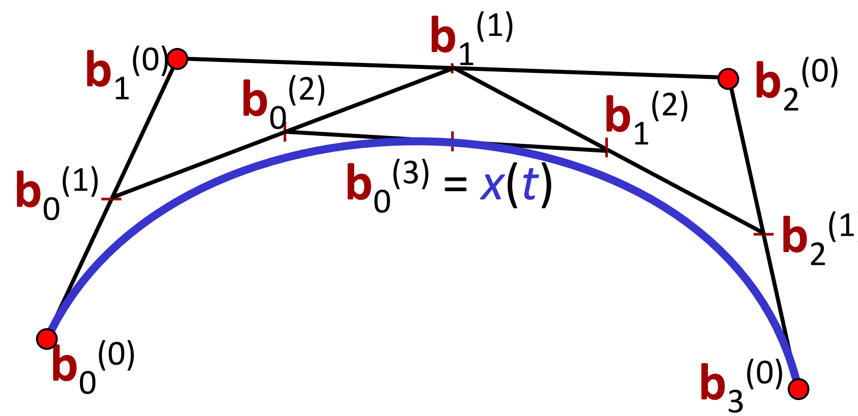


De Casteljau algorithm

- **Second derivative of a Bezier curve**



de Casteljau scheme



$$\ddot{\mathbf{x}}(t) = n(n-1)\left(\mathbf{b}_2^{(n-2)} - 2\mathbf{b}_1^{(n-2)} + \mathbf{b}_0^{(n-2)}\right)$$

Bezier Curves

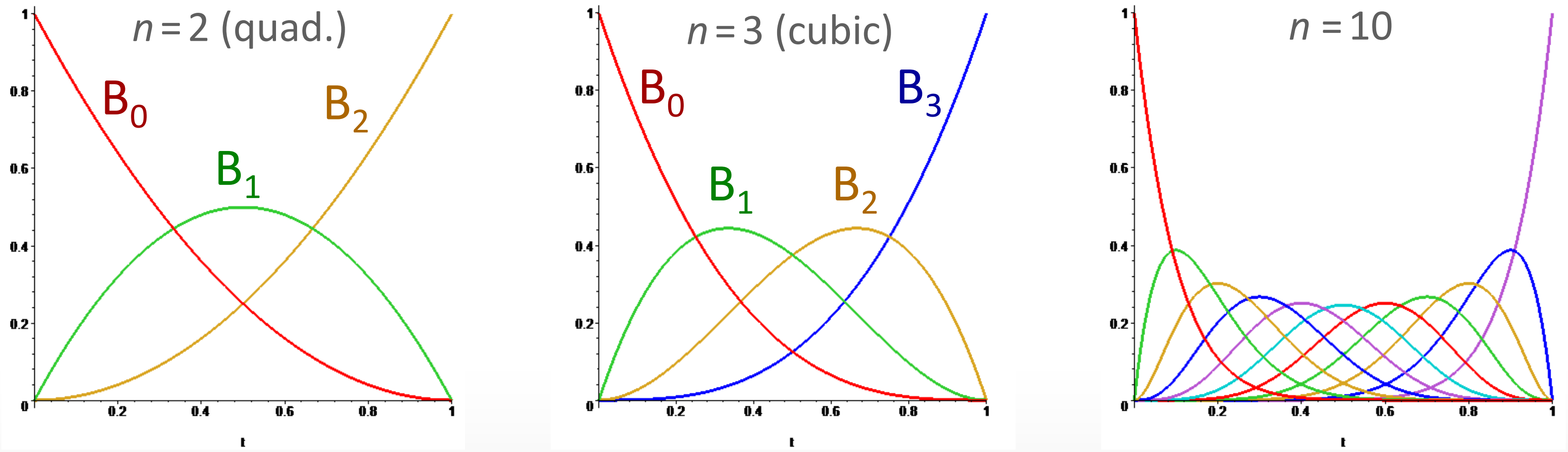
Bernstein form

Bernstein Basis

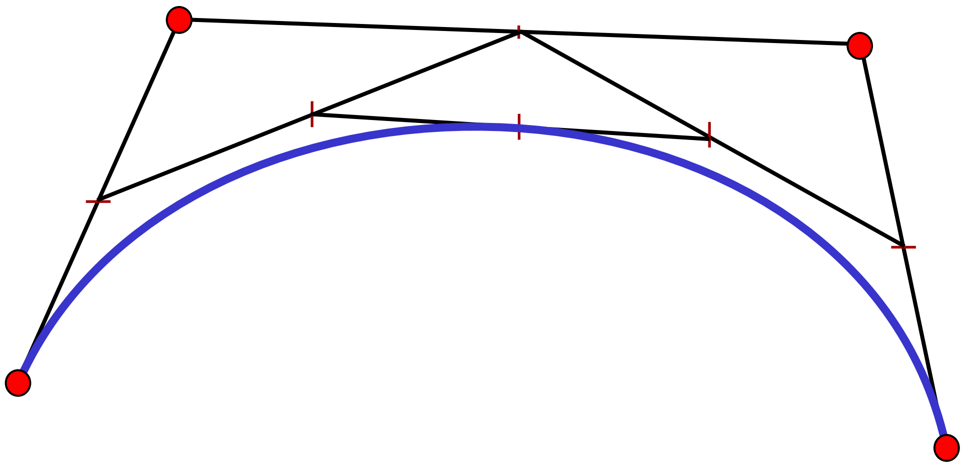
Bezier curves are algebraically defined using the Bernstein basis:

- Bernstein basis of degree *n*: $B = \left\{ B_0^{(n)}, B_1^{(n)}, ..., B_n^{(n)} \right\}$

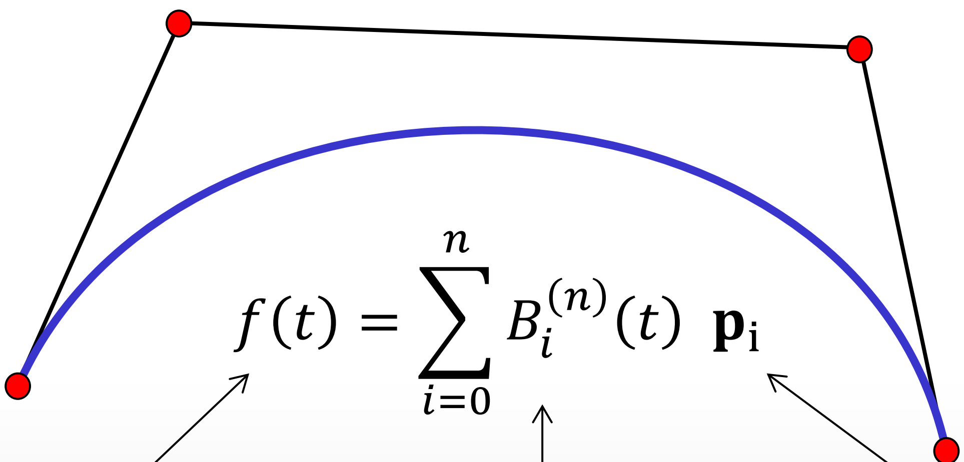
$$B_i^{(n)}(t) := \binom{n}{i} t^i (1-t)^{n-i}$$



Bernstein Basis



de Casteljau algorithm



Bernstein form

$$f(t) = \sum_{i=0}^{n} B_i^{(n)}(t)\ \mathbf{p}_\mathrm{i}$$

curve — basis function — control point

Examples

The first three Bernstein bases:

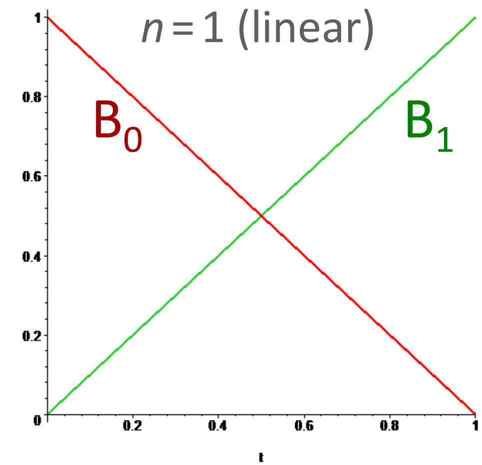
$$B_0^{(0)} := 1$$

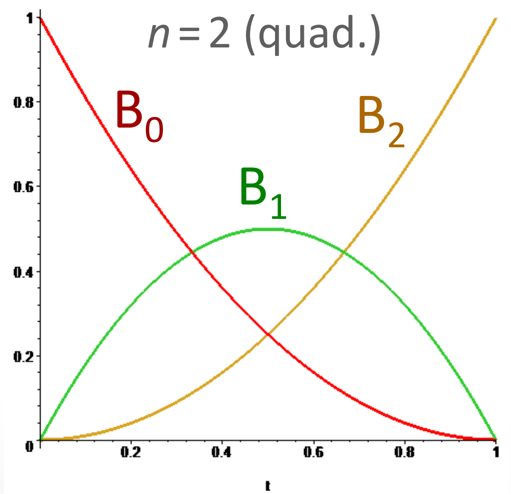
$$B_0^{(1)} := (1-t) \qquad B_1^{(1)} := t$$

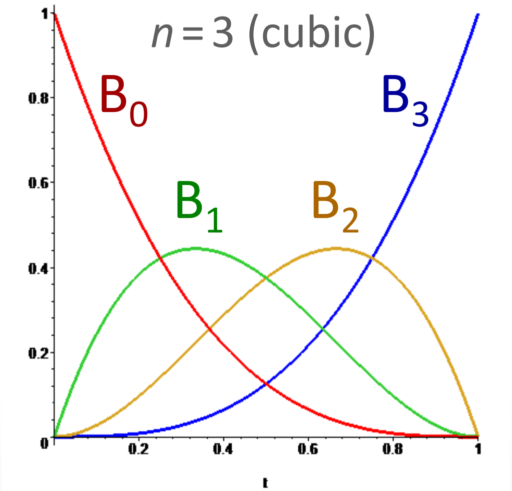
$$B_0^{(2)} := (1-t)^2 \qquad B_1^{(2)} := 2t(1-t) \qquad B_2^{(2)} := t^2$$

$$B_0^{(3)} := (1-t)^3 \qquad B_1^{(3)} := 3t(1-t)^2$$

$$B_2^{(3)} := 3t^2(1-t) \qquad B_3^{(3)} := t^3$$







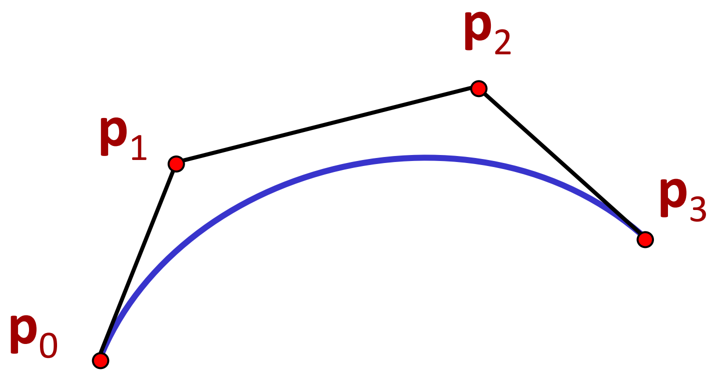
$$B_i^{(n)}(t) := \binom{n}{i} t^i (1-t)^{n-i}$$

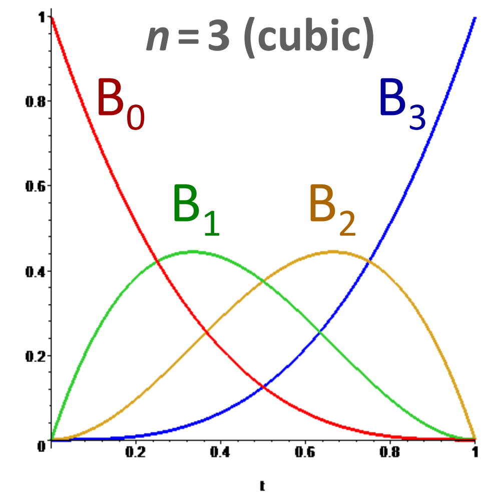
Bezier Curves in Bernstein form

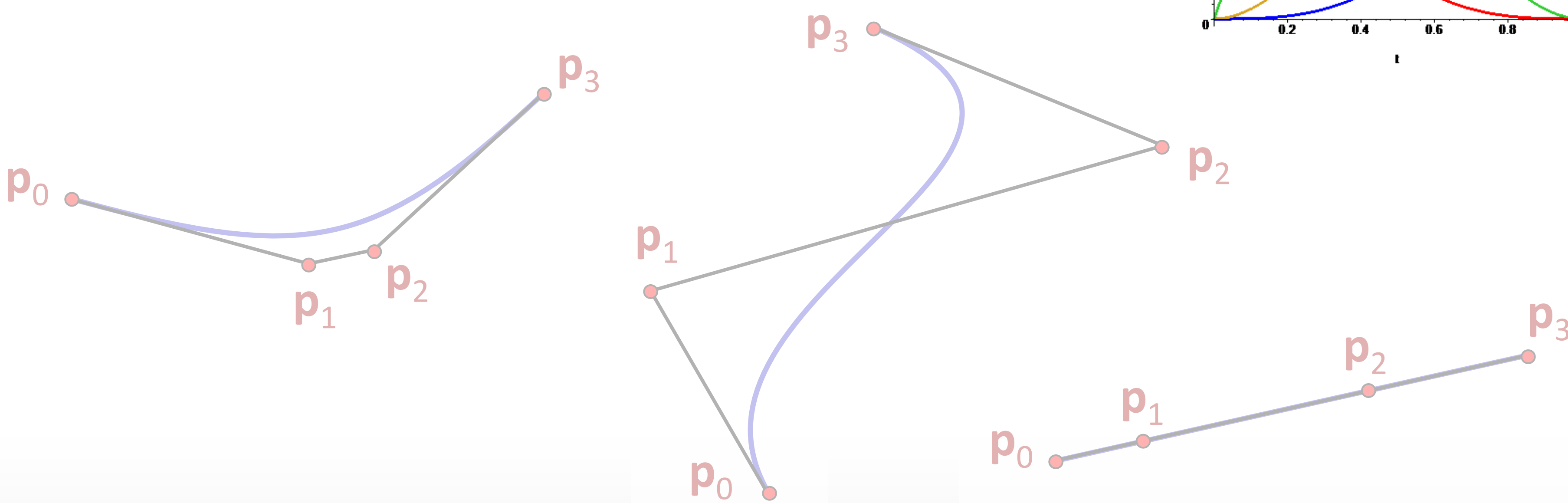
Bezier Curves:

- $\mathbf{f}(t) = \sum_{i=0}^{n} \mathbf{p}_i B_i^{(n)}$

  $t \in [0..1]$



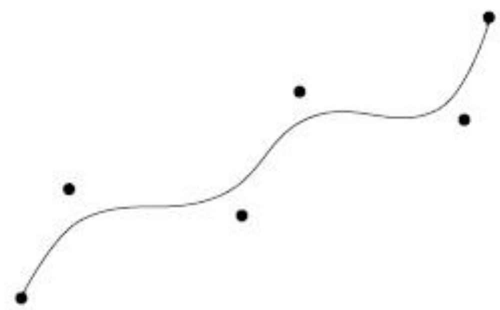




Summary for Bezier Curves

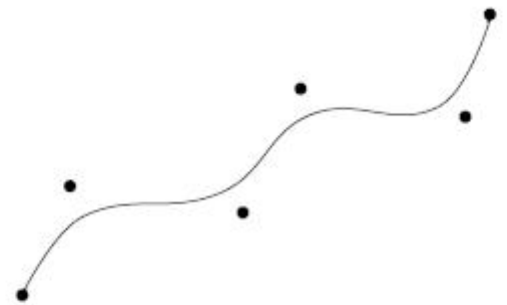
Bezier curves and curve design:

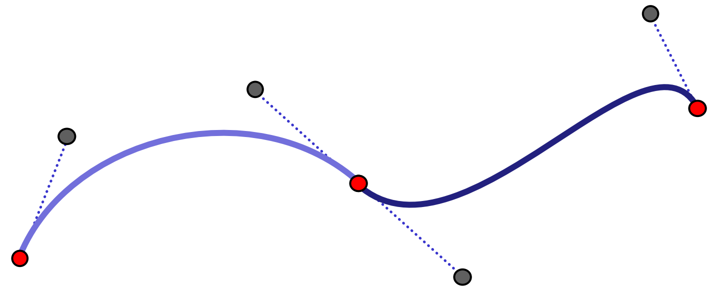
- The rough form is specified by the position of the control points
- Result: smooth curve approximating the control points
- Computation / Representation:
  - de Casteljau algorithm
  - Bernstein form



- Problems:
  - high polynomial degree
  - moving a control point can change the whole curve
  - interpolation of points
  - → **Bezier splines**

Towards Bezier Splines





Approximation → Interpolation

Towards Bezier Splines

Interpolation problem:

- given:

  $\mathbf{k}_0, ..., \mathbf{k}_n \in \mathbb{R}^3$ control points

  $t_0, ..., t_n \in \mathbb{R}$ knot sequence

  $t_i < t_{i+1}$ für $i = 0, ..., n-1$

- wanted:

  interpolating curve $\mathbf{x}(t)$, i.e., $\mathbf{x}(t_i) = \mathbf{k}_i$ for $i = 0, \ldots, n$

- Approach:
  "Joining" of n Bezier curves with certain intersection conditions

Towards Bezier Splines

The following issues arise when stitching together Bezier curves:

- Continuity
- Degree
- (Parameterization)

Bezier Splines

Parametric and Geometric Continuity

Continuity

Joining of curves - continuity

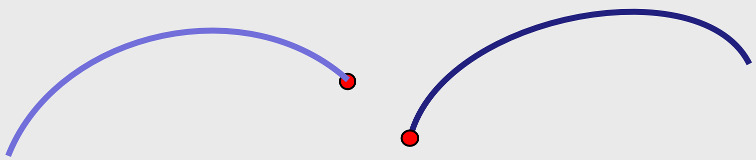
- given: 2 curves

  $\mathbf{x}_1(t)$ over $[t_0, t_1]$

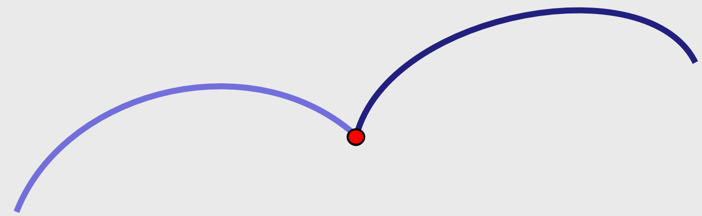
  $\mathbf{x}_2(t)$ over $[t_1, t_2]$

- $\mathbf{x}_1$ and $\mathbf{x}_2$ are C^r continuous in t_1, if they coincide in $0^{th} - r^{th}$ derivative vector in t_1.

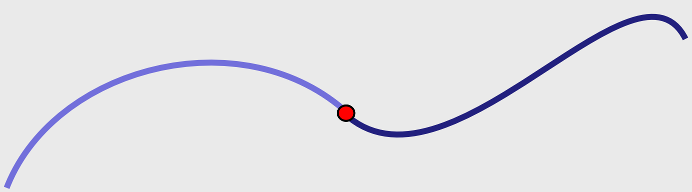
Continuity



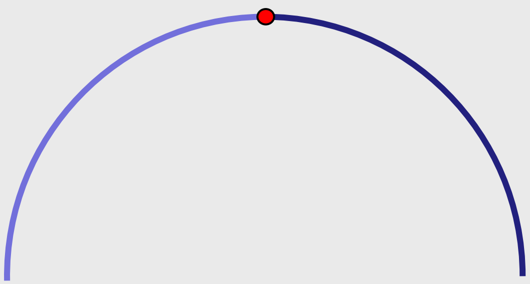
C^{-1} continuity



C^0 continuity



C^1 continuity



C^2 continuity

Continuity

Parametric Continuity C^r**:**

- C^0, C^1, C^2... continuity.
- Does a particle moving on this curve have a smooth trajectory (position, velocity, acceleration,...)?
- Useful for animation (object movement, camera paths)
- **Depends** on parameterization

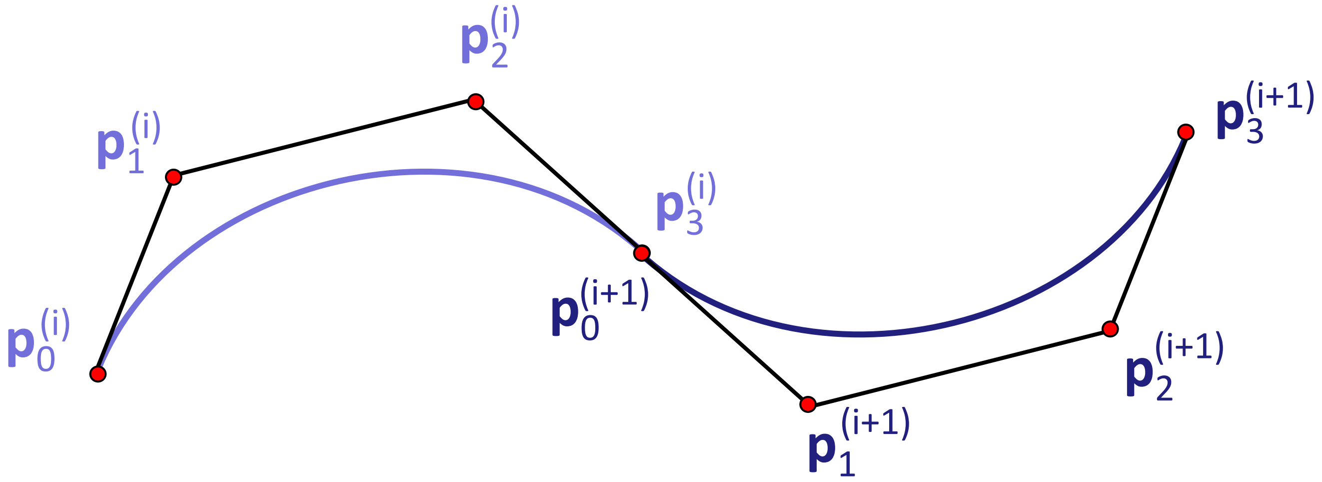
Geometric Continuity G^r**:**

- **Independent** of parameterization
- Is the curve itself smooth?
- More relevant for modeling (curve design)

Bezier Splines

Local control: Bezier splines

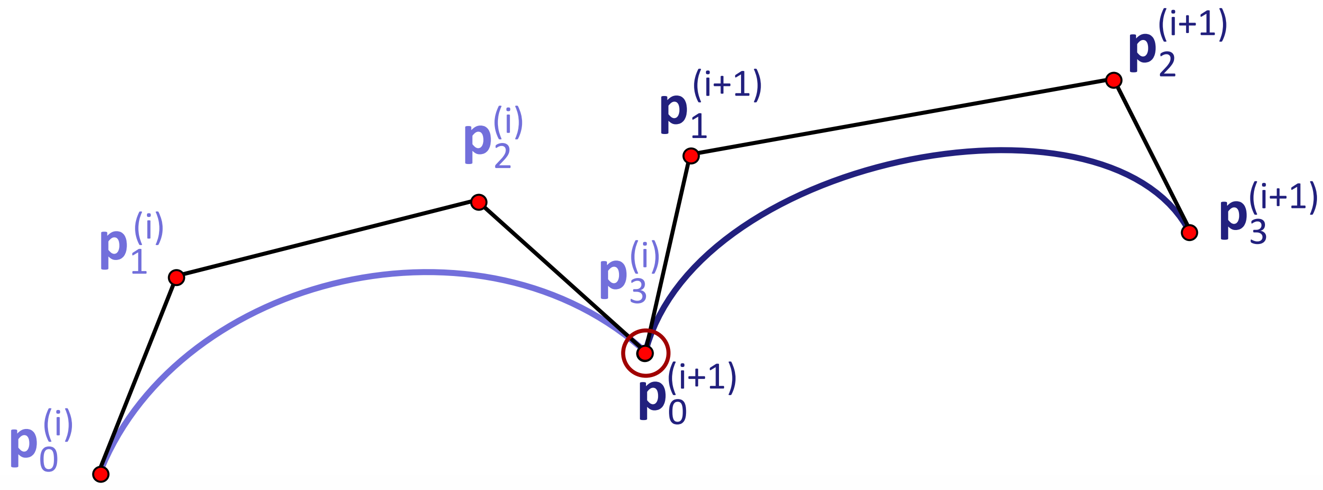
- Concatenate several curve segments
- Question: Which constraints to place upon the control points in order to get C^{-1}, C^0, C^1, C^2 continuity?



Bezier Spline Continuity

Rules for Bezier spline continuity:

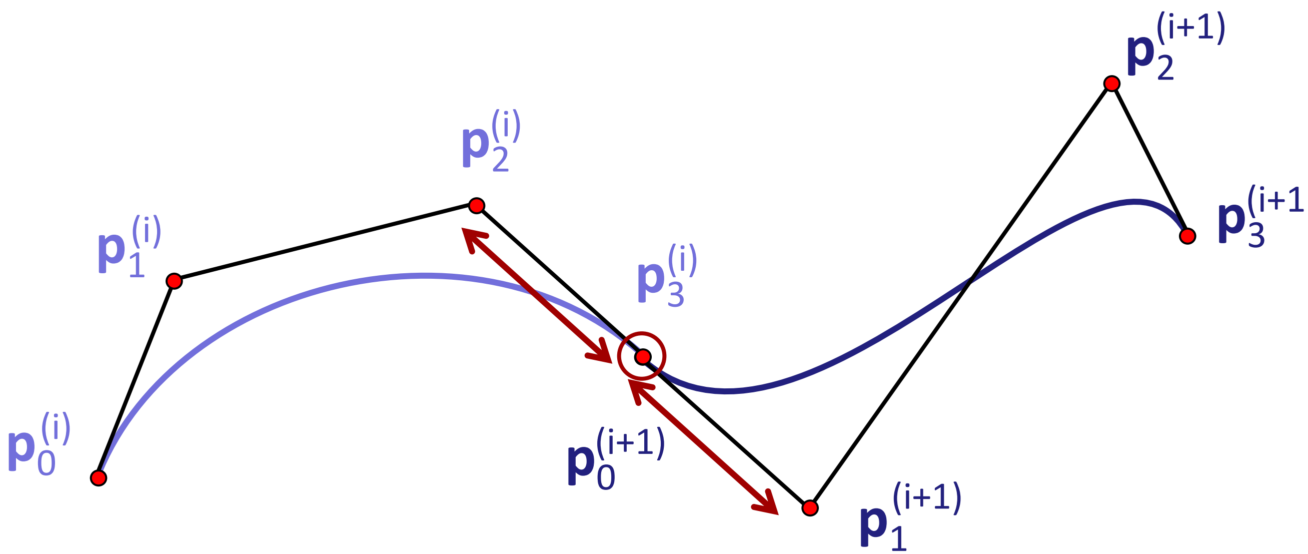
- C^0 continuity:
  - Each spline segment interpolates the first and last control point
  - Therefore: Points of neighboring segments have to coincide for C^0 continuity.



Bezier Spline Continuity

Rules for Bezier spline continuity:

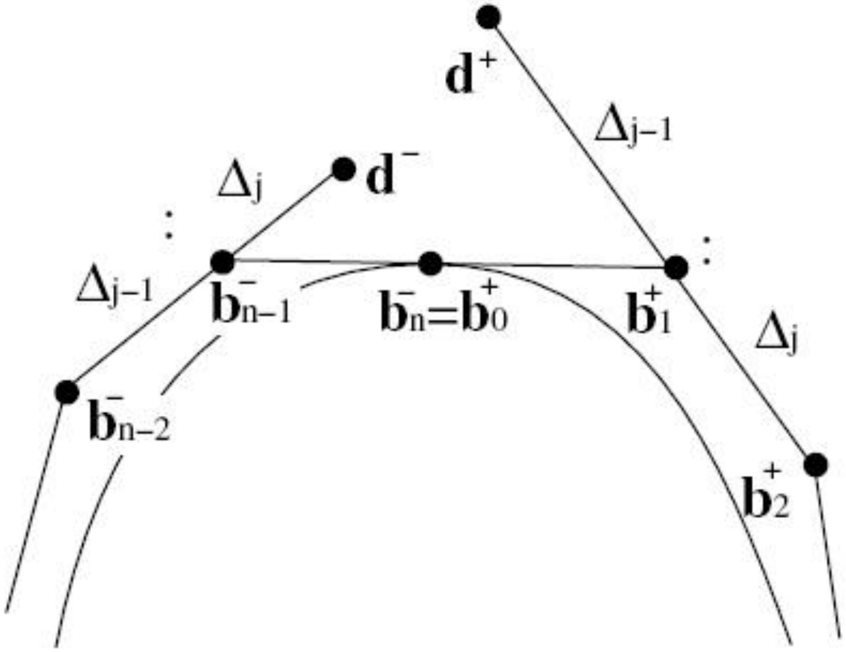
- Additional requirement for C^1 continuity:
  - Tangent vectors are proportional to differences $\mathbf{p}_1 - \mathbf{p}_0$, $\mathbf{p}_n - \mathbf{p}_{n-1}$
  - Therefore: These vectors must be identical for C^1 continuity



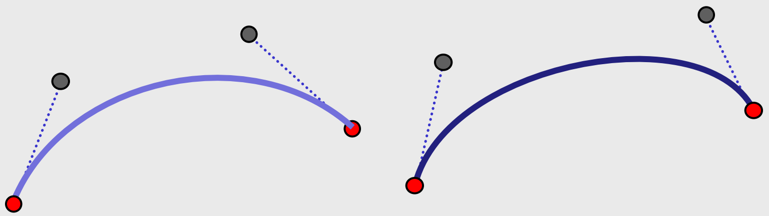
Bezier Spline Continuity

Rules for Bezier spline continuity:

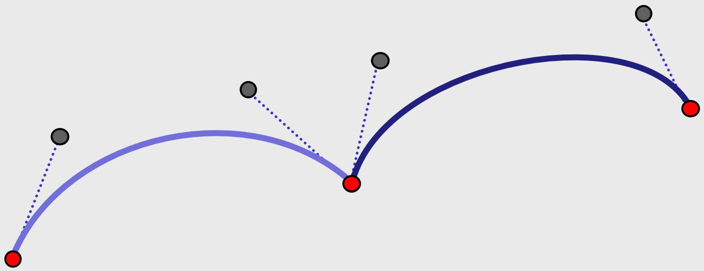
- Additional requirement for C^2 continuity:
  - $\mathbf{d}^- = \mathbf{d}^+$



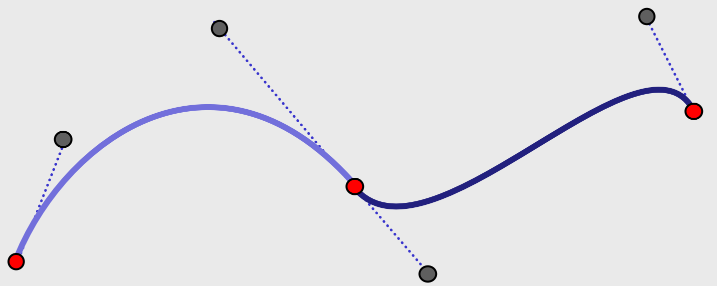
Continuity



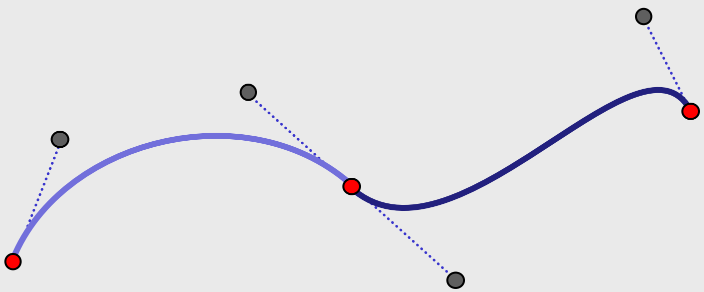
C^{-1} continuity



C^{0} continuity



G^{1} continuity



C^{1} continuity

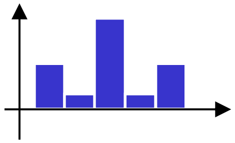
Bezier Splines

Choosing the degree

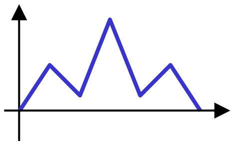
Choosing the Degree...

Candidates:

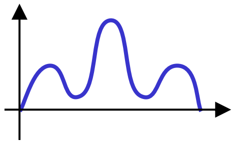
- *d* = 0 (piecewise constant): not smooth



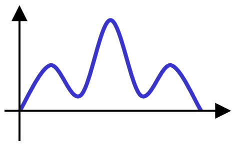
- *d* = 1 (piecewise linear): not smooth enough



- *d* = 2 (piecewise quadratic): constant 2nd derivative, still too inflexible



- *d* = 3 (piecewise cubic): degree of choice for computer graphics applications



Cubic Splines

Cubic piecewise polynomials:

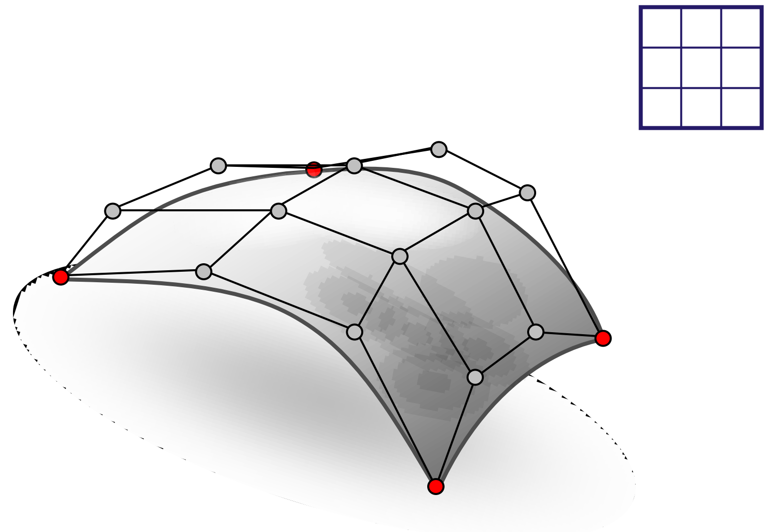
- We can attain C^2 continuity without fixing the second derivative throughout the curve
- C^2 continuity is perceptually important
  - We can see second order shading discontinuities (esp.: reflective objects)
  - Motion: continuous *position*, *velocity* & *acceleration* Discontinuous acceleration noticeable (object/camera motion)
- One more argument for cubics:
  - Among all C^2 curves that interpolate a set of points (and obey to the same end conditions), a piecewise cubic curve has the least integral acceleration ("smoothest curve you can get").
    - see `AdditionalMaterial/CubicsMinimizeAcceleration.pdf`

Spline Surfaces

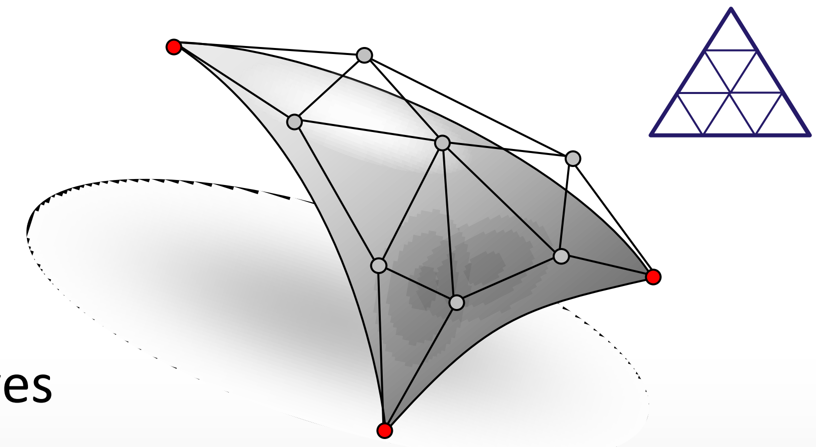
Spline Surfaces

Two different approaches

- Tensor product surfaces
  - Simple construction
  - Everything carries over from curve case
  - Quad patches
  - Degree anisotropy



- Total degree surfaces
  - Not as straightforward
  - Isotropic degree
  - Triangle patches
  - “Natural” generalization of curves

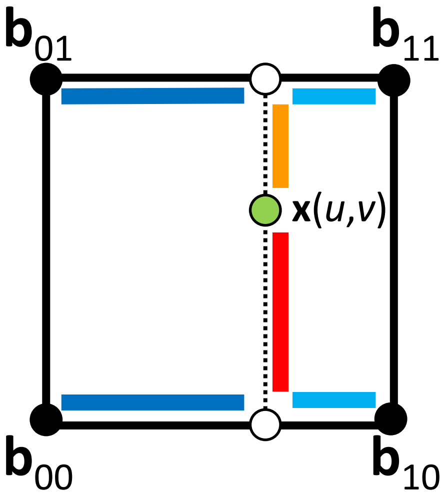


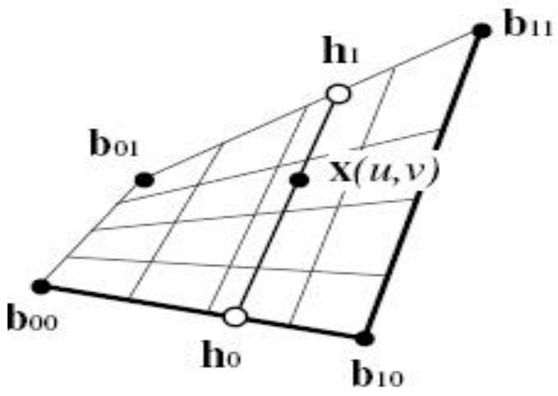
Tensor Product Surfaces

Tensor Product Bezier Surfaces

Bezier curves:
repeated linear interpolation

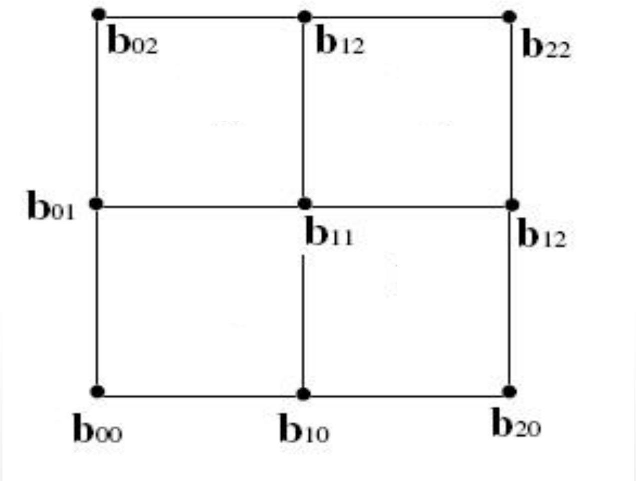
now a different setup:
4 points $\mathbf{b}_{00}$, $\mathbf{b}_{10}$, $\mathbf{b}_{11}$, $\mathbf{b}_{01}$
parameter area $[0,1] \times [0,1]$





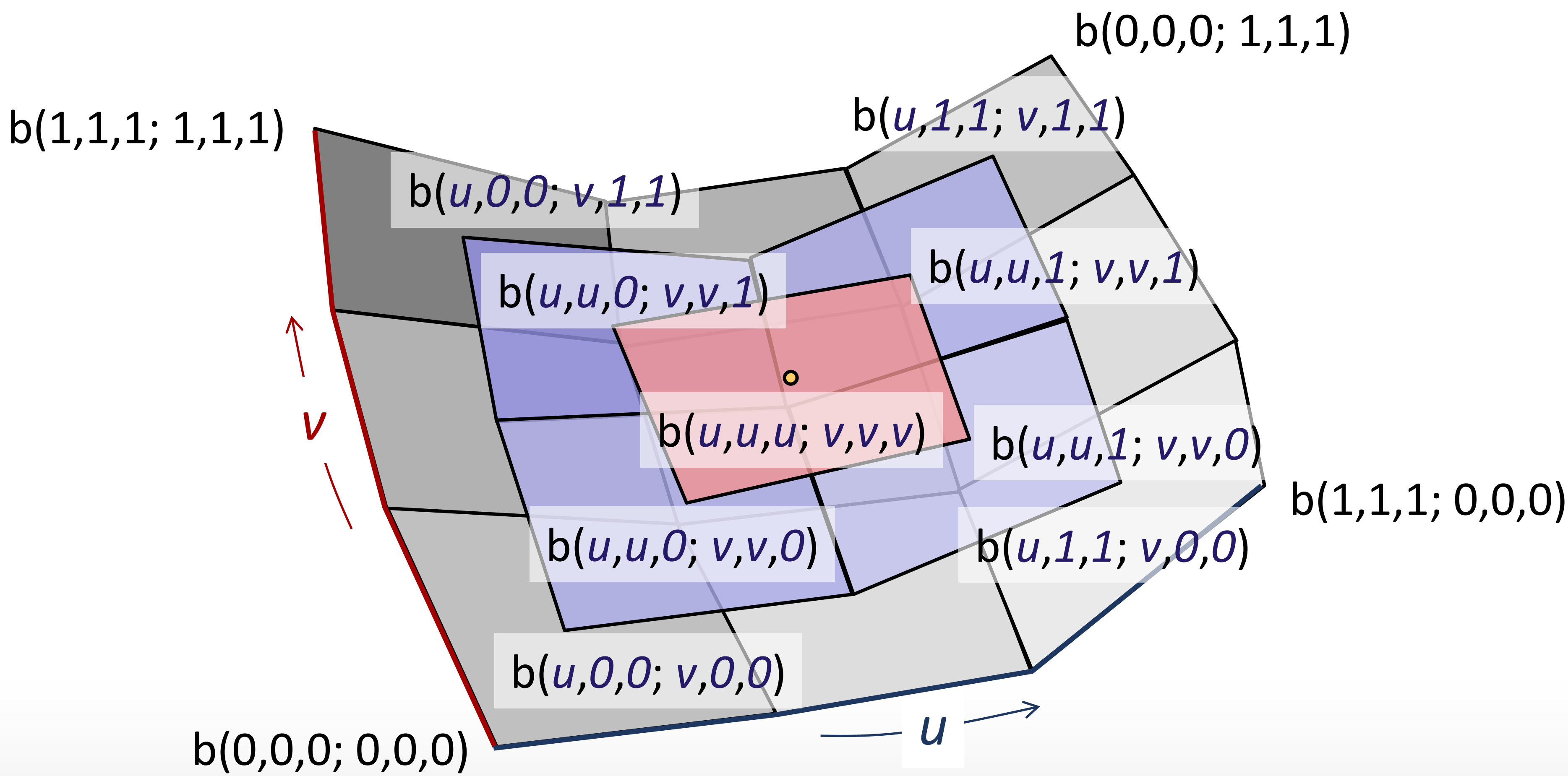
bilinear interpolation:
repeated linear interpolation

repeated bilinear interpolation:
gives us tensor product Bezier surfaces
(example shows quadratic Bezier surface)



De Casteljau Algorithm

De Casteljau algorithm for tensor product surfaces:



Tensor Product Surfaces

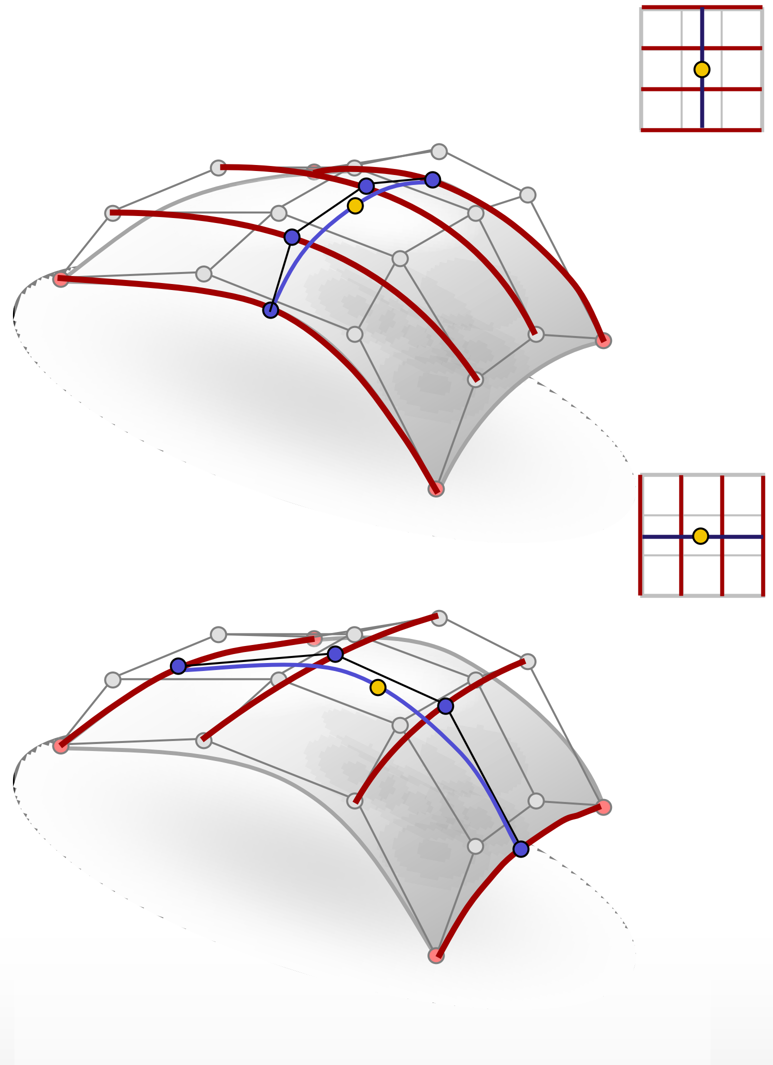
Tensor Product Surfaces:

$$\mathbf{f}(u,v) = \sum_{i=1}^{n}\sum_{j=1}^{n} b_i(u) b_j(v) \mathbf{p}_{i,j}$$

$$= \sum_{i=1}^{n} b_i(u) \sum_{j=1}^{n} b_j(v) \mathbf{p}_{i,j}$$

$$= \sum_{j=1}^{n} b_j(u) \sum_{i=1}^{n} b_i(v) \mathbf{p}_{i,j}$$

- "Curves of Curves"
- Order does not matter



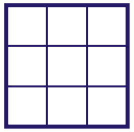
Tensor Product Surfaces

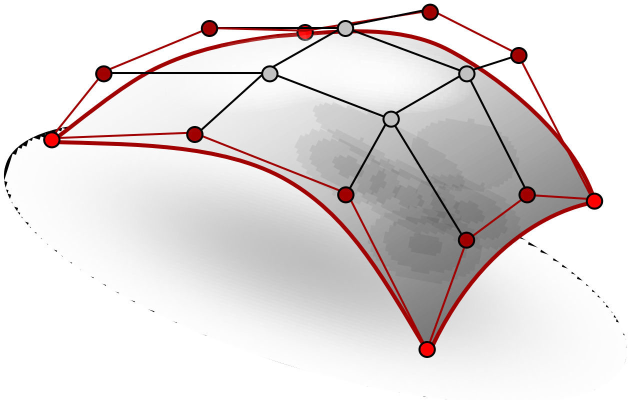
Bezier Patches

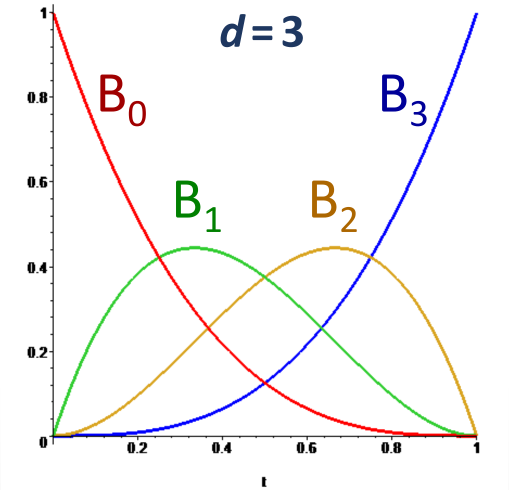
Bezier Patches

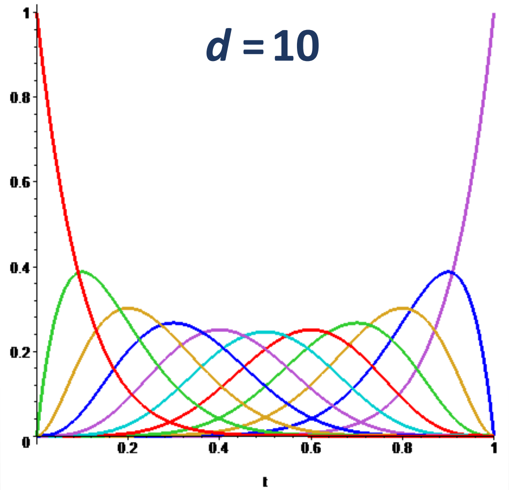
Bezier Patches:

- Remember endpoint interpolation:
  - Boundary curves are Bezier curves of the boundary control points









Continuity Conditions

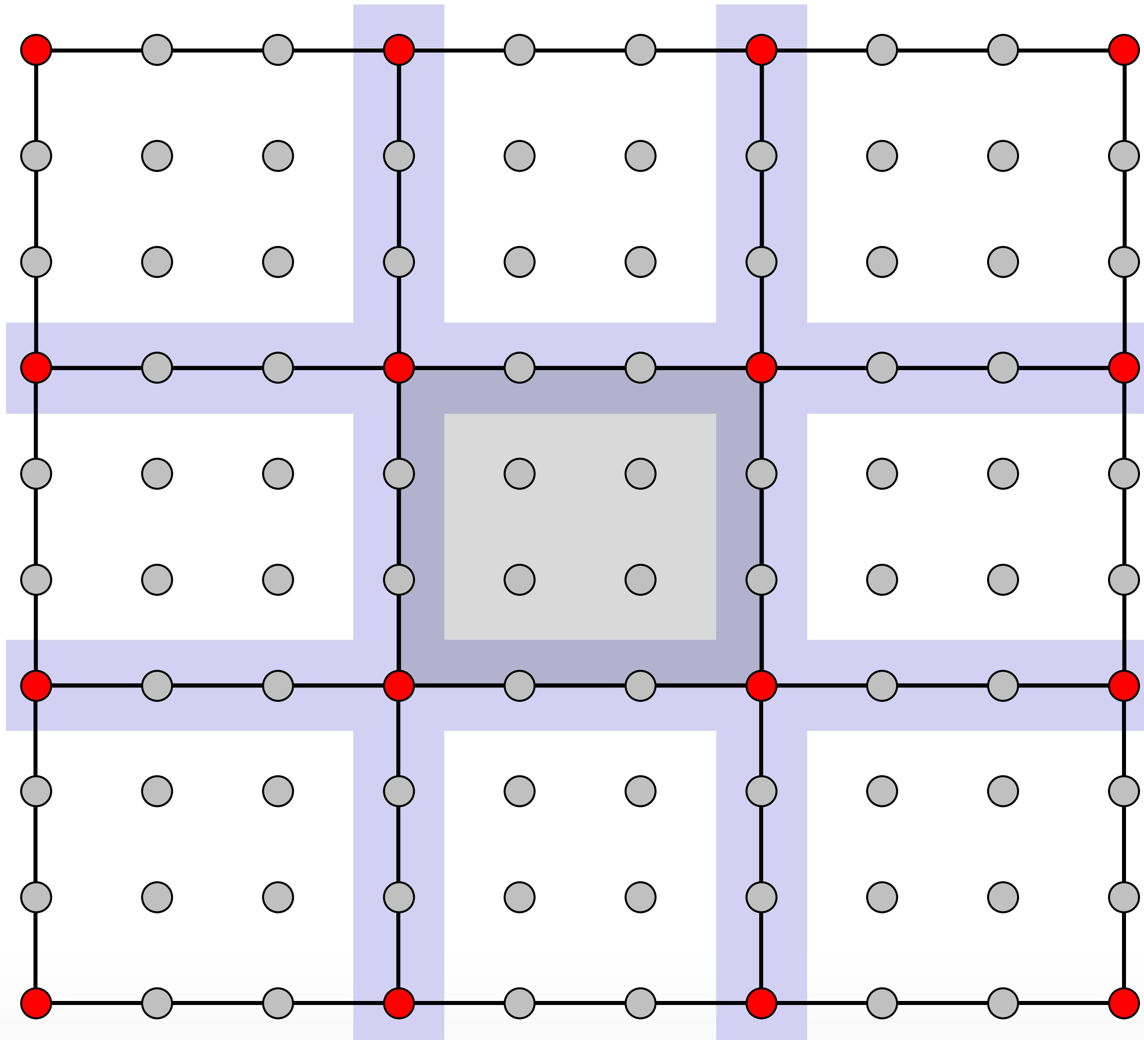
For C^0 continuity:

- Boundary control points must match

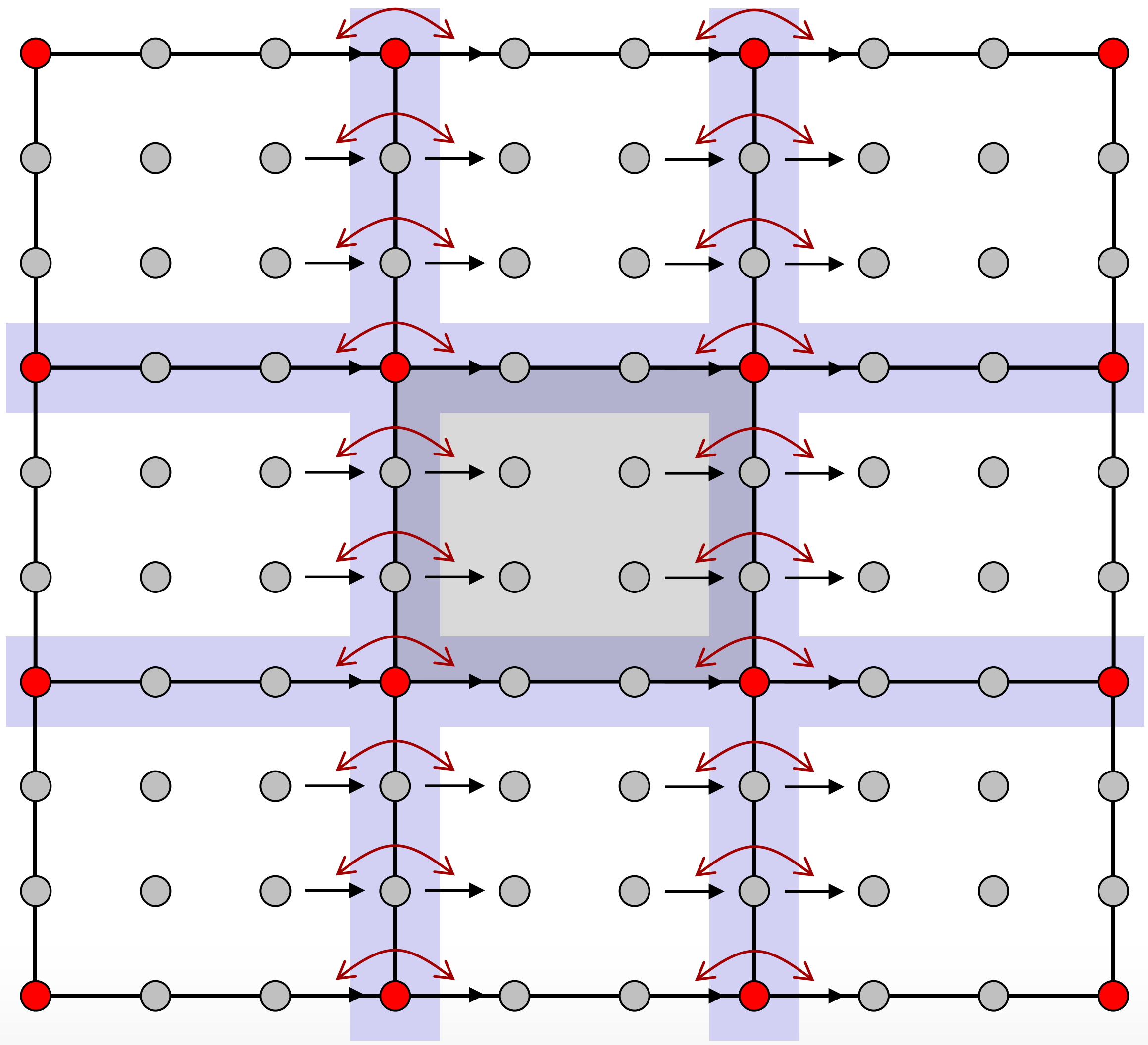
For C^1 continuity:

- Difference vectors must match at the boundary

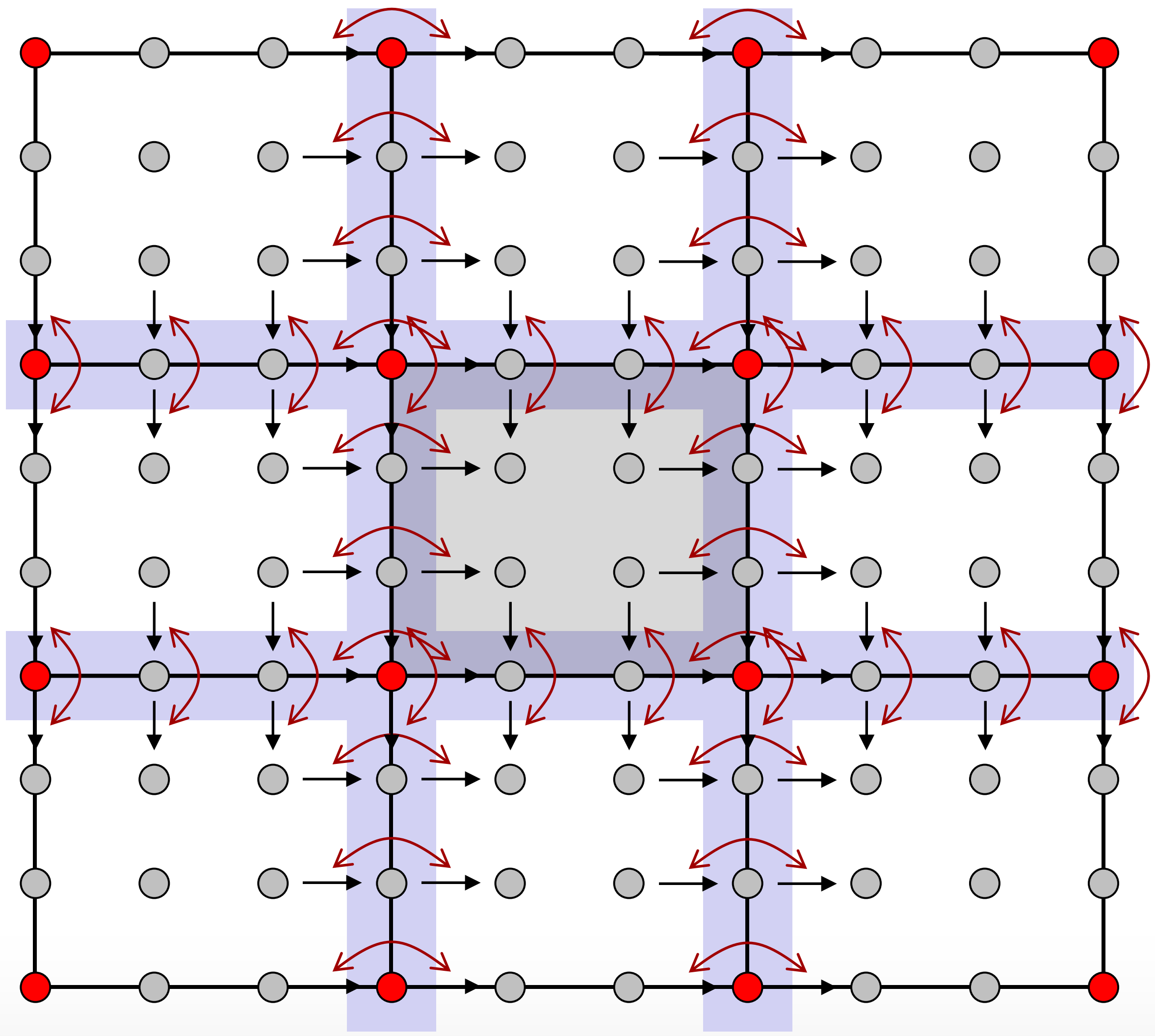
C^0 Continuity



C^1 Continuity



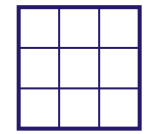
C¹ Continuity

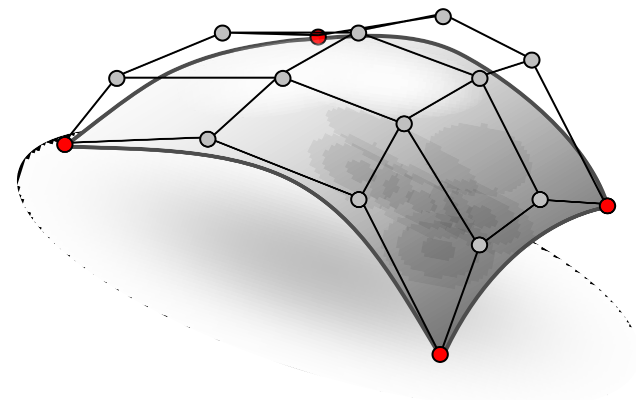


Total Degree Surfaces

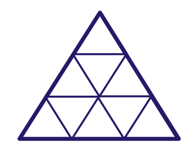
Spline Surfaces

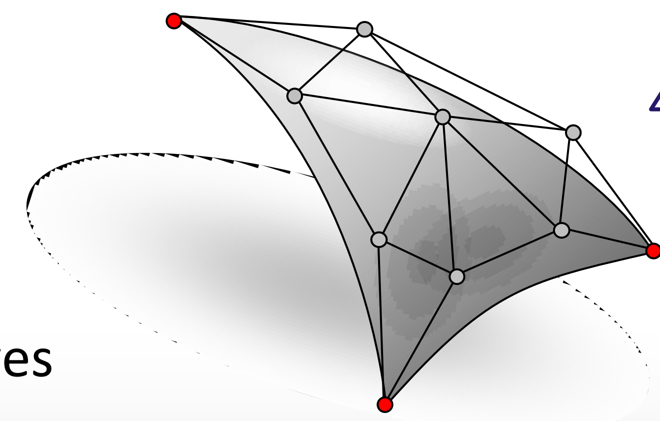
Two different approaches





- Tensor product surfaces
  - Simple construction
  - Everything carries over from curve case
  - Quad patches
  - Degree anisotropy



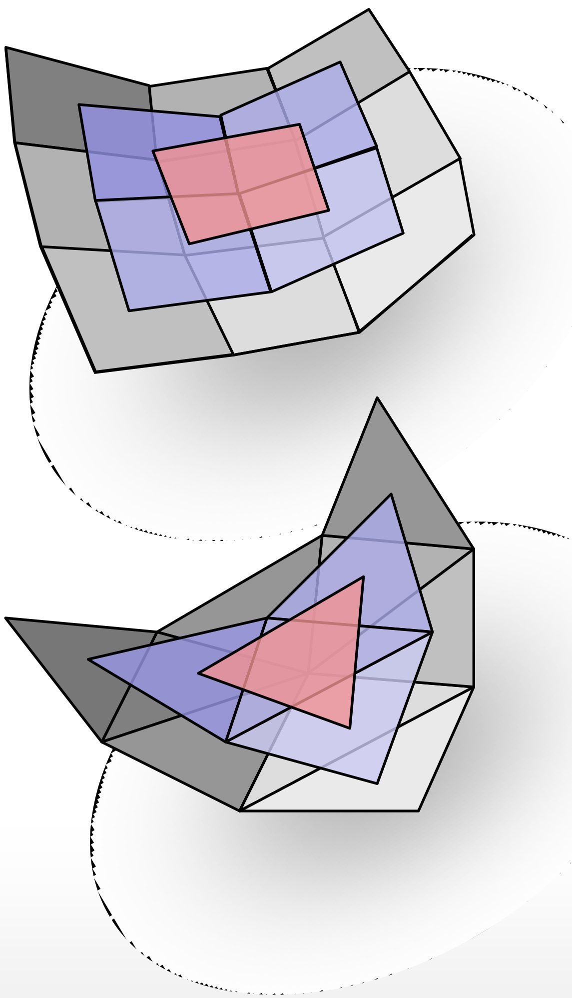


- Total degree surfaces
  - Not as straightforward
  - Isotropic degree
  - Triangle patches
  - “Natural” generalization of curves

Bezier Triangles

Alternative surface definition: Bezier triangles

- Constructed according to given total degree
  - Completely symmetric:
    No degree anisotropy
- Can be derived using a triangular de Casteljau algorithm
  - Barycentric interpolation



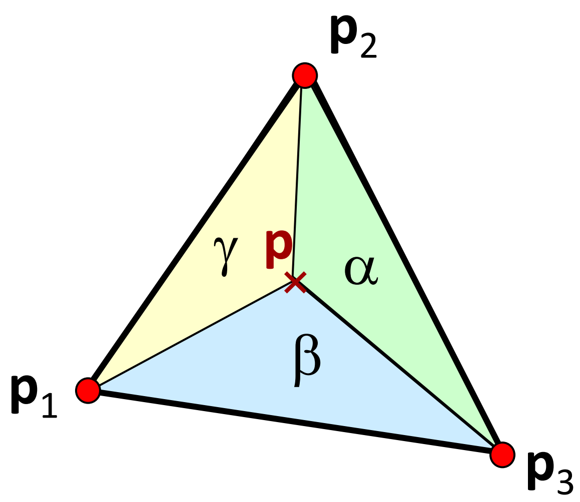
Barycentric Coordinates

Barycentric Coordinates:

- Planar case:
  Barycentric combinations of 3 points

  $$\mathbf{p} = \alpha \mathbf{p}_1 + \beta \mathbf{p}_2 + \gamma \mathbf{p}_3, \text{with} : \alpha + \beta + \gamma = 1$$

  $$\gamma = 1 - \alpha - \beta$$

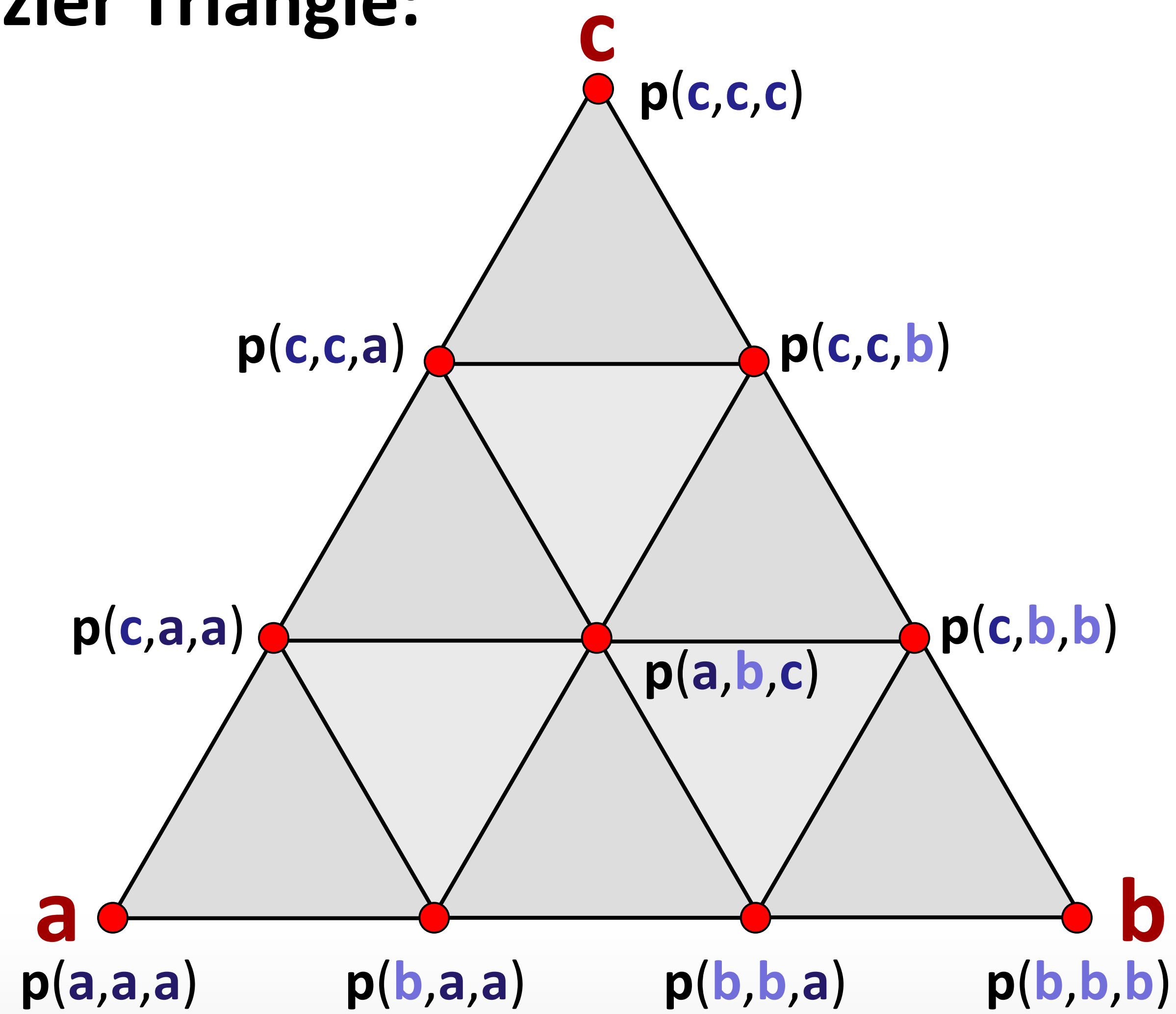


- Area formulation:

$$\alpha = \frac{area(\Delta(\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}))}{area(\Delta(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3))}, \beta = \frac{area(\Delta(\mathbf{p}_1, \mathbf{p}_3, \mathbf{p}))}{area(\Delta(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3))}, \gamma = \frac{area(\Delta(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}))}{area(\Delta(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3))}$$

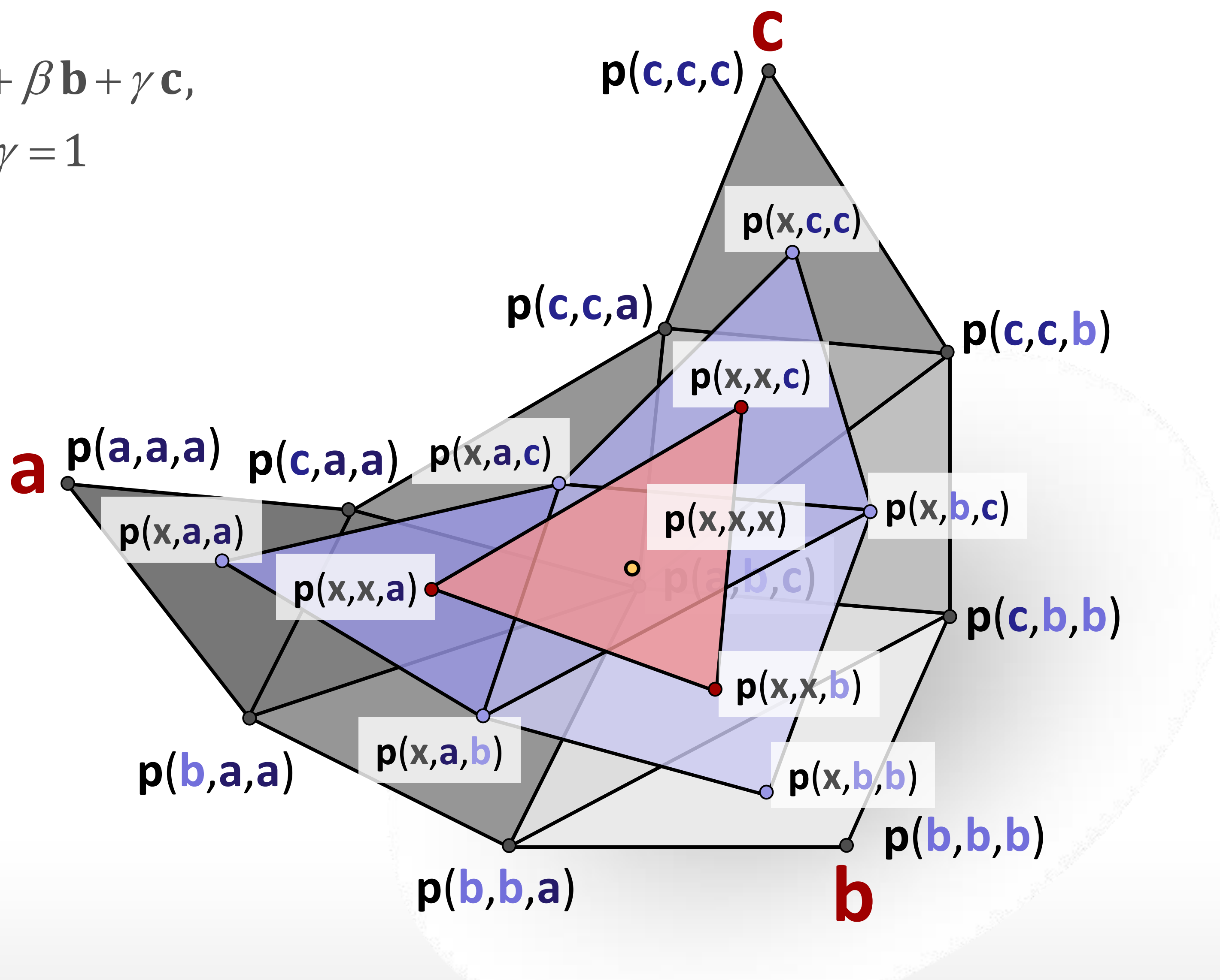
Example

Cubic Bezier Triangle:



De Casteljau Algorithm

$$\mathbf{x} = \alpha\,\mathbf{a} + \beta\,\mathbf{b} + \gamma\,\mathbf{c},$$
$$\alpha + \beta + \gamma = 1$$



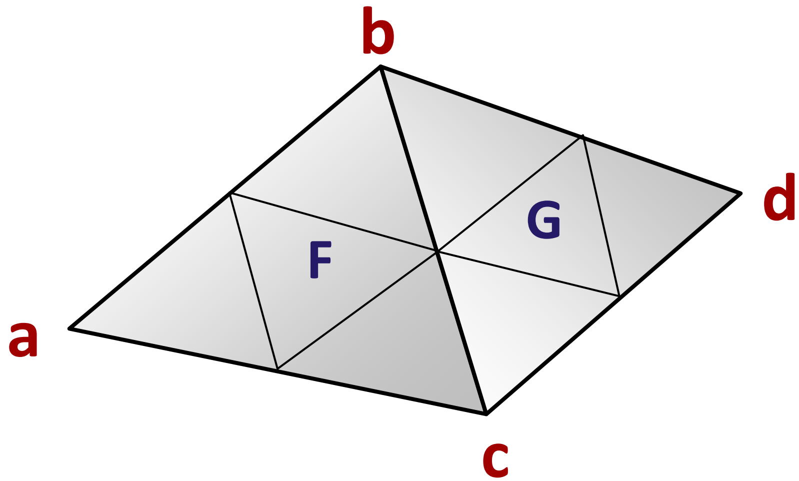
Continuity

We need to assemble Bezier triangles continuously:

- What are the conditions for C^0, C^1 continuity?
- As an example, we will look at the quadratic case...

Continuity

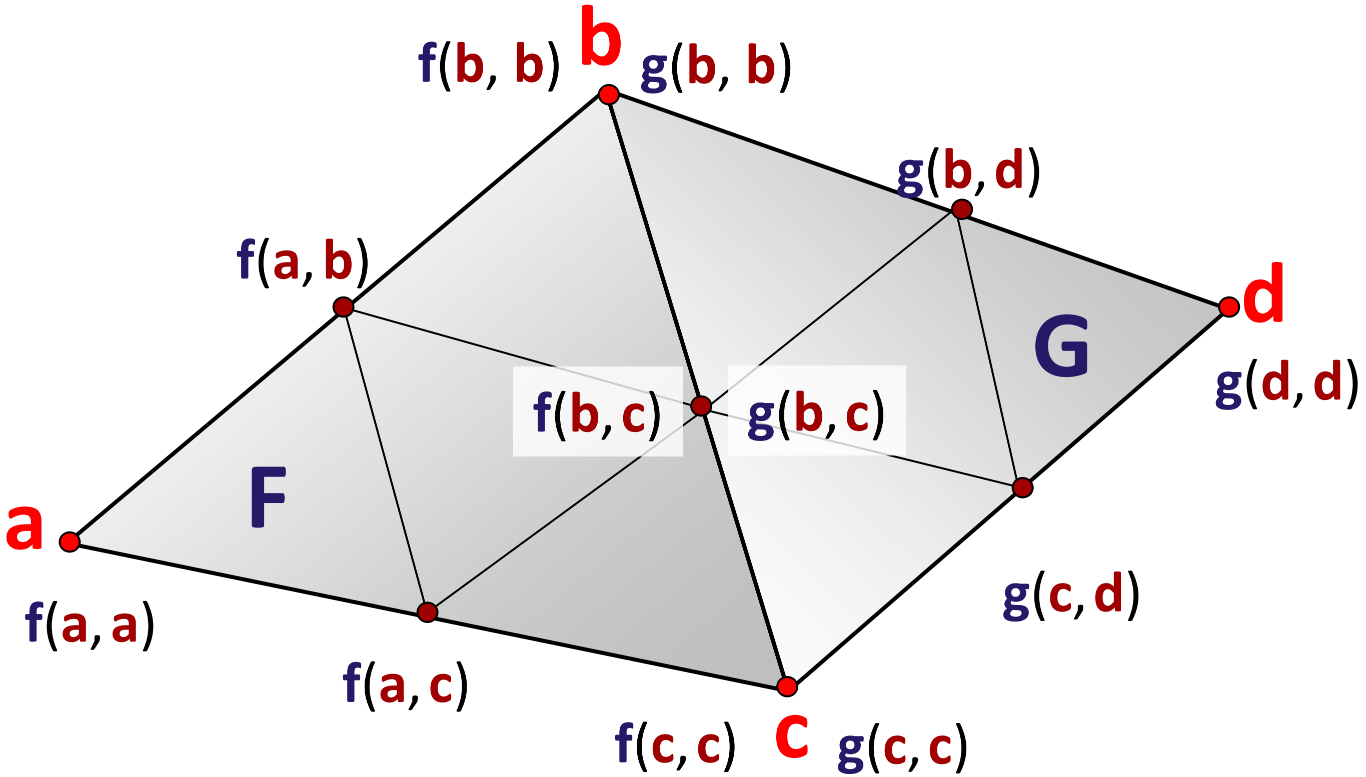
Situation:



- Two Bezier triangles meet along a common edge.
  - Parametrization: T_1 = {**a**, **b**, **c**}, T_2 = {**c**, **b**, **d**}
  - Polynomial surfaces **F**(T_1), **G**(T_2)
  - Control points:
    - **F**(T_1): **f**(**a**, **a**), **f**(**a**, **b**), **f**(**b**, **b**), **f**(**a**, **c**), **f**(**c**, **c**), **f**(**b**, **c**)
    - **G**(T_2): **g**(**d**, **d**), **g**(**d**, **b**), **g**(**b**, **b**), **g**(**d**, **c**), **g**(**c**, **c**), **g**(**b**, **c**)

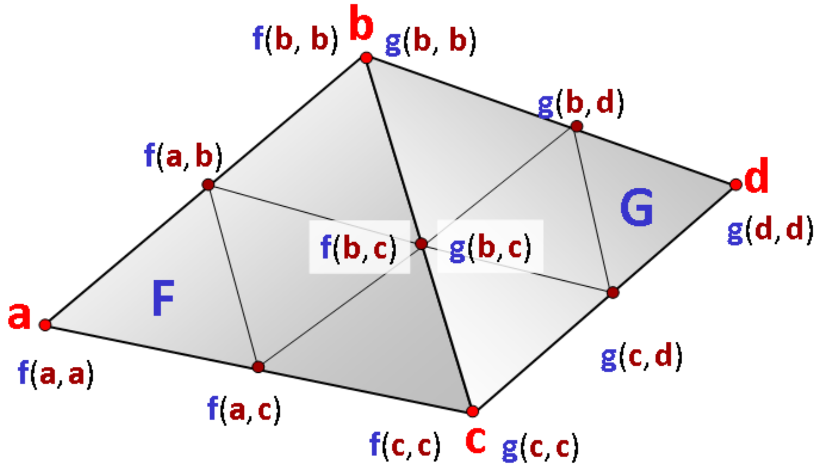
Continuity

Situation:



Continuity

C^0 Continuity:



- The points on the boundary have to agree:

  $\mathbf{f}(\mathbf{b},\mathbf{b}) = \mathbf{g}(\mathbf{b},\mathbf{b})$

  $\mathbf{f}(\mathbf{b},\mathbf{c}) = \mathbf{g}(\mathbf{b},\mathbf{c})$

  $\mathbf{f}(\mathbf{c},\mathbf{c}) = \mathbf{g}(\mathbf{c},\mathbf{c})$

- Proof: Let $\mathbf{x} := \beta\mathbf{b} + \gamma\mathbf{c},\ \beta + \gamma = 1$

$$\begin{aligned}
\mathbf{f}(\mathbf{x},\mathbf{x}) &= \beta\,\mathbf{f}(\mathbf{b},\mathbf{x}) + \gamma\,\mathbf{f}(\mathbf{c},\mathbf{x}) \\
&= \beta^2\,\underset{\displaystyle \mathbf{g}(\mathbf{b},\mathbf{b})}{\underset{\parallel}{\mathbf{f}(\mathbf{b},\mathbf{b})}} + 2\beta\gamma\,\underset{\displaystyle \mathbf{g}(\mathbf{b},\mathbf{c})}{\underset{\parallel}{\mathbf{f}(\mathbf{b},\mathbf{c})}} + \gamma^2\,\underset{\displaystyle \mathbf{g}(\mathbf{c},\mathbf{c})}{\underset{\parallel}{\mathbf{f}(\mathbf{c},\mathbf{c})}} \\
&= \beta^2\,\mathbf{g}(\mathbf{b},\mathbf{b}) + 2\beta\gamma\,\mathbf{g}(\mathbf{b},\mathbf{c}) + \gamma^2\,\mathbf{g}(\mathbf{c},\mathbf{c}) \\
&= \beta\,\mathbf{g}(\mathbf{b},\mathbf{x}) + \gamma\,\mathbf{g}(\mathbf{c},\mathbf{x}) = \mathbf{g}(\mathbf{x},\mathbf{x})
\end{aligned}$$

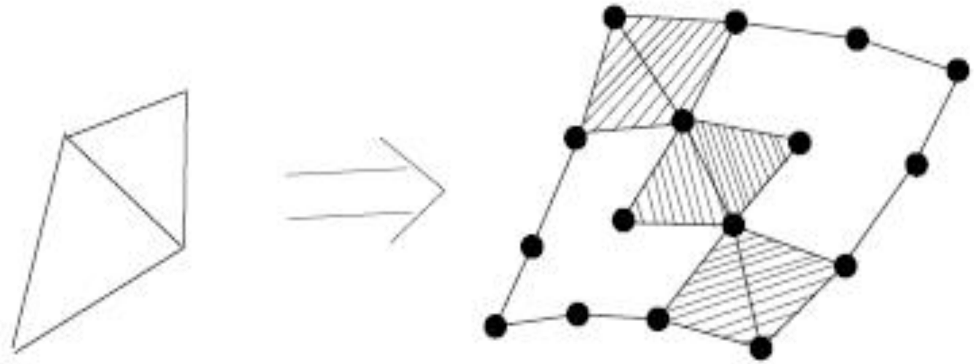
Continuity

C^1 Continuity:

- We need C^0 continuity.

In addition:

- Points at hatched quadrilaterals are coplanar
- Hatched quadrilaterals are an affine image of the same parameter quadrilateral



Curves on Surfaces, trimmed NURBS

Quad patch problem:

- All of our shapes are parameterized over rectangular or triangular regions
- General boundary curves are hard to create
- Topology fixed to a disc (or cylinder, torus)
- No holes in the middle
- Assembling complicated shapes is painful
  - Lots of pieces
  - Continuity conditions for assembling pieces become complicated
  - Cannot use C^2 B-Splines continuity along boundaries when using multiple pieces

Curves on Surfaces, trimmed NURBS

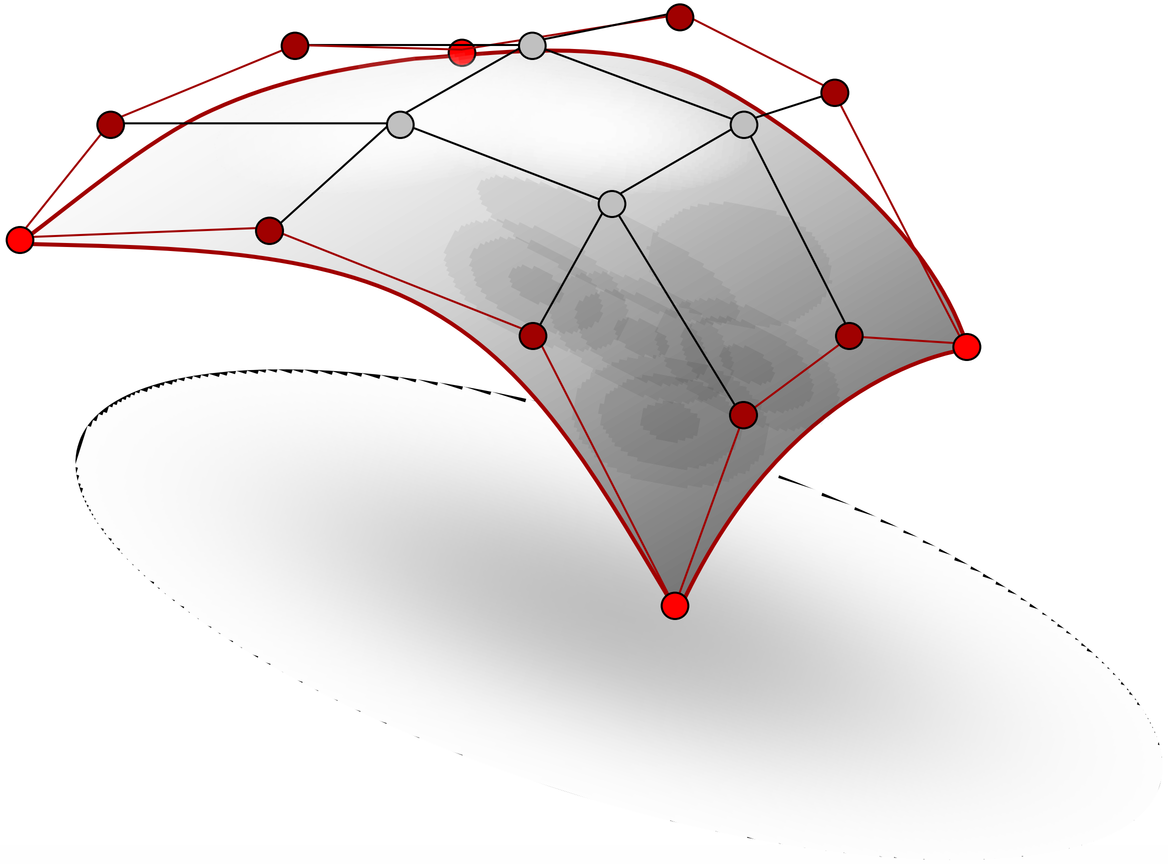
Consequence:

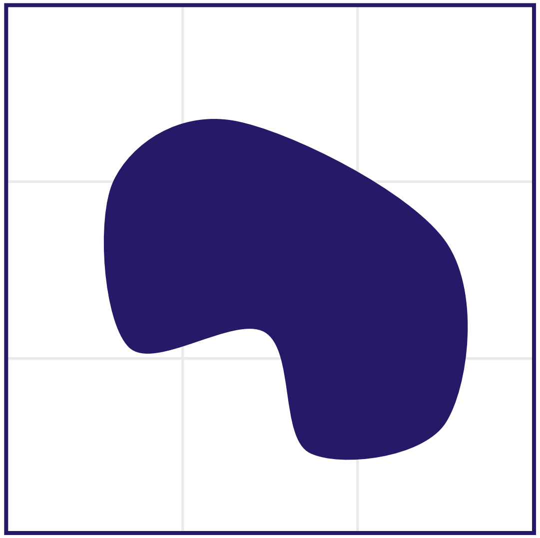
- We need more control over the parameter domain
- One solution is *trimming* using *curves on surfaces (CONS)*
- Standard tool in CAD: *trimmed NURBS*

Basic idea:

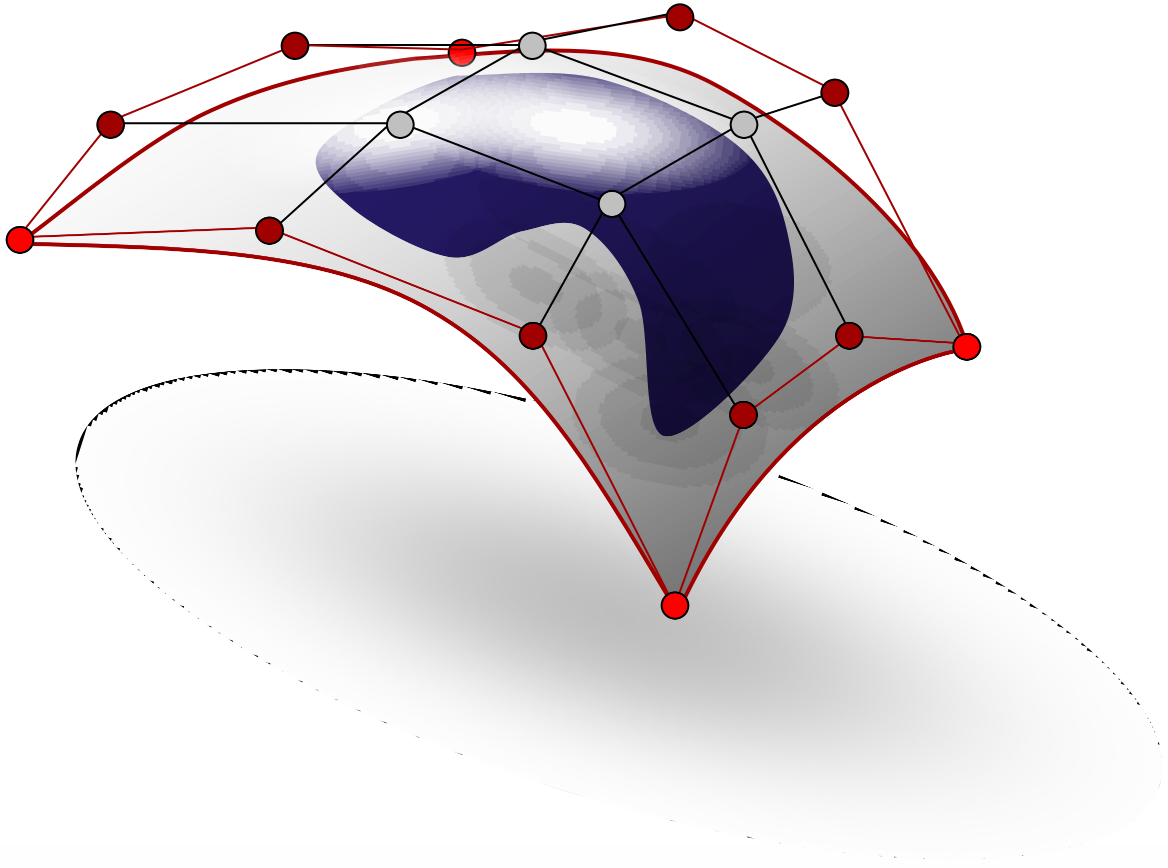
- Specify a curve in the parameter domain that encapsulates one (or more) pieces of area
- Tessellate the parameter domain accordingly to cut out the trimmed piece (rendering)

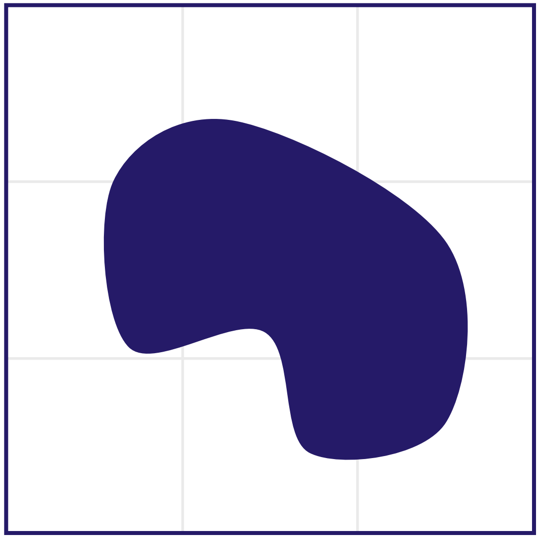
Curves-on-Surfaces (CONS)



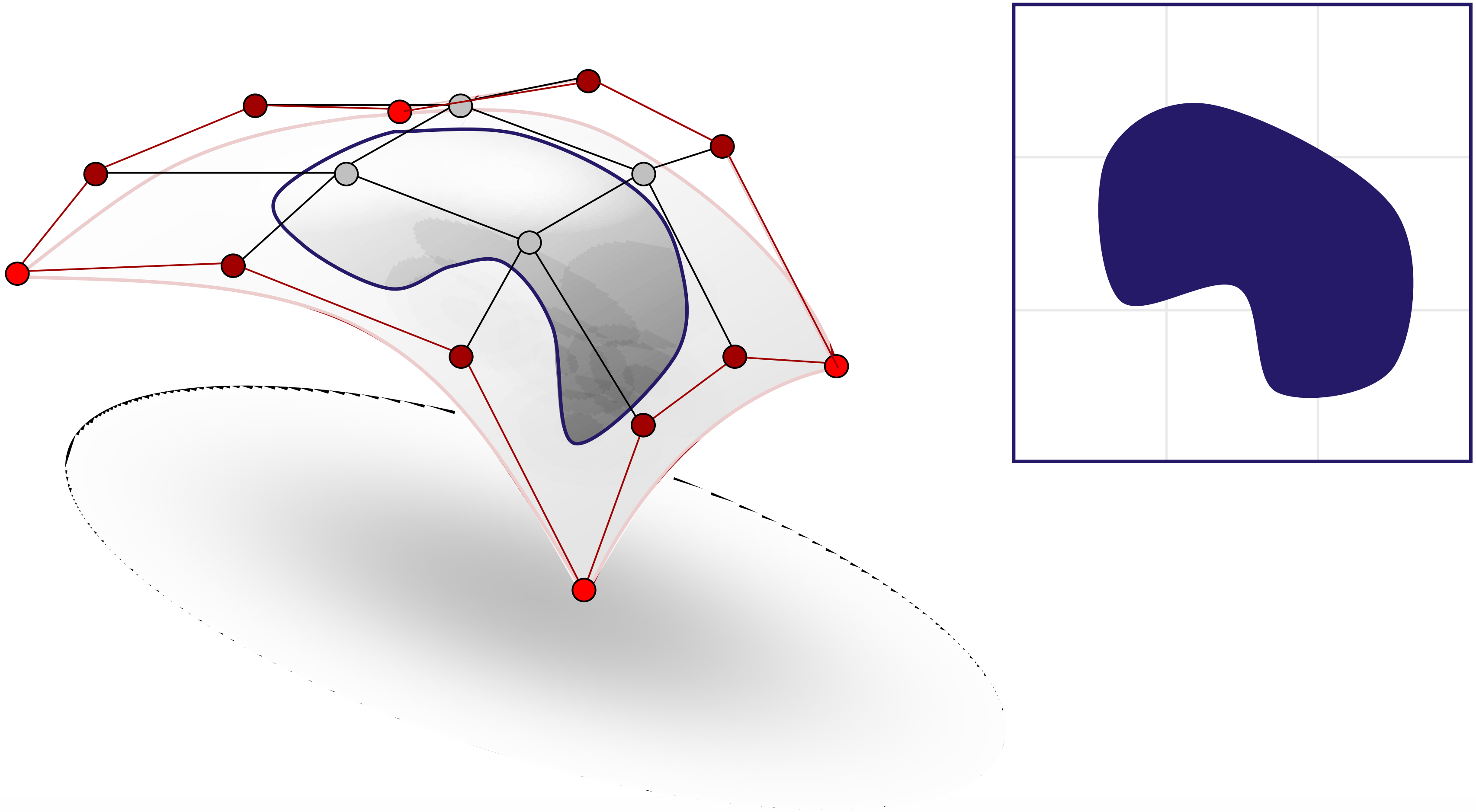


Curves-on-Surfaces (CONS)





Curves-on-Surfaces (CONS)



Summary

- Bezier Curves
  - de Casteljau algorithm
  - Bernstein form
- Bezier Splines
- Bezier Tensor Product Surfaces
- Bezier Total Degree Surfaces