

Algorithmik kontinuierlicher Systeme

Bézier-Kurven, Modellierung von Freiformkurven und -flächen



- **Regelgeometrie:** 1D : Linie, Kreis(-bogen)
2D : Ebene, Kugel, Kegel, Zylinder
- **Freiformgeometrie:** alles andere
Beispiele: Karosserieteile, Tragflächen, Telefonhörer...

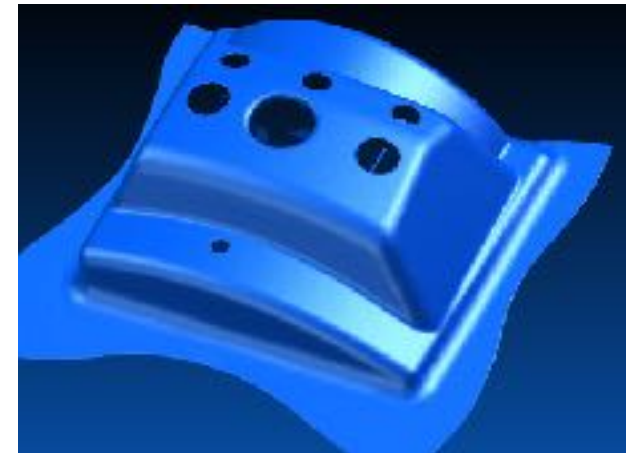
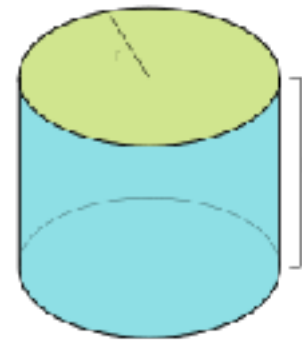
1D (Kurven)

- Regel



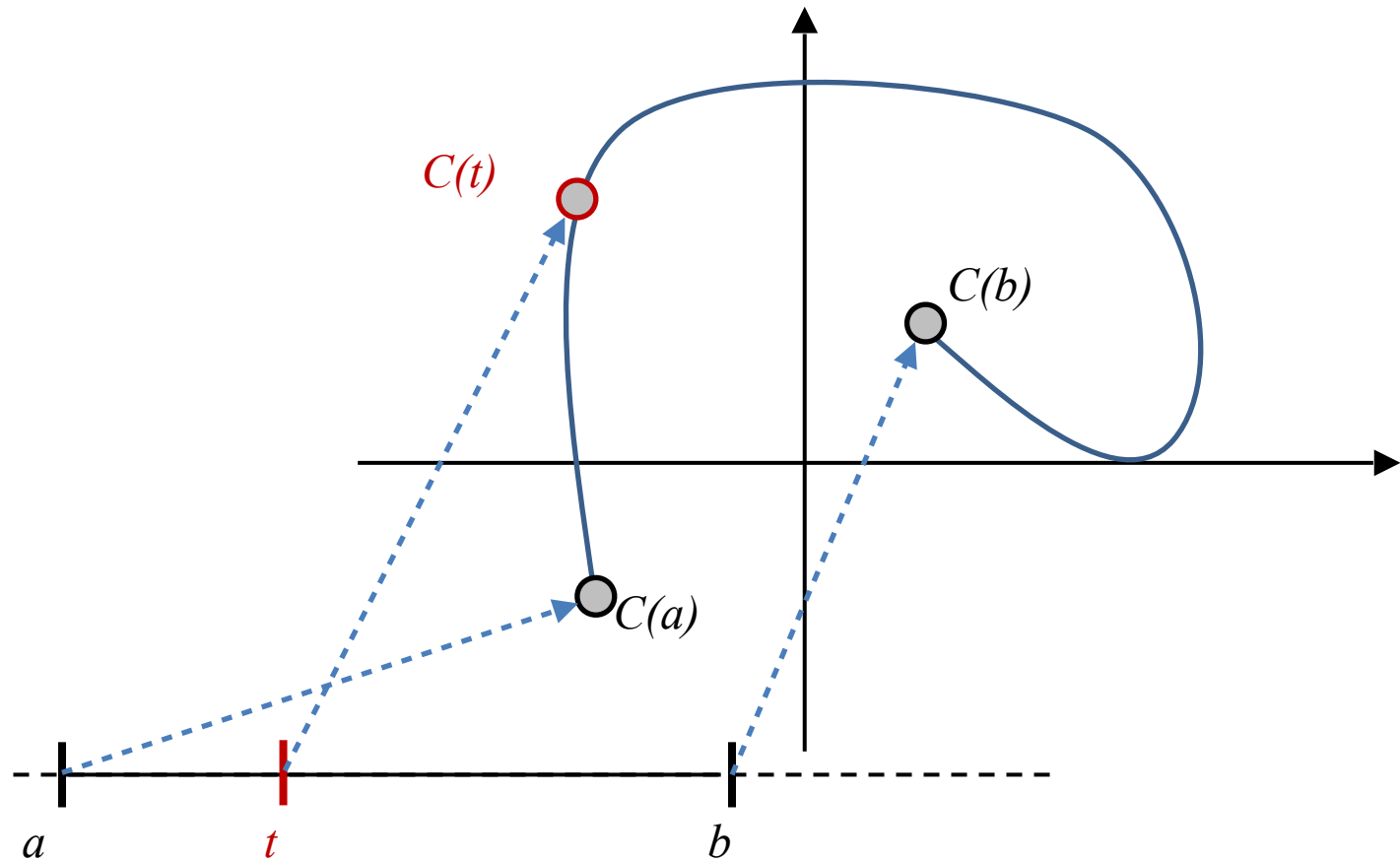
Freiform

2D (Flächen)



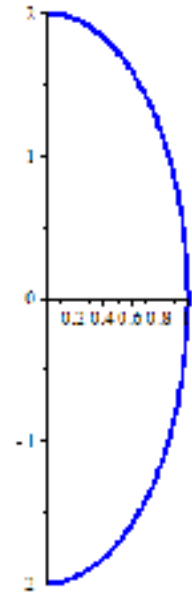
- Freiformkurven werden meist parametrisch beschrieben:

$$C : [a, b] \rightarrow \mathbb{R}^d, \quad (d = 2, 3, \dots)$$

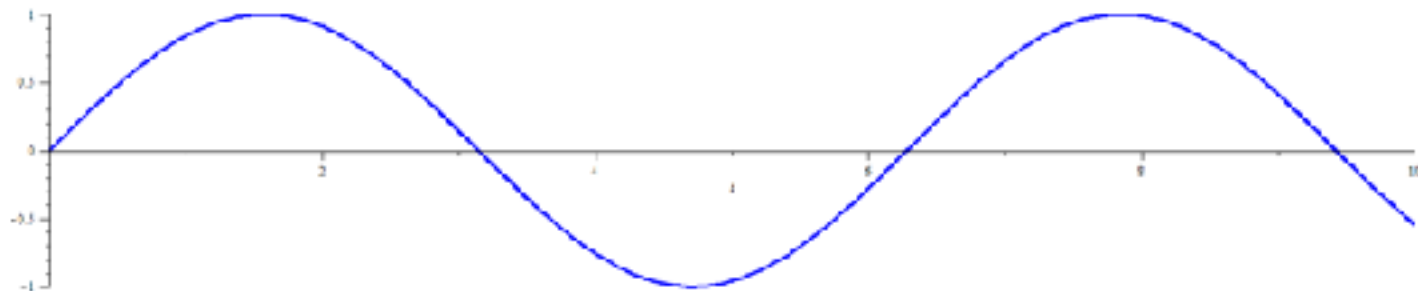


- $C : [a,b] \rightarrow \mathbb{R}^d$, ($d = 2,3,\dots$)
- Konkrete Beispiele:

$$C_1 : [0,\pi] \rightarrow \mathbb{R}^2, C_1(t) = [\sin(t), 2 \cdot \cos(t)]$$

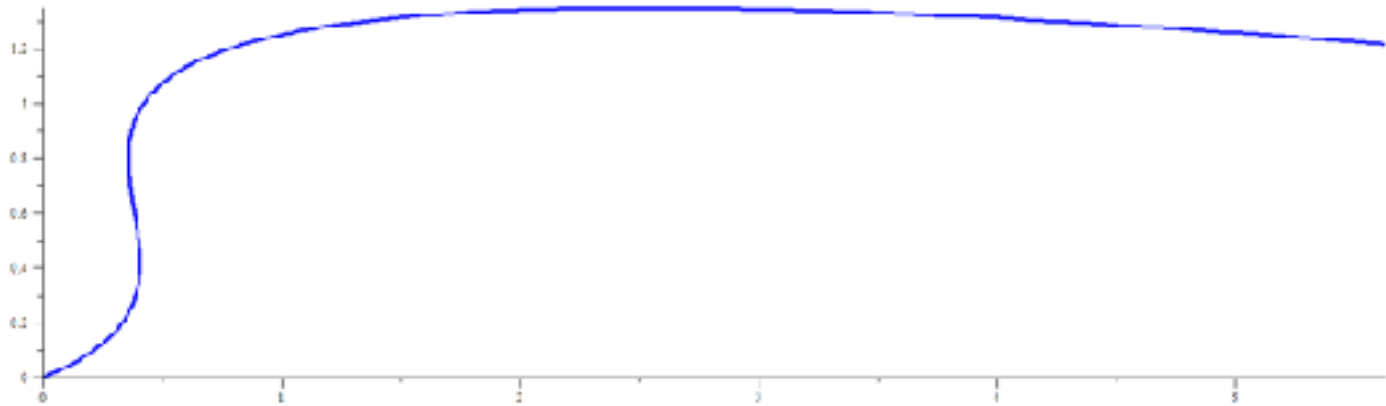


$$C_2 : [0,10] \rightarrow \mathbb{R}^2, C_2(t) = [t, \sin(t)]$$



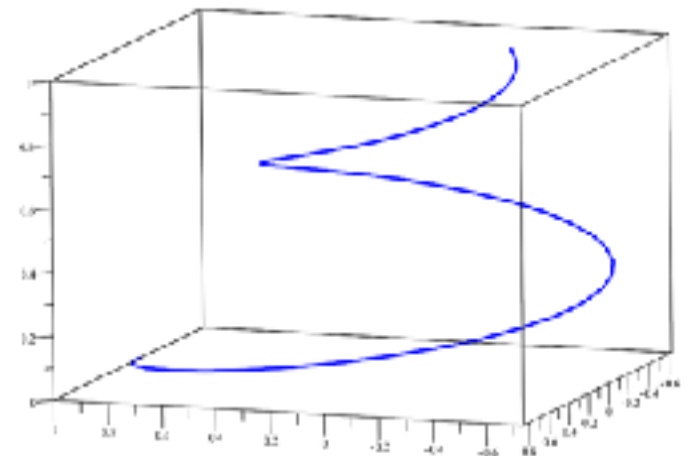
- $C : [a,b] \rightarrow \mathbb{R}^d$, ($d = 2,3,\dots$)
- Konkrete Beispiele:

$$C_3 : [0,1.5] \rightarrow \mathbb{R}^2, C_3(t) = [5t^3-7t^2+3t, -(3/4)t^3+t^2+t]$$

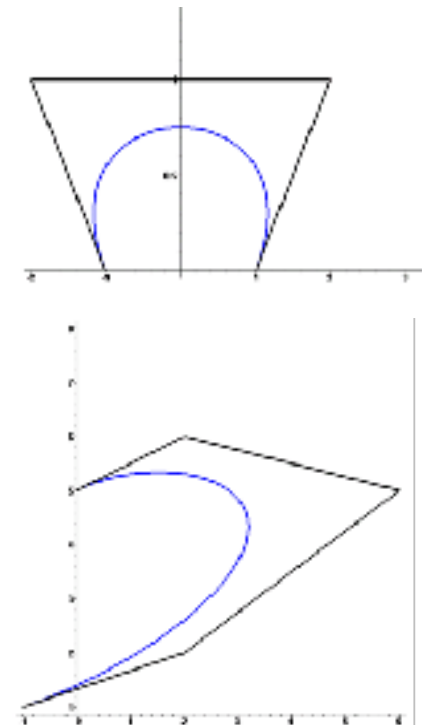


$$C_4 : [0,4\pi] \rightarrow \mathbb{R}^3,$$

$$C_4(t) = [e^{-t} \cdot \cos(10t), e^{-t} \cdot \sin(10t), t]$$



- Anforderungen: Intuitive Benutzerschnittstelle
- Kontrollierbarkeit:
Benutzer manipuliert gewisse Parameter der Kurve/
Fläche.
 - ▶ vorhersagbarer und kontrollierbarer Effekt auf die Gestalt der Kurve.
 - ▶ Polynome $p(t) = a_0 + a_1t + a_2t^2 + \dots + a_nt^n$
oder stückweise Polynome (Splines):
→ einfache Verarbeitung im Rechner
 - ▶ ungünstig: Direkte Eingabe der a_i
(Koeffizienten in der Monom-Basis)
 - ▶ besser: Eingabe einiger Kurvenpunkte,
dann Interpolation → die Kurve verläuft
immer durch diese Punkte.
 - ▶ noch besser: **Kontrollpunkte**



- **Lokalität:**
Eine lokale Veränderung der Eingabedaten (z.B. eines Stützwertes) darf möglichst nur lokale Auswirkungen haben, idealerweise ändert sich die Kurvengestalt nur in einem kleinen Bereich um den Stützpunkt.
 - ▶ schlecht: B-Spline-**Interpolation**
 - ▶ besser: Catmull-Rom
 - ▶ (stückweise) Bézier

- **Die wichtigsten Ansätze sind:**
 - ▶ Bézier-Kurven (Grundlagen werden besprochen)
 - ▶ B-Splines mit **Kontrollpunkten** (voraussichtlich nicht in diesem Semester)
 - ▶ NURBS (Non-Uniform Rational B-Splines – sh. Vorlesung über Geometrische Modellierung)

- Die Bernstein-Polynome vom Grad n

$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} t^i \quad (i = 0, 1, 2, \dots, n)$$

bilden ein Basis der Polynome (vom Grad n)

- Die Binomial-Koeffizienten $\binom{n}{i} = \frac{n!}{i! \cdot (n-i)!}$

- Rekursive Definition (Pascalsches Dreieck) $\binom{n+1}{i} = \binom{n}{i} + \binom{n}{i-1}$

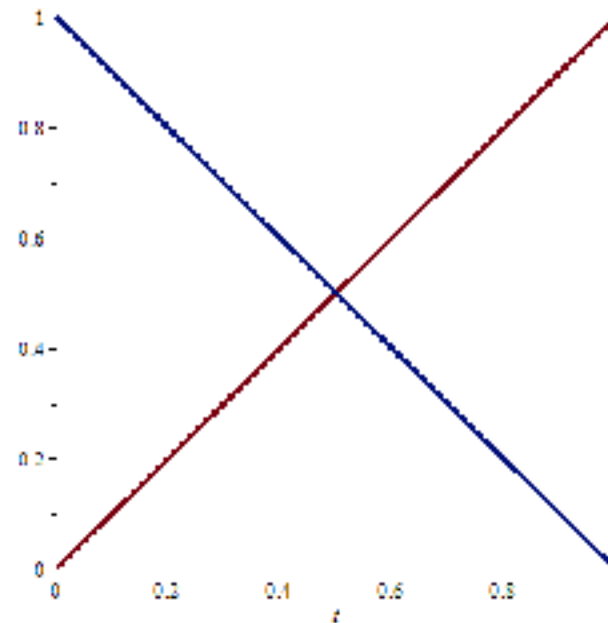
- Binomische Formel

$$(a+b)^n = \sum_{i=0}^n \binom{n}{i} \cdot a^{n-i} b^i$$

$$\begin{array}{ccccccc}
 & & & & & & 1 \\
 & & & & & 1 & 1 \\
 & & & 1 & 2 & 1 \\
 & & 1 & 3 & 3 & 1 \\
 1 & 4 & 6 & 4 & 1
 \end{array}$$

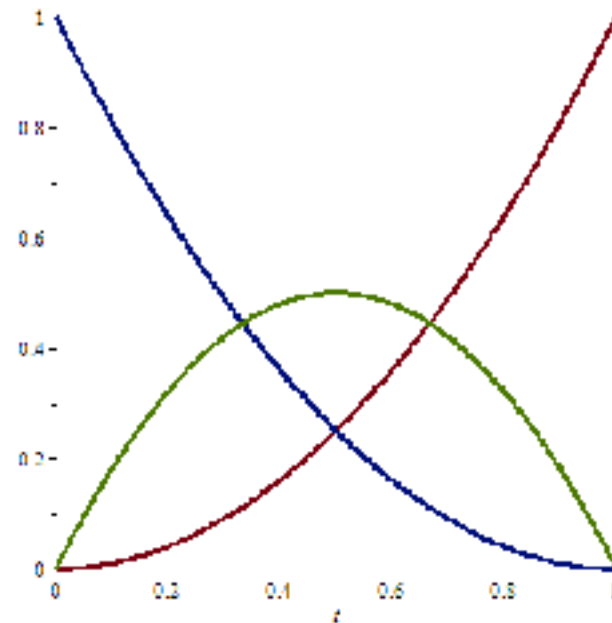
- Bernstein-Polynome vom Grad $n=1$

$$B_0^1(t) = 1 - t, \quad B_1^1(t) = t;$$



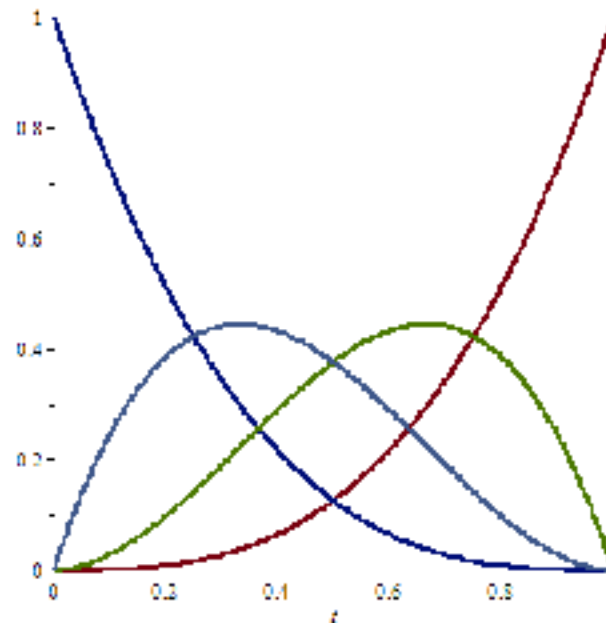
- Bernstein-Polynome vom Grad $n=2$
(quadratische Bernstein-Polynome)

$$B_0^2(t) = (1-t)^2, \quad B_1^2(t) = 2(1-t)t, \quad B_2^2(t) = t^2;$$



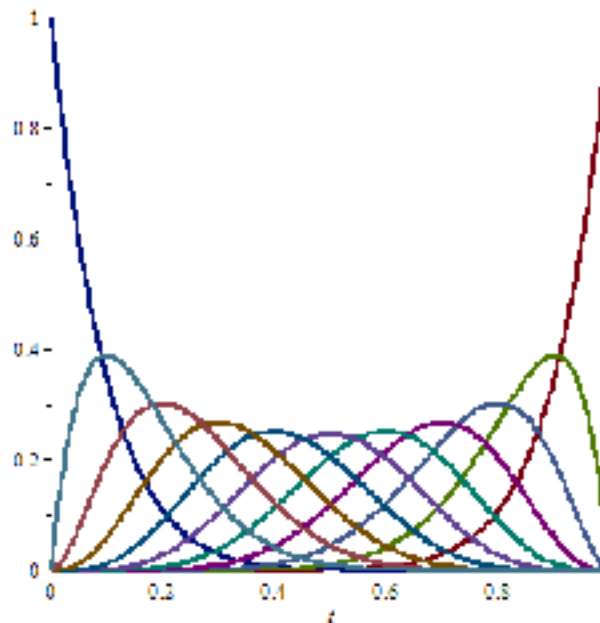
- Bernstein-Polynome vom Grad $n=3$
(kubische Bernstein-Polynome)

$$B_0^3(t) = (1-t)^3, B_1^3(t) = 3(1-t)^2 t, B_2^3(t) = 3(1-t)t^2, B_3^3(t) = t^3;$$



- Bernstein-Polynome vom Grad $n=10$

$$B_0^{10}(t) = (1-t)^{10}, B_1^{10}(t) = 10(1-t)^9 t, \dots, B_{10}^{10}(t) = t^{10};$$



$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} t^i \quad (i = 0, 1, 2, \dots, n)$$

- Es gilt

$$0 \leq B_i^n(t) \leq 1 \quad \text{für } t \in [0, 1]$$

$$B_i^n(t) \quad \text{hat eine } i\text{-fache Nullstelle in } t = 0$$

$$B_i^n(t) \quad \text{hat eine } (n-i)\text{-fache Nullstelle in } t = 1$$

$$\sum_{i=0}^n B_i^n(t) = 1 \quad \text{für alle } t$$

Sergei Natanovich Bernstein
1880-1968
russischer Mathematiker



Pierre Étienne Bézier
1910-1999
französischer Ingenieur (Renault)

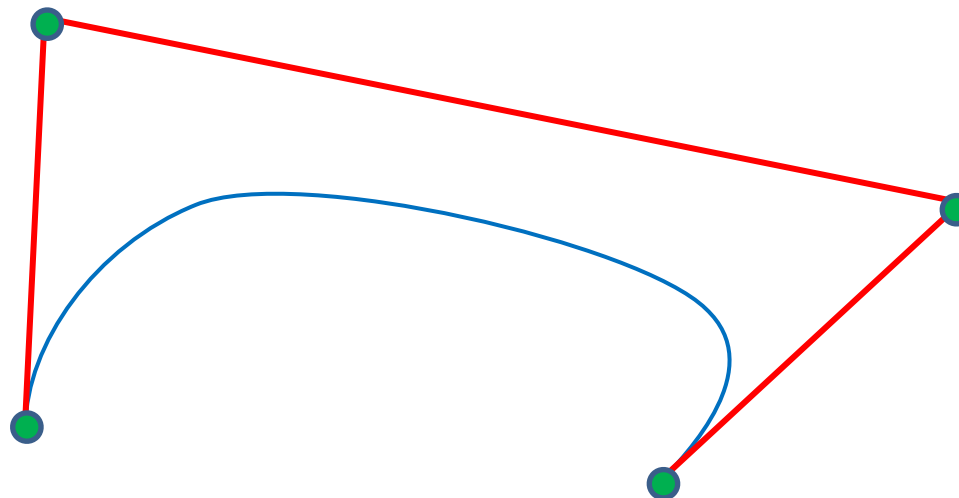
- Gegeben sind sog. **Kontrollpunkte** (auch **Bézier-Punkte**)

$$b_0, b_1, \dots, b_{n-1}, b_n \in \mathbb{R}^d$$

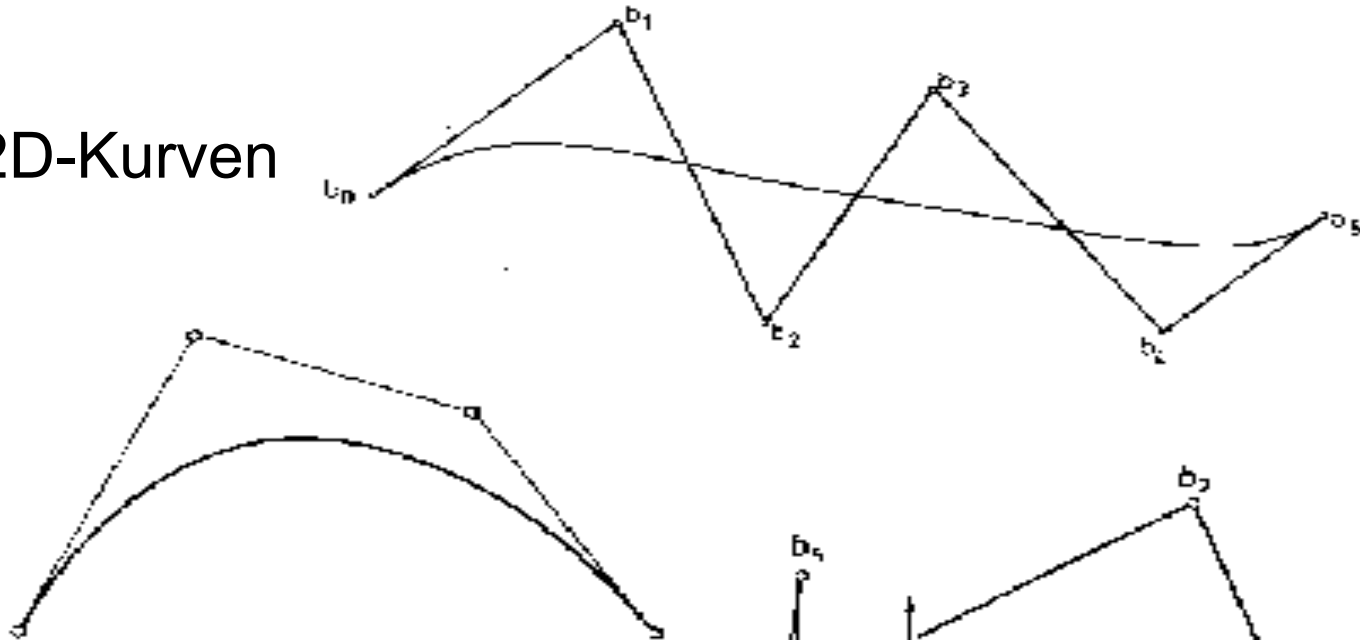
- Die Kurve im \mathbb{R}^d $C(t) = \sum_{i=0}^n b_i B_i^n(t)$ mit $t \in [0,1]$

heißt **Bézier-Kurve** vom Grad n

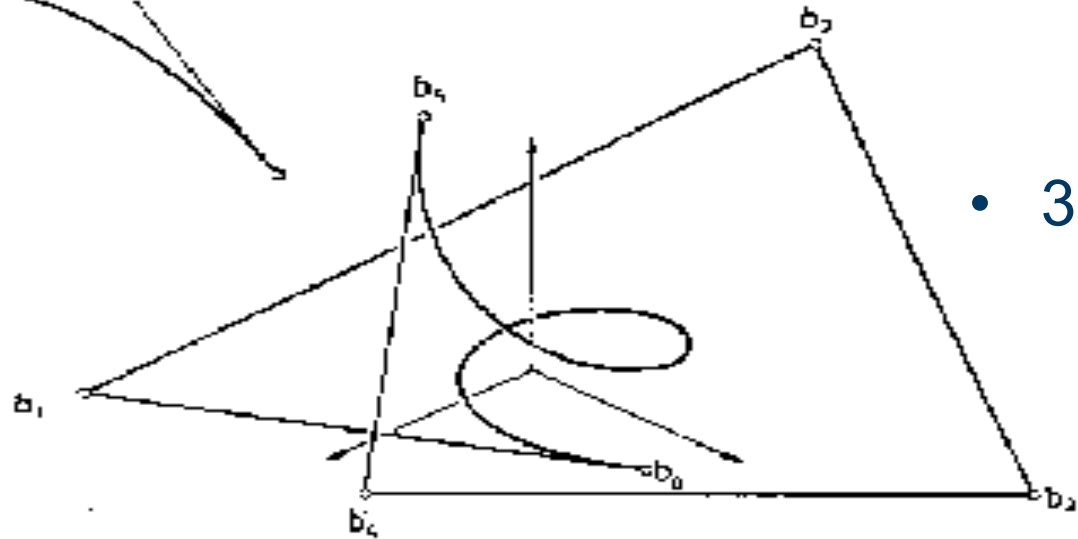
- Verbindet man die **Bézier-Punkte** durch einen Polygonzug erhält man das **Kontrollpolygon** der **Bézier-Kurve**.



- 2D-Kurven



- 3D-Kurve



- Geometrie des Kontrollpolygons spiegelt (grob) die Geometrie der Kurve wider
- **Formeigenschaften**
 1. Interpolation der Endpunkte
 2. In den Endpunkten tangential an das Kontrollpolygon
 3. Bézier-Kurve liegt in der konvexen Hülle der Kontrollpunkte
 4. affine Invarianz
 5. Variationsreduzierend
- Konsequenz: Modifikation der Kontrollpunkte führt zu vorhersehbarer (intuitiver) Änderung der Bézier-Kurve.

- **Endpunkt-Interpolation**

$$C(0) = \sum_{i=0}^n b_i \cdot B_i(0) = b_0$$

$$C(1) = \sum_{i=0}^n b_i \cdot B_i(1) = b_n$$

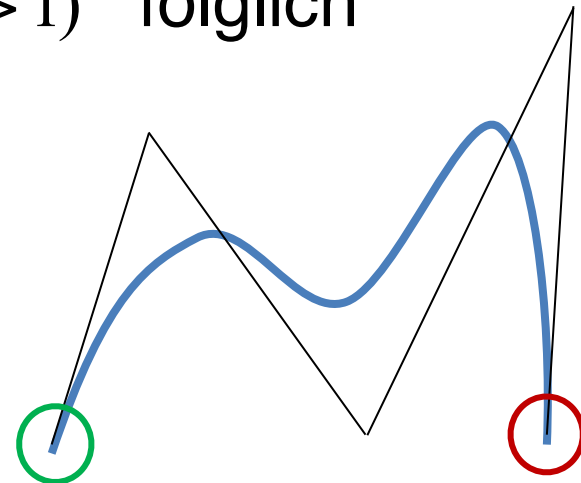
- **Tangentenbedingung**

$$C'(t) = \sum_{i=0}^n b_i \cdot B_i'(t), \text{ dabei}$$

$$B_0'(0) = -n, \quad B_1'(0) = n, \quad B_i'(0) = 0 \quad (i > 1) \quad \text{folglich}$$

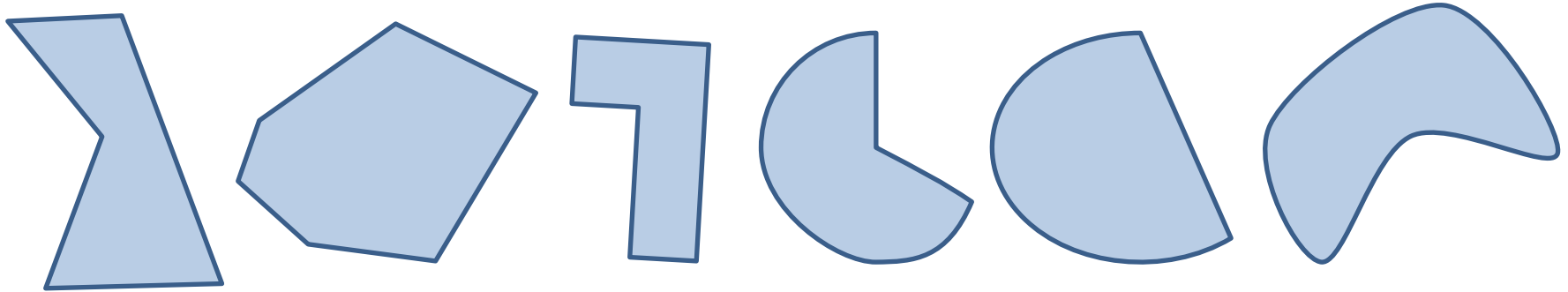
$$C'(0) = \sum_{i=0}^n b_i \cdot B_i'(0) = n(b_1 - b_0)$$

$$C'(1) = \sum_{i=0}^n b_i \cdot B_i'(1) = n(b_n - b_{n-1})$$



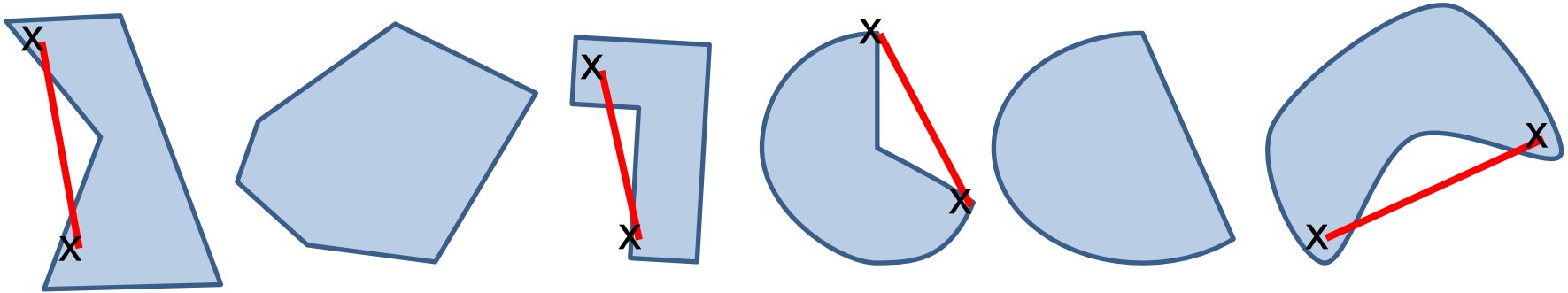
- ▶ Bézier-Kurve liegt in der **konvexen Hülle** der Kontrollpunkte

★ M ist **konvex**: $a, b \in M$ & $0 < t < 1 \Rightarrow (1-t)a + tb \in M$

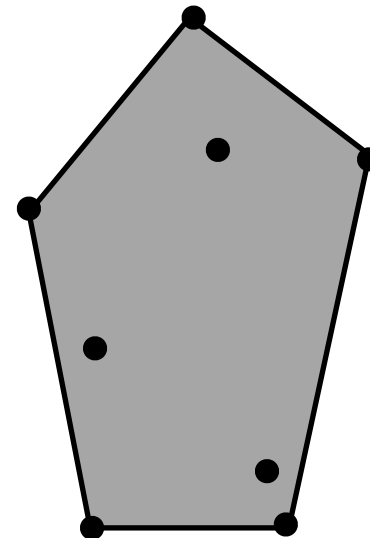


- Bézier-Kurve liegt in der **konvexen Hülle** der Kontrollpunkte

★ M ist **konvex**: $a, b \in M$ & $0 < t < 1 \Rightarrow (1-t)a + tb \in M$

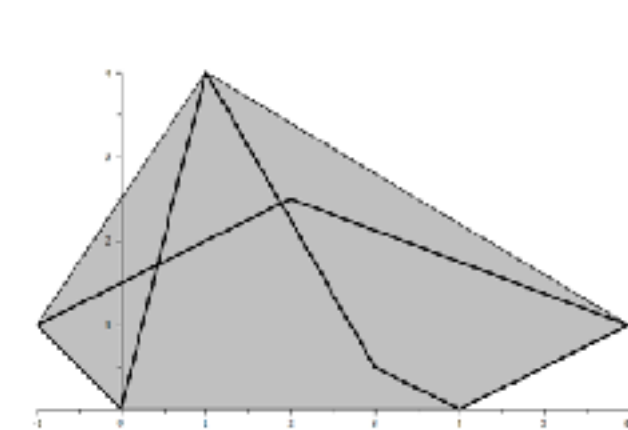
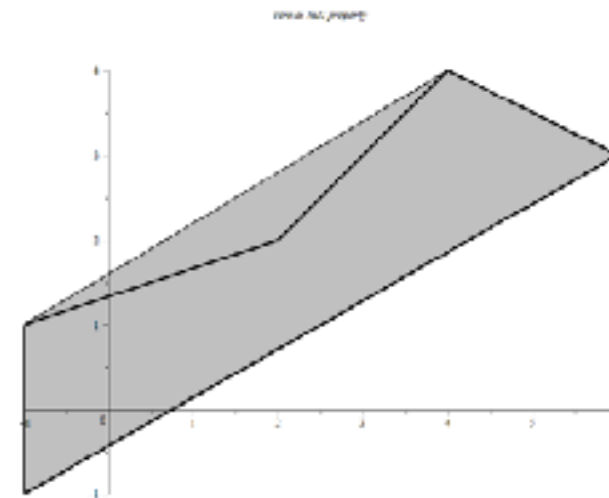
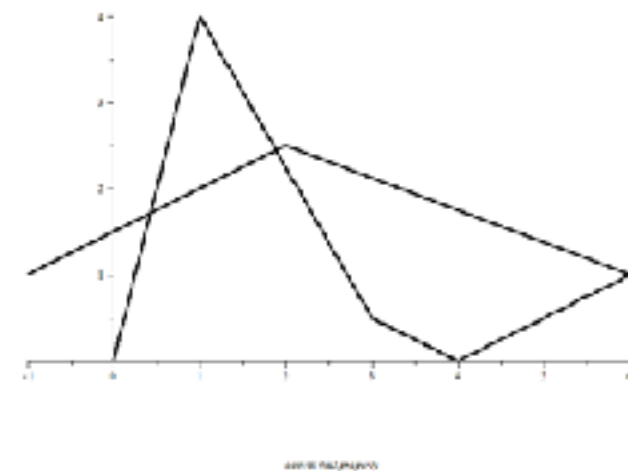
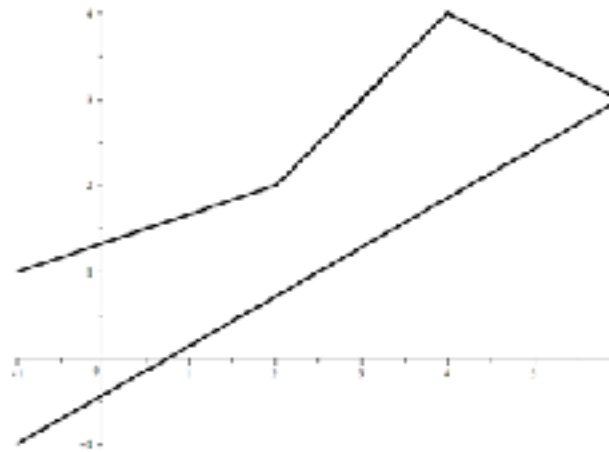


- ★ **Konvexe Hülle** einer Menge A ist die kleinste konvexe Obermenge von A



- Konvexe Hülle:
Bézier-Kurve liegt in **konvexer Hülle** ihrer Kontrollpunkte

- Beispiele



► affine Invarianz

- ★ **Affine Abbildung:** $\Phi(x) = Ax + b$

Translation und lineare Abbildung

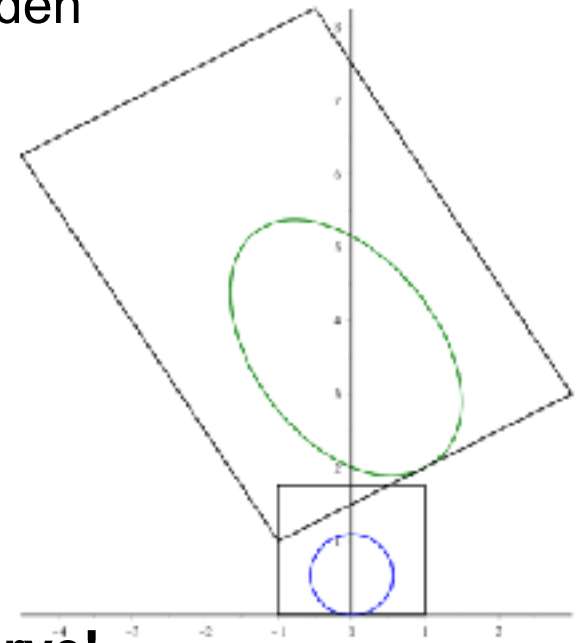
(z.B. Rotation, Spiegelung, Skalierung, Scherung,)

- ★ Soll eine Bézierkurve affin transformiert werden, dann müssen nur die Kontrollpunkte affin transformiert werden

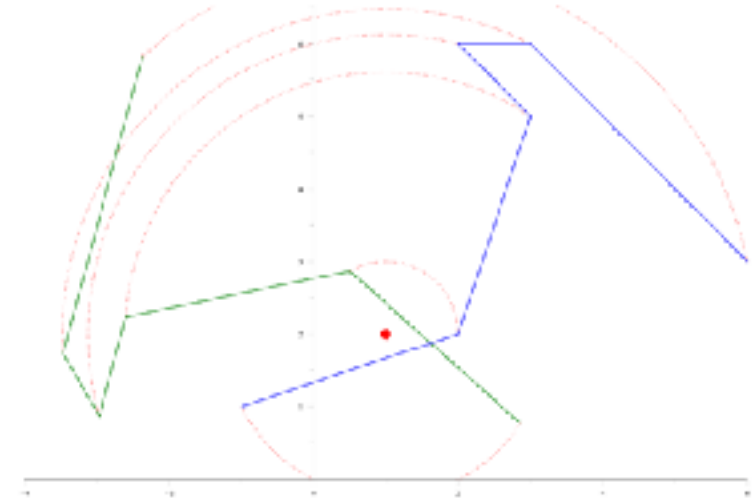
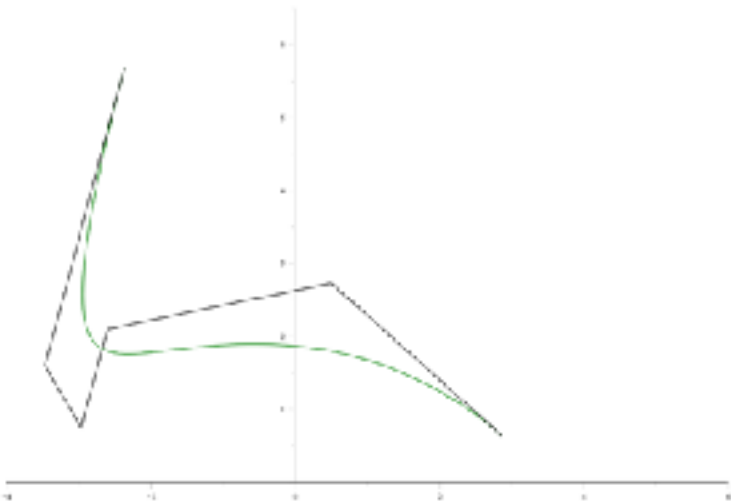
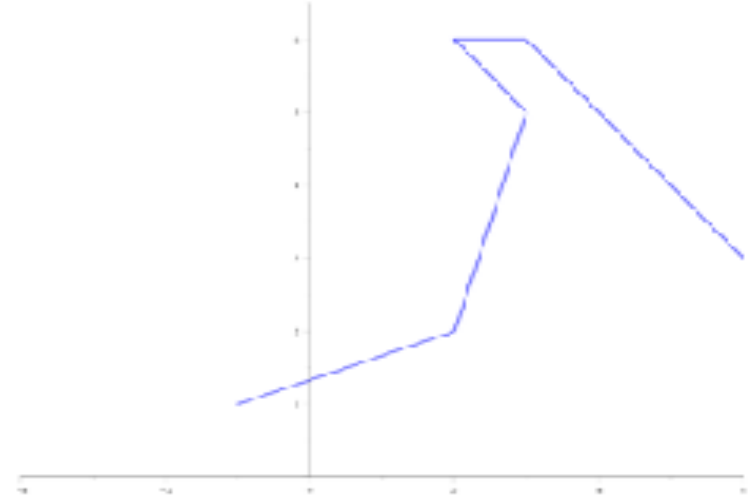
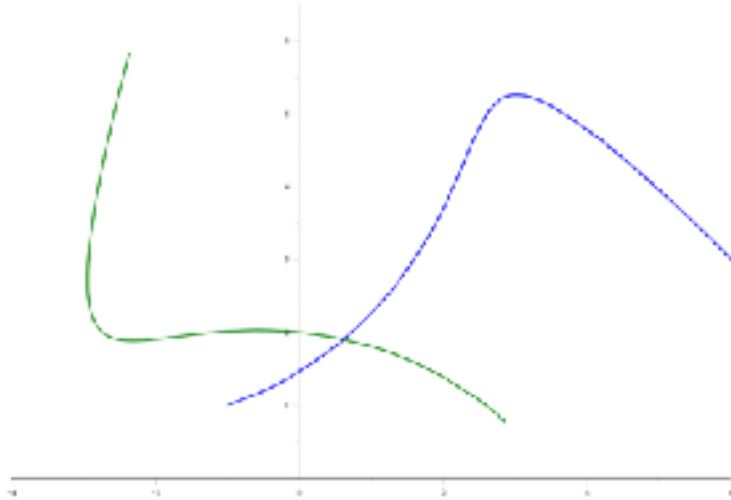
$$C(t) = \sum_{i=0}^n b_i \cdot B_i^n(t) \quad \text{dann}$$

$$\Phi(C(t)) = \sum_{i=0}^n \Phi(b_i) \cdot B_i^n(t)$$

- ★ Andere Sichtweise:
Die transformierten Kontrollpunkte erzeugen die transformierte Bézier-Kurve!



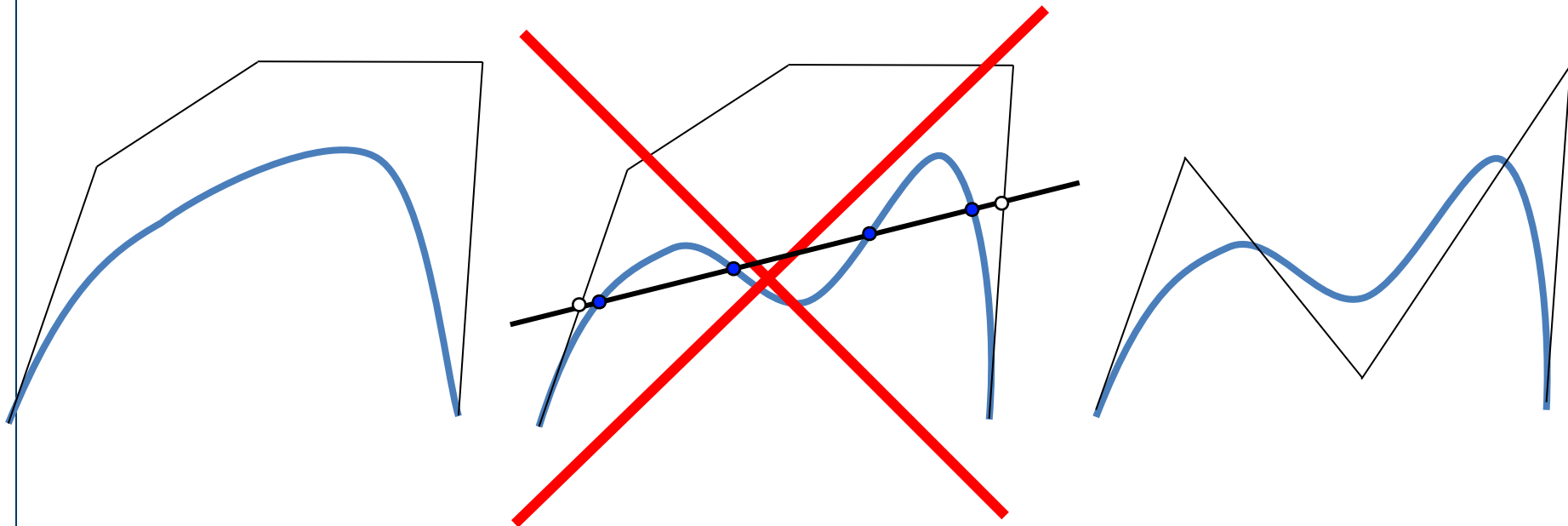
- Affine Invarianz: Beispiel (Drehung um 120°)



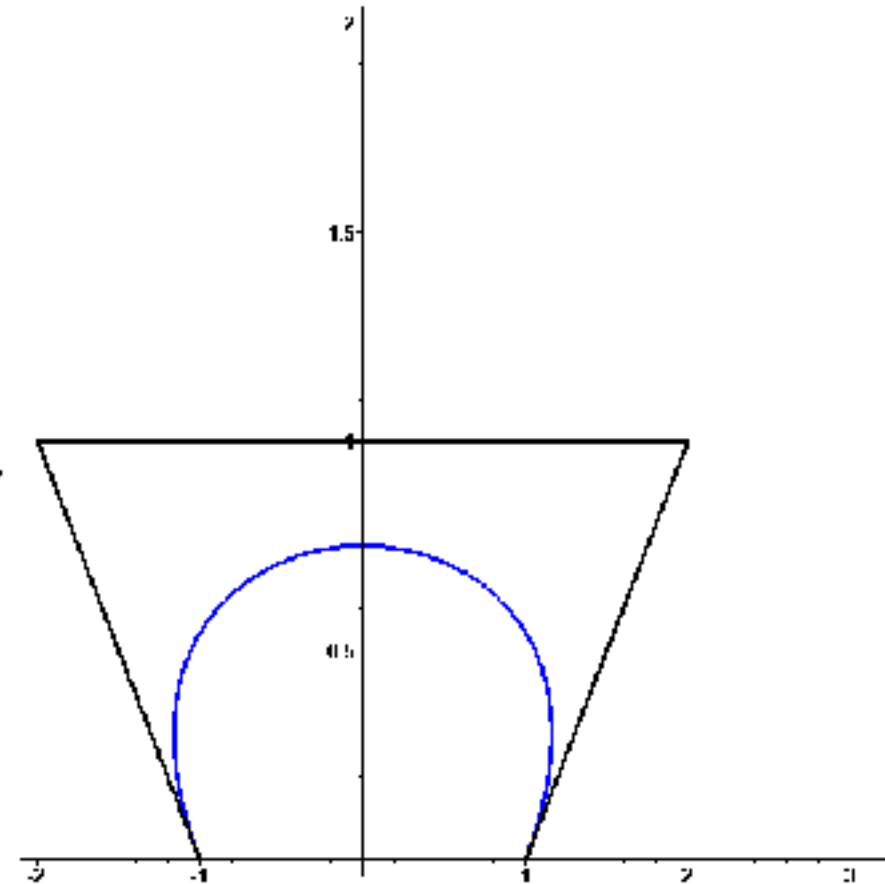
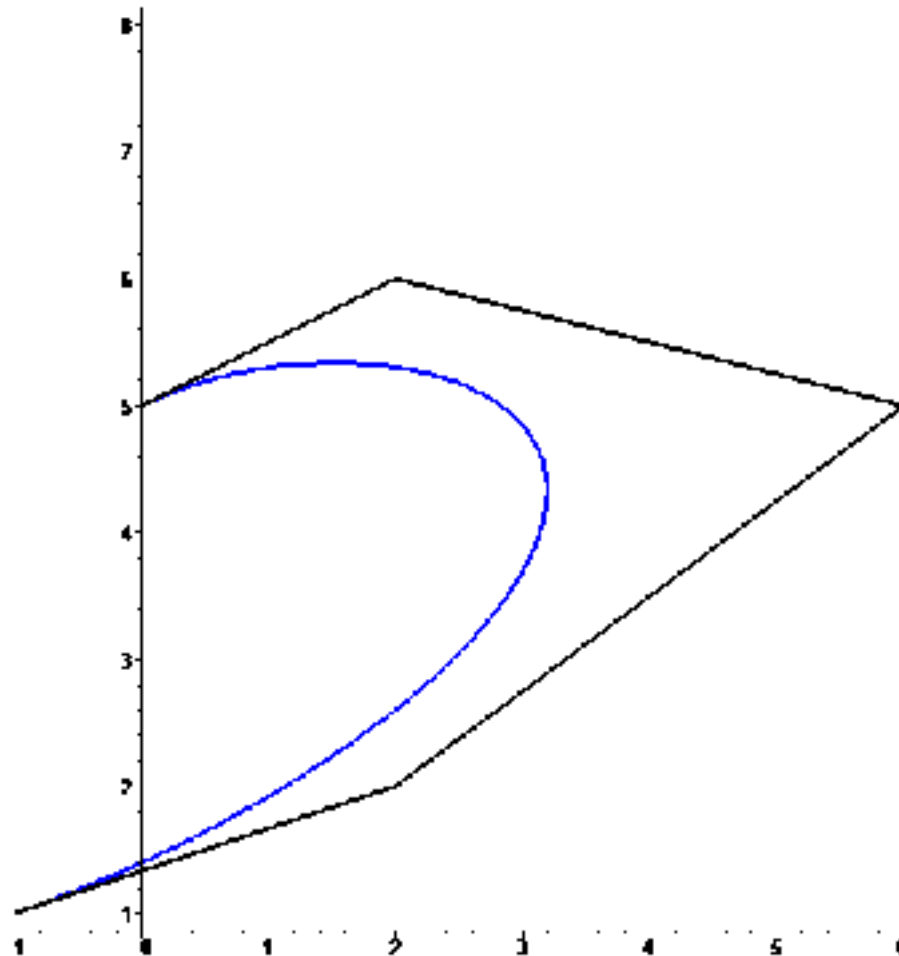
• Variationsreduktion

- ▶ anschaulich: Die Kurve variiert (schwankt) höchstens so stark wie das Kontrollpolygon
- ▶ exakt: Für jede Gerade g gilt:

$\#(\text{Schnittpunkte von } g \text{ mit Kurve}) \leq \#(\text{Schnittpunkte von } g \text{ mit Kontrollpolygon})$



1. Interpolation der Endpunkte
2. In den Endpunkten tangential an das Kontroll-Polygon
3. Bézier-Kurve liegt in der konvexen Hülle
4. affine Invarianz



- (Rekursive) Auswertung der Bernstein Polynome: teuer

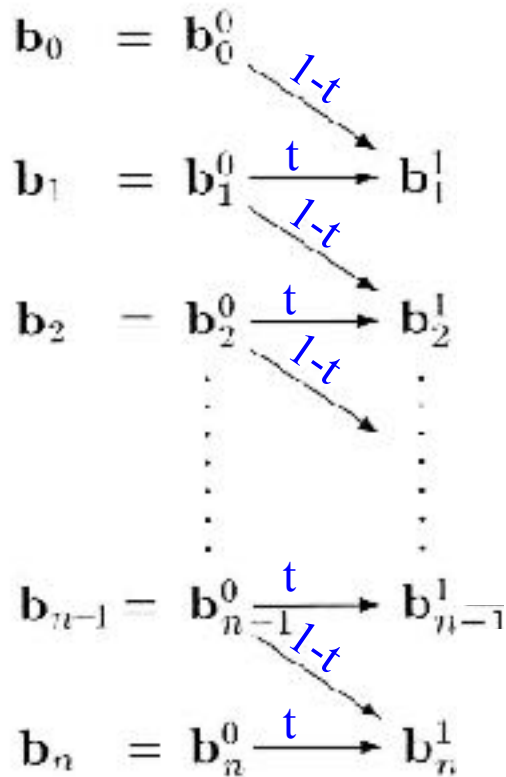
$$B_i^{r+1}(t) = (1-t)B_i^r(t) + tB_{i-1}^r(t)$$

$$B_i^0(t) = \begin{cases} 1 & \text{falls } i = 0 \\ 0 & \text{sonst} \end{cases}$$

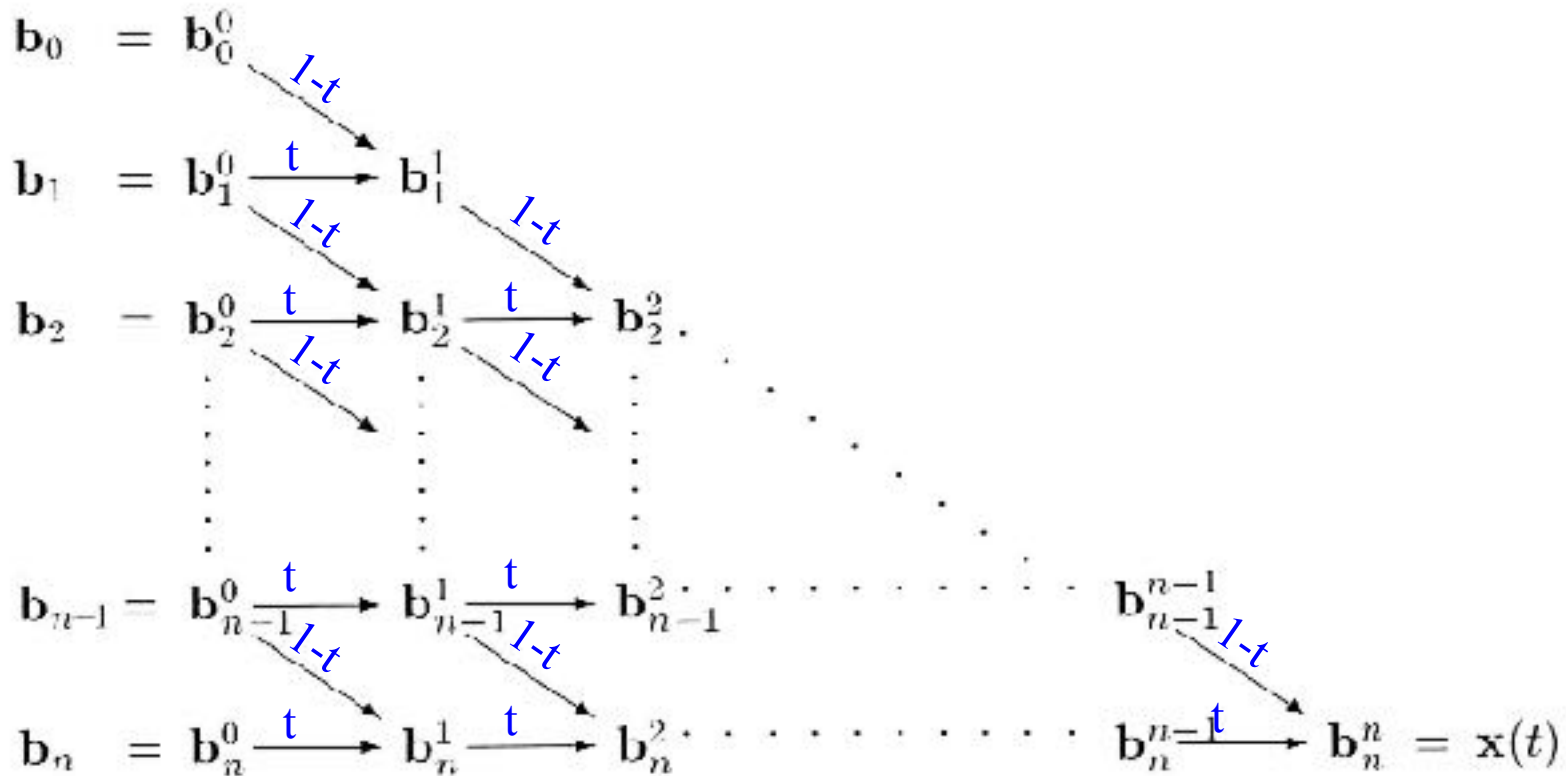
- oder durch **Horner-Bézier**: effizient

$$\begin{aligned} C(t) &= \sum_{i=0}^n b_i B_i^n(t) = \sum_{i=0}^n b_i \binom{n}{i} (1-t)^{n-i} t^i \\ &= (1-t)^n \left(\tilde{b}_0 + \tilde{b}_1 \left(\frac{t}{1-t} \right)^1 + \dots + \tilde{b}_{n-1} \left(\frac{t}{1-t} \right)^{n-1} + \tilde{b}_n \left(\frac{t}{1-t} \right)^n \right) \\ &= t^n \left(\tilde{b}_0 \left(\frac{1-t}{t} \right)^n + \tilde{b}_1 \left(\frac{1-t}{t} \right)^{n-1} + \dots + \tilde{b}_{n-1} \left(\frac{1-t}{t} \right)^1 + \tilde{b}_n \right) \end{aligned}$$

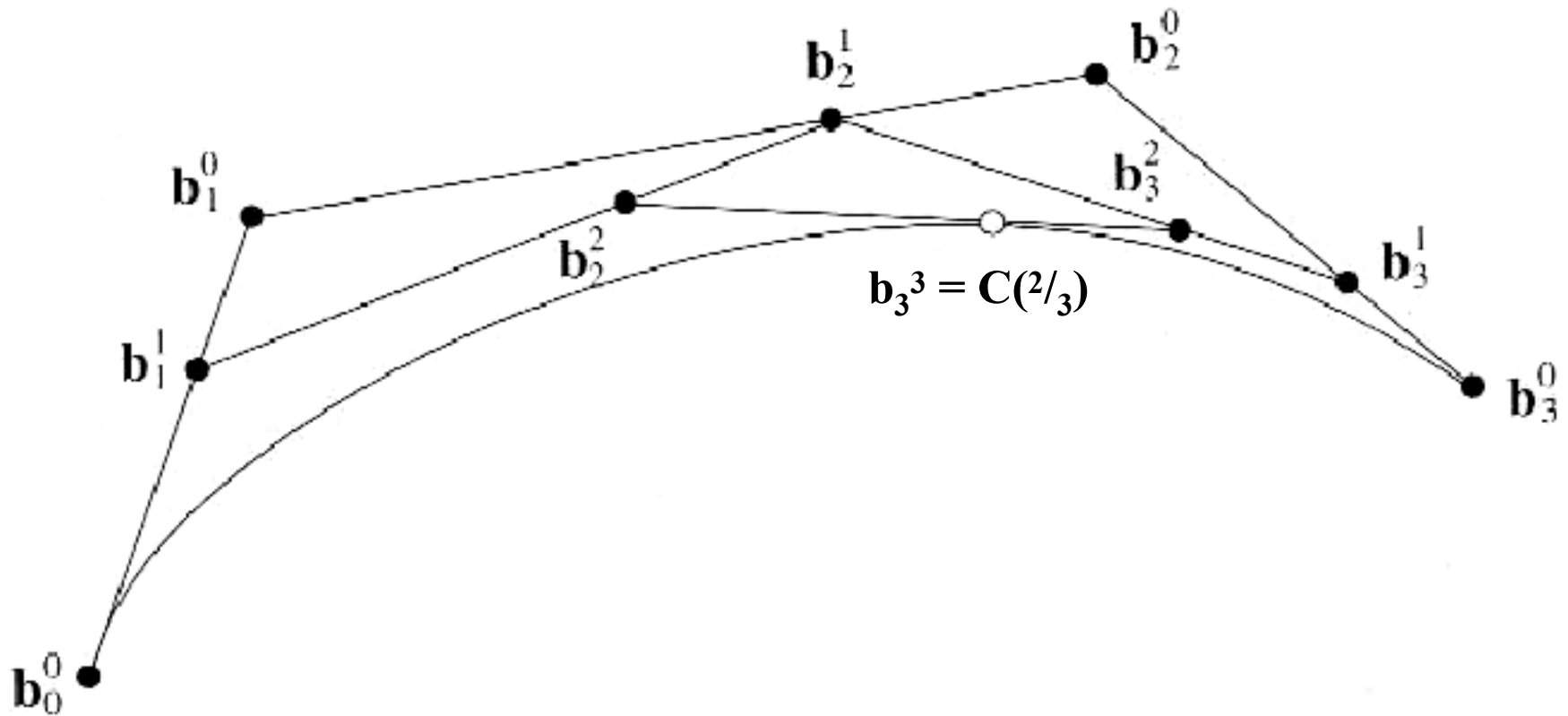
- Oder durch fortgesetzte lineare Interpolation:
der **Algorithmus von de Casteljau** (stabil)



- Oder durch fortgesetzte lineare Interpolation:
der **Algorithmus von de Casteljau** (stabil)



- der **Algorithmus von de Casteljau** (geometrisch)



Algorithmus von *de Casteljau* für $t = 2/3$ und $n = 3$

- Algorithmus von *de Casteljau*: ein Beispiel:
kubische Bézier-Kurve mit folgenden Kontrollpunkten:

$$b_0 = \begin{bmatrix} 0 \\ 9 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 9 \\ 0 \end{bmatrix}, \quad b_3 = \begin{bmatrix} 18 \\ 18 \end{bmatrix}$$

ausgewertet in $t = 1/3$.

- Algorithmus von *de Casteljau*: PseudoCode

```
# Initialisierung
for i = 0..n
     $b_i^0 = b_i;$ 

# deCasteljau Pyramide/Dreieck
for k = 1..n
    for i = k..n
         $b_i^k = (1-t) * b_{i-1}^{k-1} + t * b_i^{k-1}$ 

return  $b_n^n$  # Kurvenpunkt C(t)
```

- Algorithmus von *de Casteljau* mit Überschreiben der Werte (speicher-effizienter):

```
# deCasteljau Pyramide
for k = 1..n
  for i = 0..n-k
     $b_i = (1-t) * b_i + t * b_{i+1}$ 

return(  $b_0$  );    # Kurvenpunkt C(t)
```


- Der Benutzer manipuliert die Kontrollpunkte (interaktiv)
 - Verschieben von Punkten
 - Einfügen/Löschen von Kontrollpunkten (Erhöhung/Verringerung des Grades)
 - Beachte: nur Anfangs- und Endpunkt liegen auf der Kurve.

- Detail erfordert uU viele Kontrollpunkte
 - Mit einer Bézier-Kurve → hoher Polynomgrad (instabil)
 - Besser: mehrere Bézier-Kurven (mit jeweils moderatem Grad) aneinander fügen (→ **Bézier-Splines**)
 - wichtig dabei: ein stetiger, oft ein glatter Übergang zwischen den Segmenten
 - Stetigkeit wird erreicht durch
„letzter Kontrollpunkt = erster Kontrollpunkt“

- Glatter Übergang zwischen benachbarten Bézier-Kurven

$$C(t) = \sum_i \mathbf{b}_i B_i^n(t) \text{ und } D(t) = \sum_i \mathbf{c}_i B_i^n(t),$$

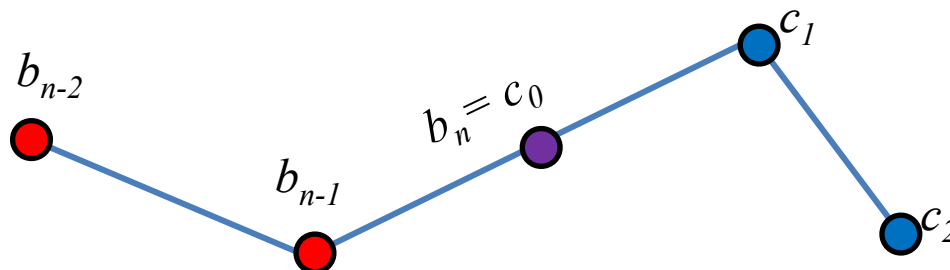
- ▶ Stetigkeit falls $\mathbf{b}_n = \mathbf{c}_0$ (folgt aus Endpunktinterpolation!)

- ▶ Die Tangenten im Punkt $\mathbf{b}_n = \mathbf{c}_0$ sind gegeben durch

$$C'(1) = n(\mathbf{b}_n - \mathbf{b}_{n-1}) \quad (\text{folgt aus Tangentenbedingung})$$

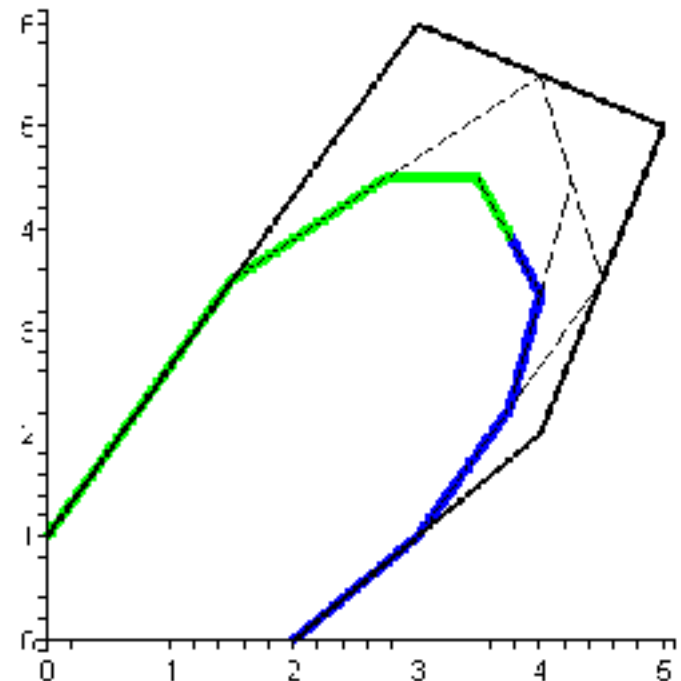
$$D'(0) = n(\mathbf{c}_1 - \mathbf{c}_0)$$

- ▶ \rightarrow ein glatter Übergang falls die drei Kontrollpunkte $\{\mathbf{b}_{n-1}, \mathbf{b}_n = \mathbf{c}_0, \mathbf{c}_1\}$ an der Nahtstelle kollinear sein.

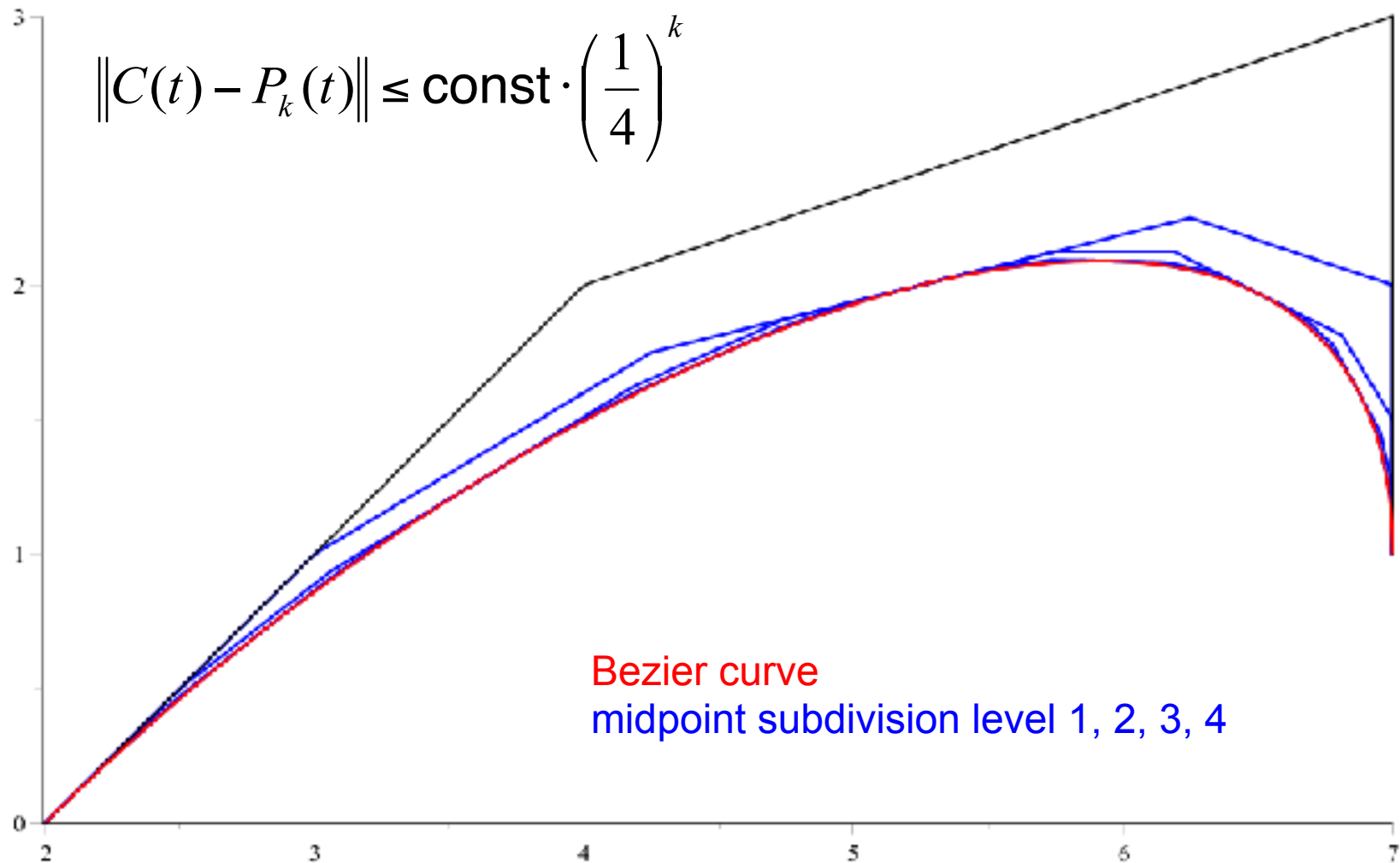


- Noch glatterer Übergang:
 - ▶ Die Glattheit im Sinne der Parametrisierung von $C(t)$ und $D(t)$ erfordert zusätzliche Bedingungen an die Kontrollpunkte.
 C^1 -Stetigkeit siehe oben.
 - ▶ Durch zusätzliche Bedingungen an die „Kontrollpunkte am Ende bzw. Anfang“ können noch stärkere Glattheitseigenschaften erzielt werden:
 C^2 -Stetigkeit: Dann sind auch die Krümmungen stetig.
 - ▶ Für zwei Polynomsegmente n -ten Grades, kann maximal C^{n-1} -Stetigkeit erreicht werden (sonst gleiches Polynom)
 - ▶ Dies führt letztendlich auf Splinefunktionen
→ **B-Spline-Kurven** (Vorlesung *Geometrische Modellierung*)

- Fakt:
Graphik-Hardware kann Linien sehr effizient zeichnen:
- Ziel: Approximiere Bézier-Kurve durch Polygonzug
- Methode: **Subdivision**
- Es gilt:
Die Kanten des *deCasteljau*-Dreiecks sind die Kontrollpolygone der Teilkurven
 $C|_{[0,t]}$ und $C|_{[t,1]}$
- t = Mittelpunkt und rekursiv weiter machen
 → **midpoint Subdivision**

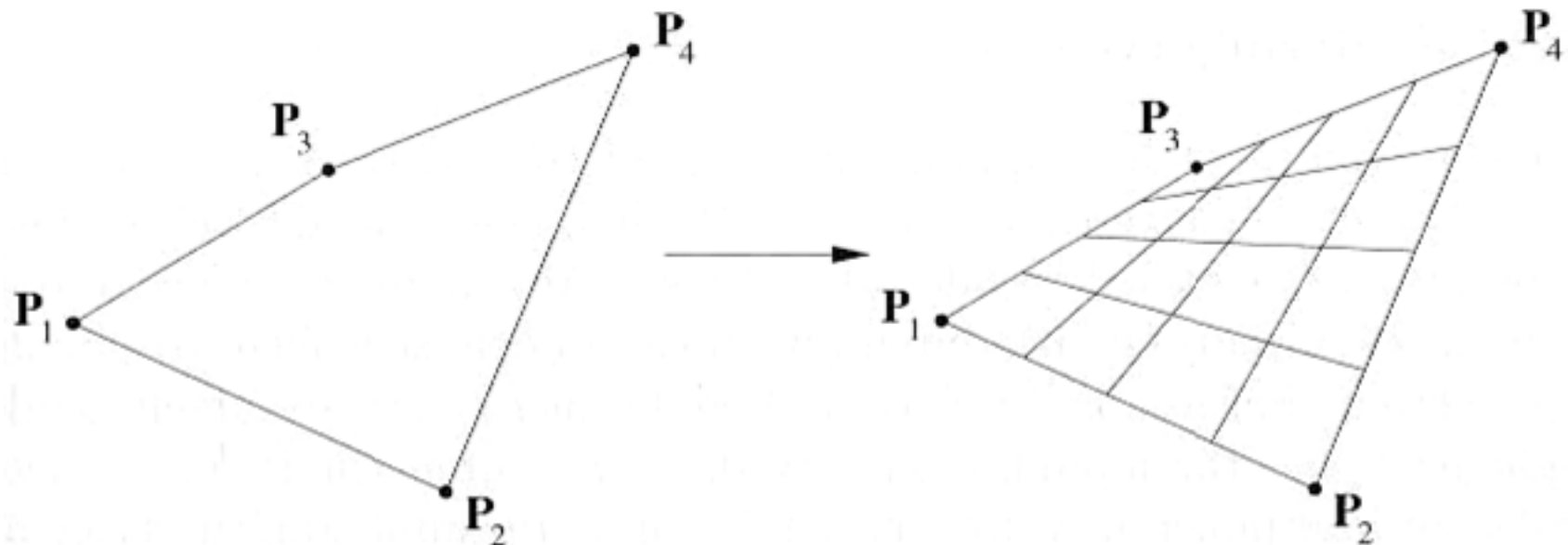


- Rekursive midpoint-subdivision konvergiert sehr schnell



- Tensorproduktansatz
- transfinite Interpolation
 - ▶ Coons-Patch

- Überstrichene Fläche bei der Bewegung von P_1 und P_2 auf geradem Weg nach P_3 bzw. P_4 , unter Mitführung der jeweiligen Verbindungsstrecke.

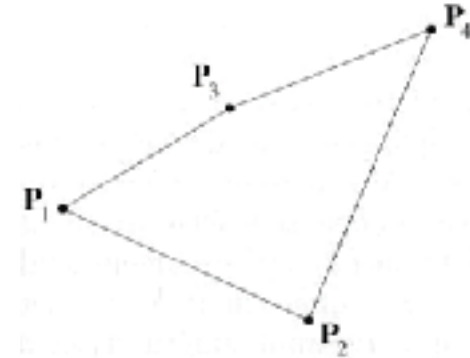


- Bilineare Interpolant kann implementiert werden als
 - zuerst lineare Interpolation in t , dann lineare Interpolation in s
 - zuerst lineare Interpolation in s , dann lineare Interpolation in t
 - oder direkt : $(1-s)(1-t) \cdot P_1 + s(1-t) \cdot P_2 + (1-s)t \cdot P_3 + st \cdot P_4$
- Anschaulich:
 - **Kurve**: einparametrische Schar von Punkten
 - **Fläche**: einparametrische Schar von Kurven
- **Tensorproduktansatz**:
 Für jeden Parameter t ist $F(t,s)$ lineare Kurve (in s).
 Die Koeffizienten dieser Kurve hängen (linear) von t ab

 oder umgekehrt!

- Vier Eckpunkte

$$P_{00} = P_1, P_{10} = P_2, P_{01} = P_3, P_{11} = P_4$$

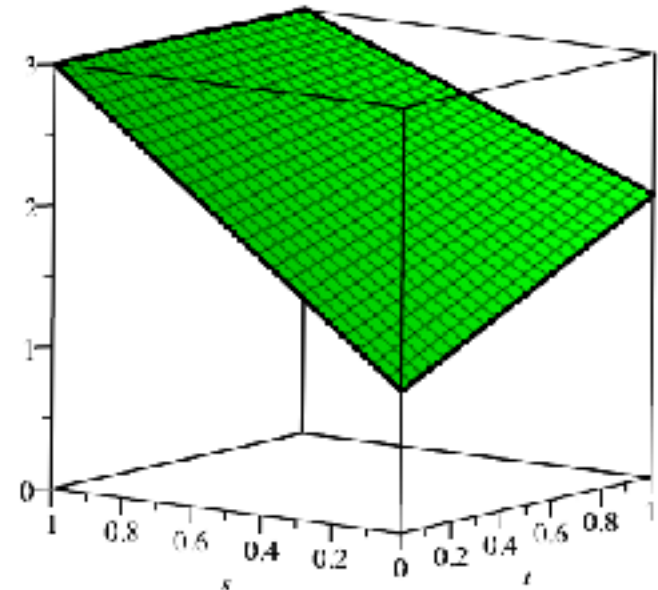


- der Interpolant ist

$$\begin{aligned} F(s, t) &= (1-t)((1-s)P_{00} + sP_{10}) + t((1-s)P_{01} + sP_{11}) \\ &= (1-s)(1-t)P_{00} + s(1-t)P_{10} + (1-s)tP_{01} + stP_{11} \\ &= \sum_{i=0}^1 \sum_{j=0}^1 P_{ij} B_i^1(s) B_j^1(t) \end{aligned}$$

- Konkretes Beispiel:

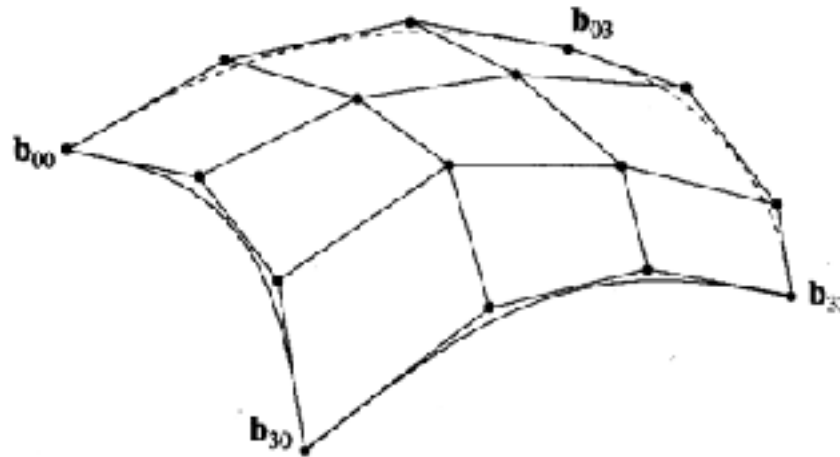
$$P_{00} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, P_{10} = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}, P_{11} = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix}, P_{01} = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}$$



- Allgemeine **Tensor-Produkt Bézier-Fläche** vom Grad n und m :

$$F(s, t) = \sum_{k=0}^m \sum_{i=0}^n \mathbf{b}_{ik} B_i^n(s) B_k^m(t) \quad \text{für } s, t \in [0, 1] \quad (\mathbf{b}_{ik} \in \mathbb{R}^3)$$

- Die Kontrollpunkte spannen jetzt ein **Kontrollnetz** auf
- Beispiel für Grad 3 ($m = n = 3$) bikubisch



- Beispiel: Utah-Teapot (*Martin Newell, 1974*)
 - besteht aus 28 bikubischen Bézier-Flächen



- http://www.sjbaker.org/wiki/index.php?title=The_History_of_The_Teapot

- Allgemeine Tensor-Produkt Bézier-Fläche vom Grad n und m :

$$F(s, t) = \sum_{k=0}^m \sum_{i=0}^n \mathbf{b}_{ik} B_i^n(s) B_k^m(t) = \sum_{i=0}^n \left(\sum_{k=0}^m \mathbf{b}_{ik} B_k^m(t) \right) B_i^n(s)$$

- Für jeden Parameter t ist $s \rightarrow F(s, t)$ eine Bézier-Kurve (vom Grad n)
(d.h. einparametrische Schar von Bézier-Kurven)

- Die Kontrollpunkte dieser Kurve sind hängen von t ab:

$$\mathbf{d}_i(t) = \sum_k \mathbf{b}_{ik} B_k^m(t)$$

- Kurven mit $s = \text{const}$ sind Bézier-Kurven in t :

$$F(s, t) = \sum_k \mathbf{c}_k(s) B_k^m(t) \quad \text{mit} \quad \mathbf{c}_k(s) = \sum_i \mathbf{b}_{ik} B_i^n(s)$$

- Allgemeine Tensor-Produkt Bézier-Fläche vom Grad n und m :

$$F(s, t) = \sum_{k=0}^m \sum_{i=0}^n \mathbf{b}_{ik} B_i^n(s) B_k^m(t) \quad \text{für } s, t \in [0, 1] \quad (\mathbf{b}_{ik} \in \mathbb{R}^3)$$

- **Formeigenschaften** der Bézier-Kurven übertragen sich auf Bézier-Flächen

1. Eckpunktinterpolation
2. Tangentialebenen in den 4 Eckpunkten
3. Affine Invarianz
4. Konvexe Hülle
5. Jedoch **nicht**: Variationsreduktion

Die 4 Randkurven einer Bézier-Fläche sind Bézier-Kurven.

Die jeweiligen Kontrollpunkte liegen am Rand des Kontrollnetzes.

- Algorithmen für Tensorprodukt-Flächen
- Allgemeines Vorgehen: Zwei Schritte:
 - ▶ Kurven-Algorithmus in t -Richtung
 - ▶ Kurven-Algorithmus in s -Richtung (oder umgekehrt)
- Beispiel: Auswertung mit deCasteljau
 - ★ Auf **jede** Spalte von Kontrollpunkten den „Kurven-deCasteljau“ in t -Richtung
 - ★ Auf die erhaltenen Punkte **einen** „Kurven-deCasteljau“ in s -Richtung
- ▶ Weiteres Beispiel: *midpoint subdivision*
 - ★ Auf **jede** Spalte von Kontrollpunkten midpoint subdivision
 - ★ Auf **jede** Zeile des erhaltenen Arrays midpoint subdivision

Beispiel: Tensorprodukt-deCasteljau

- Allgemeine Tensor-Produkt Bézier-Fläche vom Grad n und m :

$$F(s, t) = \sum_{k=0}^m \sum_{i=0}^n \mathbf{b}_{ik} B_i^n(s) B_k^m(t) \quad \text{für } s, t \in [0, 1] \quad (\mathbf{b}_{ik} \in \mathbb{R}^3)$$

- Tensor-Produkt Ansatz: 1D Version in „beide Richtungen“
 - Zuerst eindimensional in erster Richtung mit de Casteljau
Dann eindimensional in zweiter Richtung mit de Casteljau

- Andere Sichtweise

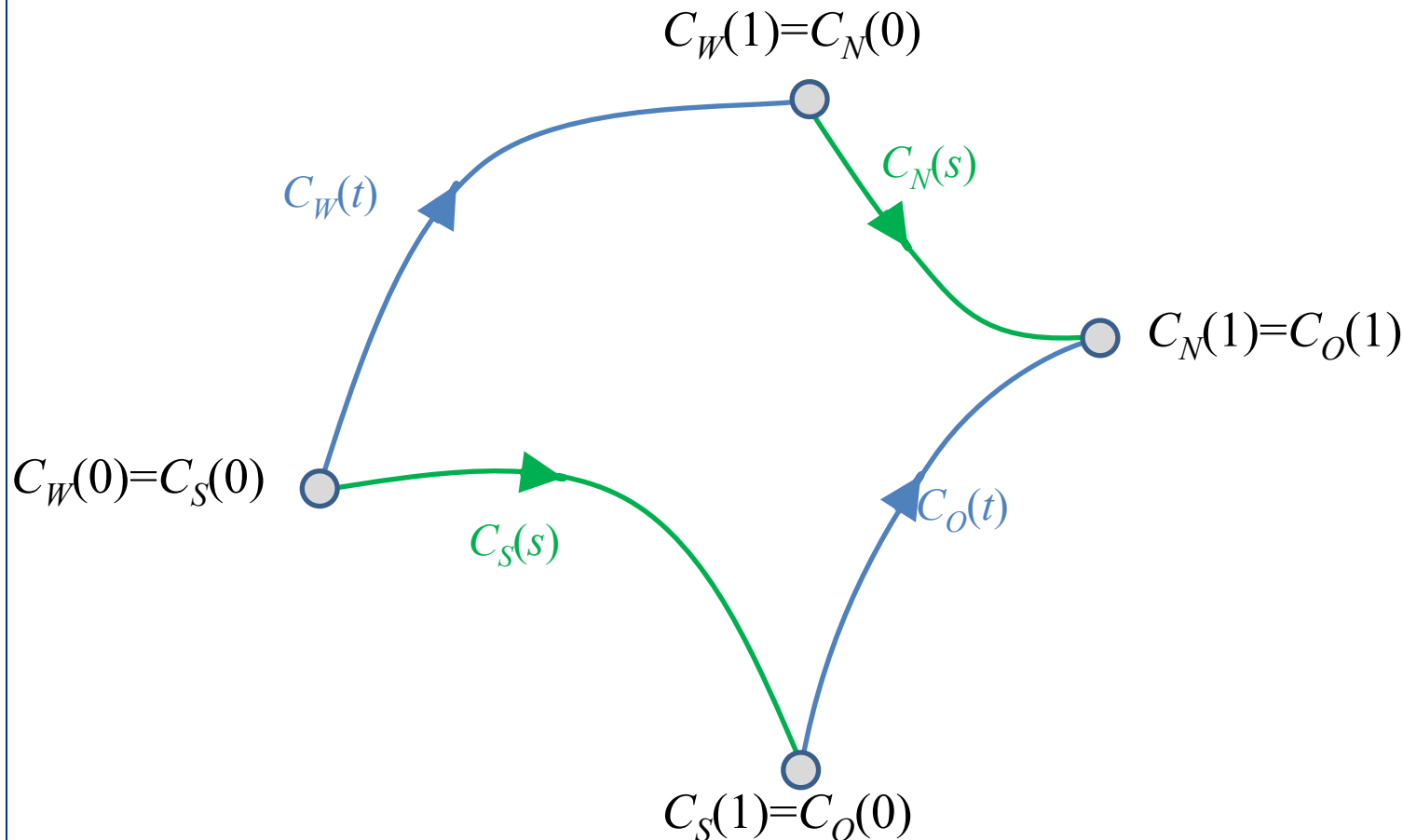
- Zuerst die Auswertung mit *deCasteljau* für jede Zeile (an der Stelle s).

$$\mathbf{b}_k(s) = \sum_{i=0}^n \mathbf{b}_{ik} B_i^n(s)$$

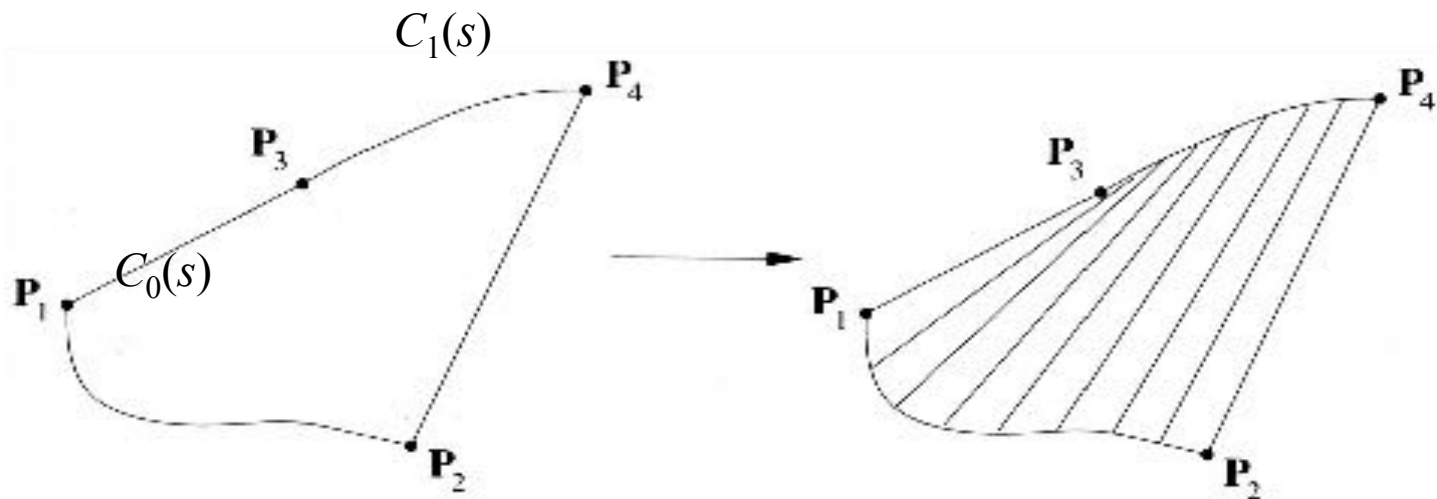
- Für die so erhaltenen Werte die Auswertung nach *deCasteljau* (an der Stelle t).

$$F(s, t) = \sum_{k=0}^m \mathbf{b}_k(s) B_k^m(t)$$

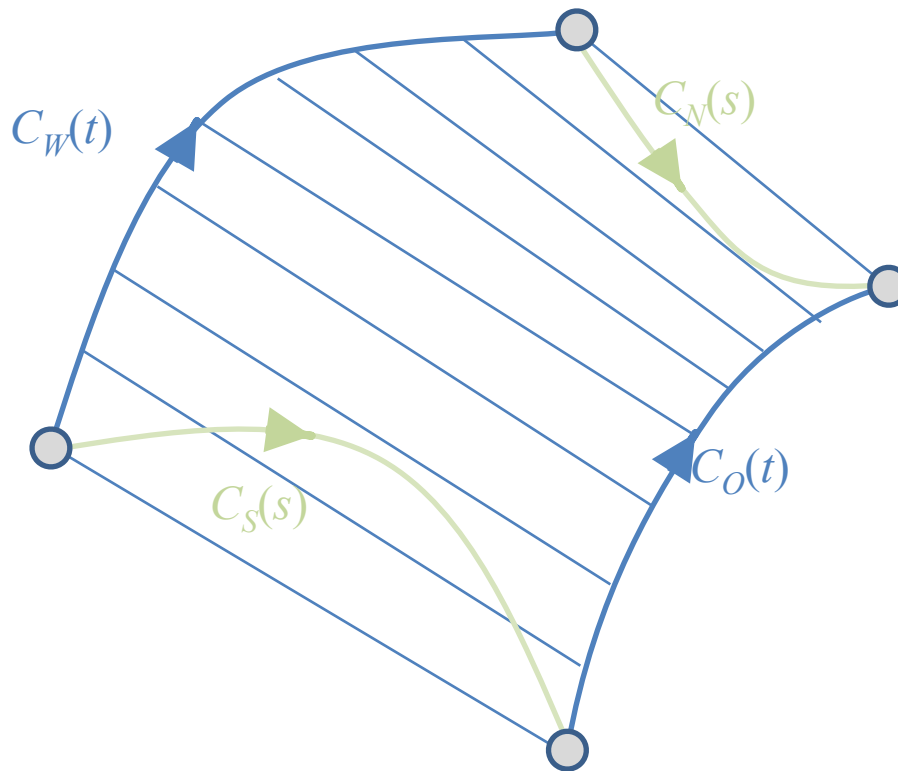
- Ziel: Verallgemeinerung der bilinearen Interpolation:
Vier (kompatible) Randkurven C_S , C_O , C_N , C_W ins Innere fortsetzen (verblenden)



- Verallgemeinerung der bilinearen Interpolation:
 - dabei waren die Randkurven lineare Strecken
 - Verbinden korrespondierender Punkte durch lineare Strecke
- Transfinite Interpolation in einer Richtung (Regelfläche):
 - Kurven $C_0(s)$ von P_1 nach P_2 und $C_1(s)$ von P_3 nach P_4
 - Dann definieren wir die **Regelfläche** $F_s(s,t) = (1-t) C_0(s) + t C_1(s)$
 - Linear in einer Richtung, „kurvig“ in der anderen



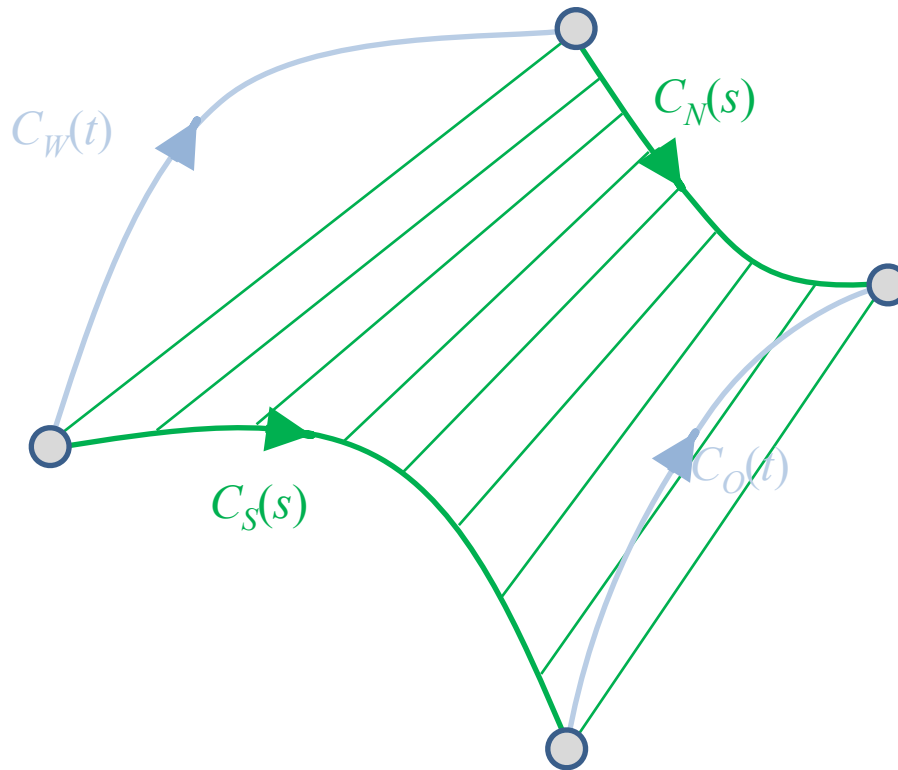
- Transfinite Interpolation in einer Richtung mit Regelfläche:
 - ▶ gegenüber liegendes Kurvenpaar $C_W(t)$ und $C_O(t)$ linear verbinden
 - ▶ Das ergibt die **Regelfläche** $F_t(s,t) = (1 - s) C_W(t) + s C_O(t)$



- Transfinite Interpolation in die andere Richtung :
 - gegenüber liegendes Kurvenpaar $C_S(s)$ und $C_N(s)$ linear verbinden

- Das ergibt die **Regelfläche**

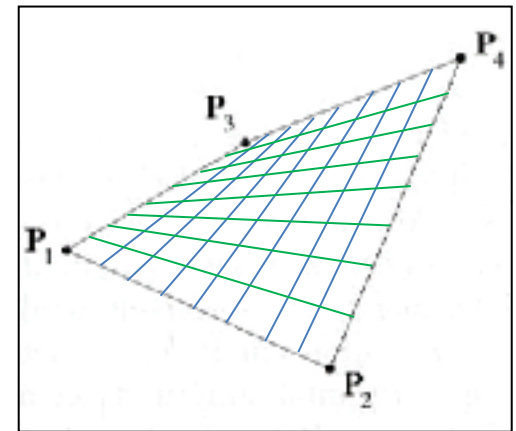
$$F_s(s, t) = (1 - t) C_S(s) + t C_N(s)$$



- Wie kann man nun die beiden jeweils „einseitigen“ transfiniten Interpolanten kombinieren?
- Die Summe $F_t(s,t) + F_s(s,t)$ enthält am Rand gerade die linearen Anteile „zu viel“.
- Lösung: das **Coons-Patch** : Subtrahiere von der Summe den **bilinearen Interpolanten der Eckpunkte**

$$P_1 = C_W(0) = C_S(0), \quad P_2 = C_O(0) = C_S(1),$$

$$P_3 = C_N(0) = C_W(1), \quad P_4 = C_N(1) = C_O(1),$$



$$F_{st}(s,t) = (1-s)(1-t) \cdot P_1 + s(1-t) \cdot P_2 + (1-s)t \cdot P_3 + st \cdot P_4$$

- Das „**Coons-Patch**“ (manchmal auch als transfinite Interpolation bezeichnet):

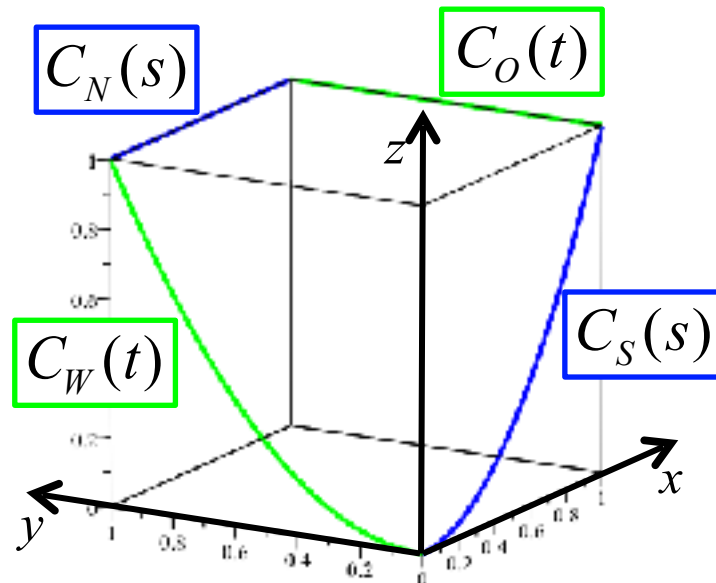
$$F(s, t) = F_t(s, t) + F_s(s, t) - F_{st}(s, t)$$

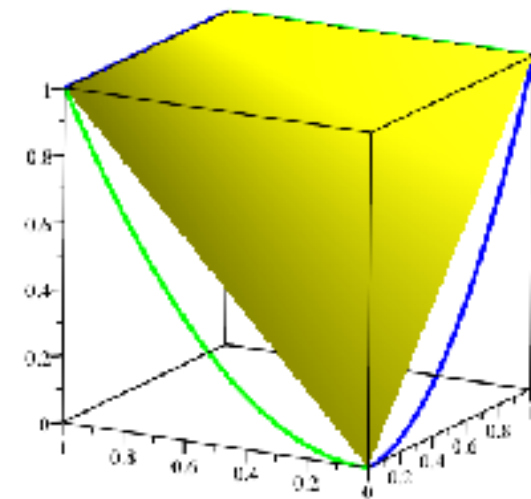
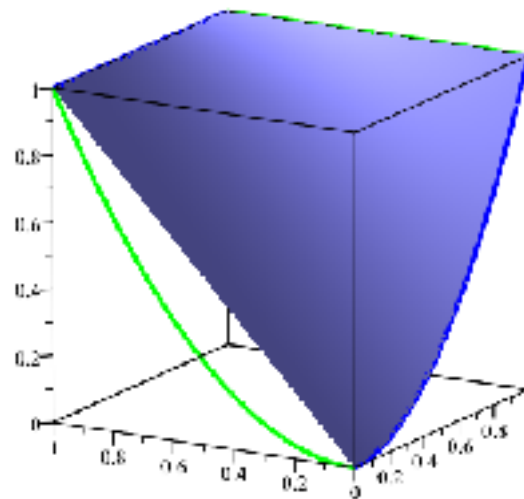
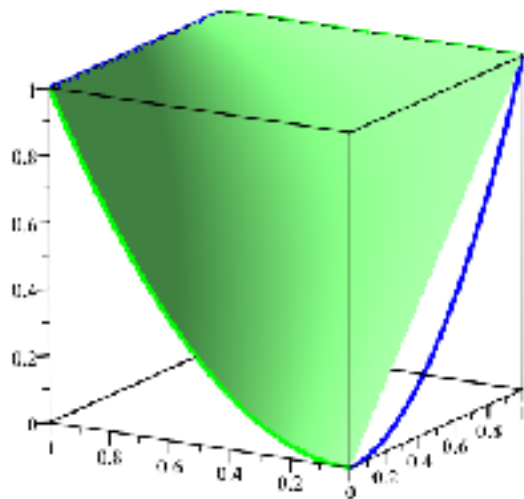
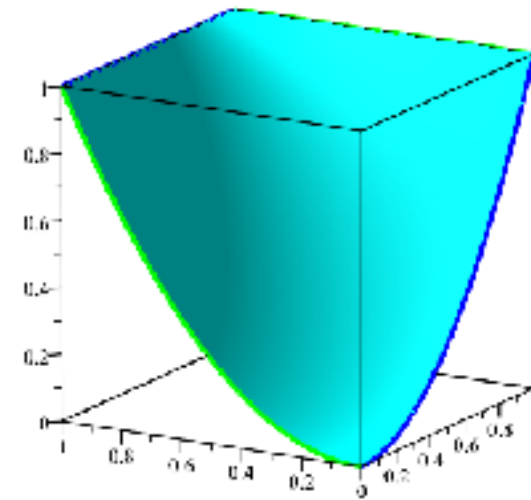
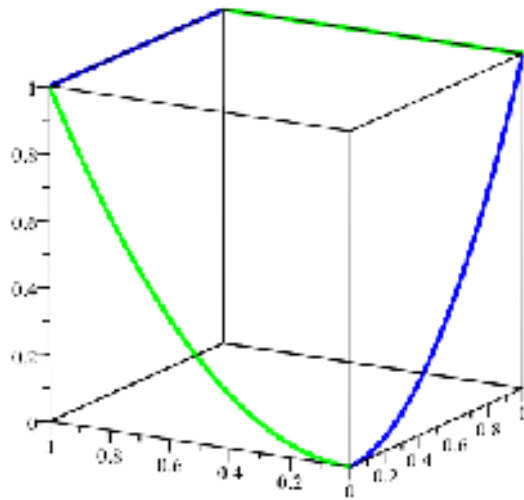
wobei $F_{st}(s, t)$ der bilineare Interpolant zu den vier Randpunkten ist

- Das Coons-Patch stimmt am Rand exakt mit den vier Randkurven überein
- Das Flächeninnere wird dabei mit Hilfe mehrfacher linearer Interpolation definiert
- Dies kann noch weiter verallgemeinert werden:
z.B. kubisches Coons-Patch

- Beispiel:
Bestimme das Coons-Patch zu folgenden Randkurven:

$$C_S(s) = \begin{bmatrix} s \\ 0 \\ s^2 \end{bmatrix}, \quad C_O(t) = \begin{bmatrix} 1 \\ t \\ 1 \end{bmatrix}, \quad C_N(s) = \begin{bmatrix} s \\ 1 \\ 1 \end{bmatrix}, \quad C_W(t) = \begin{bmatrix} 0 \\ t \\ t^2 \end{bmatrix},$$





- Beispiel (cont'd) $C_S(s) = \begin{bmatrix} s \\ 0 \\ s^2 \end{bmatrix}, \quad C_O(t) = \begin{bmatrix} 1 \\ t \\ 1 \end{bmatrix}, \quad C_N(s) = \begin{bmatrix} s \\ 1 \\ 1 \end{bmatrix}, \quad C_W(t) = \begin{bmatrix} 0 \\ t \\ t^2 \end{bmatrix},$

$$F_s(s, t) = (1-t) \cdot C_S(s) + t \cdot C_N(s) = \begin{bmatrix} (1-t)s \\ 0 \\ (1-t)s^2 \end{bmatrix} + \begin{bmatrix} ts \\ t \\ t \end{bmatrix} = \begin{bmatrix} s \\ t \\ t + s^2 - ts^2 \end{bmatrix}$$

$$F_t(s, t) = (1-s) \cdot C_W(t) + s \cdot C_O(t) = \begin{bmatrix} 0 \\ (1-s)t \\ (1-s)t^2 \end{bmatrix} + \begin{bmatrix} s \\ st \\ s \end{bmatrix} = \begin{bmatrix} s \\ t \\ s + t^2 - st^2 \end{bmatrix}$$

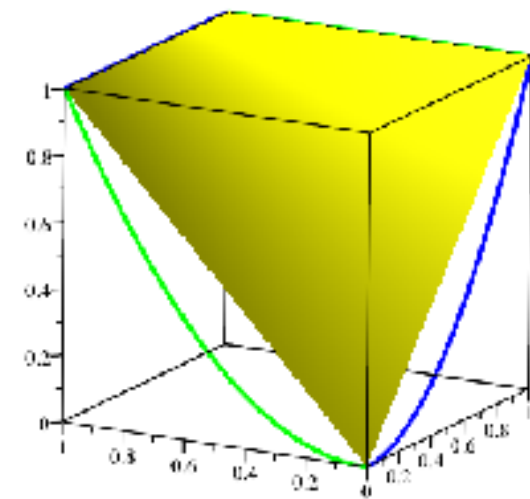
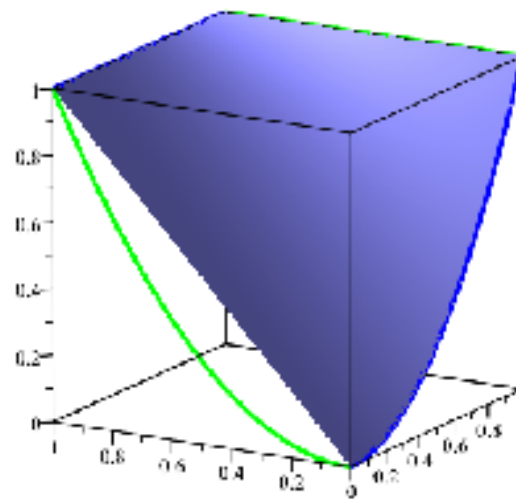
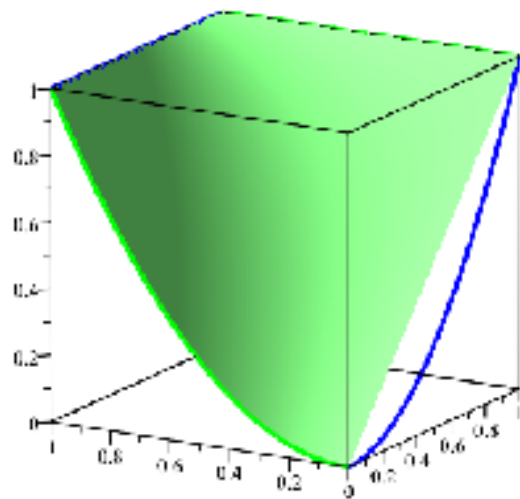
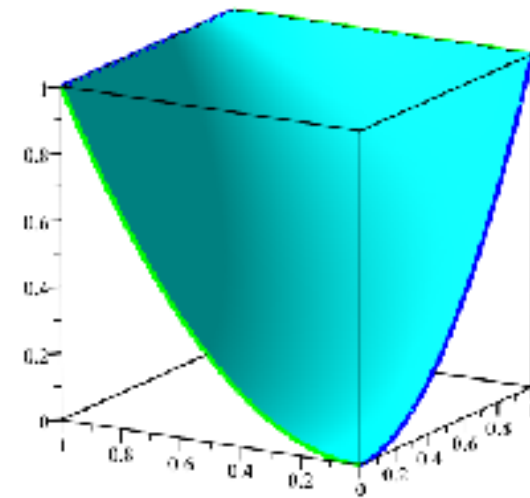
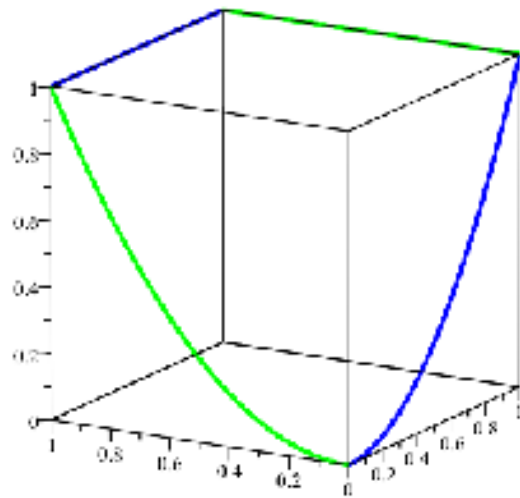
$$\begin{aligned} F_{st}(s, t) &= (1-s)(1-t)C_W(0) + s(1-t)C_O(0) + (1-s)tC_W(1) + stC_O(1) \\ &= \begin{bmatrix} 0 + s(1-t) + 0 + st \\ 0 + 0 + (1-s)t + st \\ 0 + s(1-t) + (1-s)t + st \end{bmatrix} = \begin{bmatrix} s \\ t \\ s + t - st \end{bmatrix} \end{aligned}$$

- Beispiel (cont'd) $C_S(s) = \begin{bmatrix} s \\ 0 \\ s^2 \end{bmatrix}, \quad C_O(t) = \begin{bmatrix} 1 \\ t \\ 1 \end{bmatrix}, \quad C_N(s) = \begin{bmatrix} s \\ 1 \\ 1 \end{bmatrix}, \quad C_W(t) = \begin{bmatrix} 0 \\ t \\ t^2 \end{bmatrix},$

$$F_s(s, t) = \begin{bmatrix} s \\ t \\ t + s^2 - ts^2 \end{bmatrix} \quad F_t(s, t) = \begin{bmatrix} s \\ t \\ s + t^2 - st^2 \end{bmatrix} \quad F_{st}(s, t) = \begin{bmatrix} s \\ t \\ s + t - st \end{bmatrix}$$

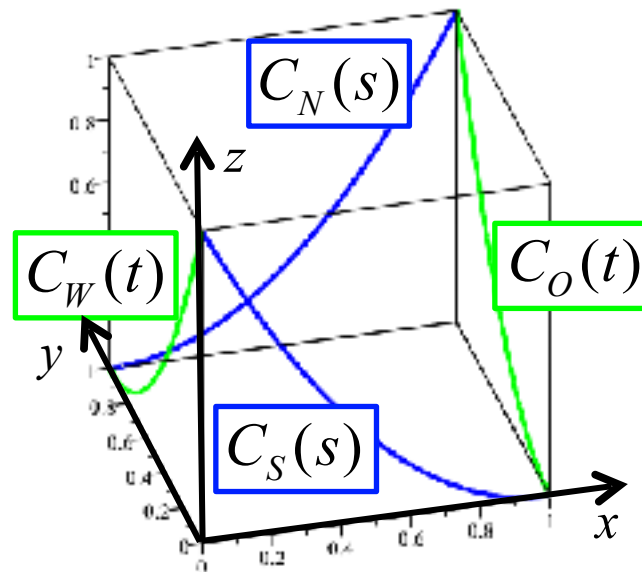
$$CP(s, t) = F_s(s, t) + F_t(s, t) - F_{st}(s, t)$$

$$= \begin{bmatrix} s \\ t \\ t + s^2 - ts^2 \end{bmatrix} + \begin{bmatrix} s \\ t \\ s + t^2 - st^2 \end{bmatrix} - \begin{bmatrix} s \\ t \\ s + t - st \end{bmatrix} = \begin{bmatrix} s \\ t \\ s^2 + t^2 - (s + t - 1)st \end{bmatrix}$$



- Beispiel:
Bestimme das Coons-Patch zu folgenden Randkurven:

$$C_S(s) = \begin{bmatrix} s \\ 0 \\ (1-s)^2 \end{bmatrix}, \quad C_O(t) = \begin{bmatrix} 1 \\ t \\ t^2 \end{bmatrix}, \quad C_N(s) = \begin{bmatrix} s \\ 1 \\ s^2 \end{bmatrix}, \quad C_W(t) = \begin{bmatrix} 0 \\ t \\ (1-t)^2 \end{bmatrix},$$



- Beispiel (cont'd)

$$C_S(s) = \begin{bmatrix} s \\ 0 \\ (1-s)^2 \end{bmatrix}, \quad C_O(t) = \begin{bmatrix} 1 \\ t \\ t^2 \end{bmatrix}, \quad C_N(s) = \begin{bmatrix} s \\ 1 \\ s^2 \end{bmatrix}, \quad C_W(t) = \begin{bmatrix} 0 \\ t \\ (1-t)^2 \end{bmatrix},$$

$$F_s(s, t) = (1-t) \cdot C_S(s) + t \cdot C_N(s) = \begin{bmatrix} (1-t)s \\ 0 \\ (1-t)(1-s)^2 \end{bmatrix} + \begin{bmatrix} ts \\ t \\ ts^2 \end{bmatrix} = \begin{bmatrix} s \\ t \\ (1-t)(1-s)^2 + ts^2 \end{bmatrix}$$

$$F_t(s, t) = (1-s) \cdot C_W(t) + s \cdot C_O(t) = \begin{bmatrix} 0 \\ (1-s)t \\ (1-s)(1-t)^2 \end{bmatrix} + \begin{bmatrix} s \\ st \\ st^2 \end{bmatrix} = \begin{bmatrix} s \\ t \\ (1-s)(1-t)^2 + st^2 \end{bmatrix}$$

$$F_{st}(s, t) = (1-s)(1-t)C_W(0) + s(1-t)C_O(0) + (1-s)tC_W(1) + stC_O(1)$$

$$= \begin{bmatrix} 0 + s(1-t) + 0 + st \\ 0 + 0 + (1-s)t + st \\ (1-s)(1-t) + 0 + 0 + st \end{bmatrix} = \begin{bmatrix} s \\ t \\ (1-s)(1-t) + st \end{bmatrix}$$

- Beispiel (cont'd)

$$C_S(s) = \begin{bmatrix} s \\ 0 \\ (1-s)^2 \end{bmatrix}, \quad C_O(t) = \begin{bmatrix} 1 \\ t \\ t^2 \end{bmatrix}, \quad C_N(s) = \begin{bmatrix} s \\ 1 \\ s^2 \end{bmatrix}, \quad C_W(t) = \begin{bmatrix} 0 \\ t \\ (1-t)^2 \end{bmatrix},$$

$$F_s(t, s)$$

=

$$\begin{bmatrix} s \\ t \\ (1-t)(1-s)^2 + ts^2 \end{bmatrix}$$

$$F_t(t, s)$$

=

$$\begin{bmatrix} s \\ t \\ (1-s)(1-t)^2 + st^2 \end{bmatrix}$$

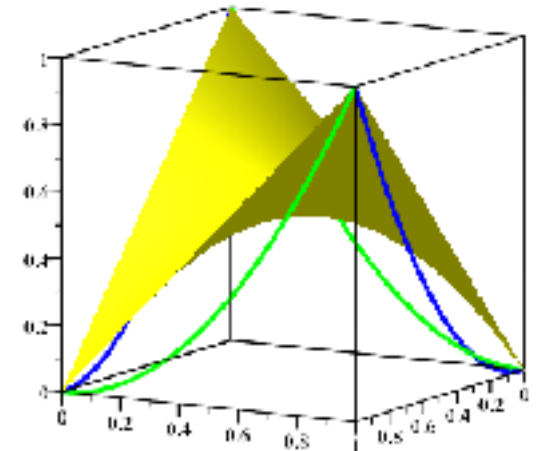
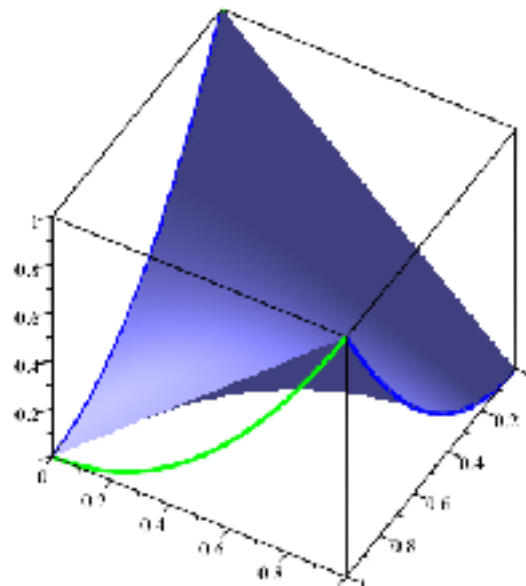
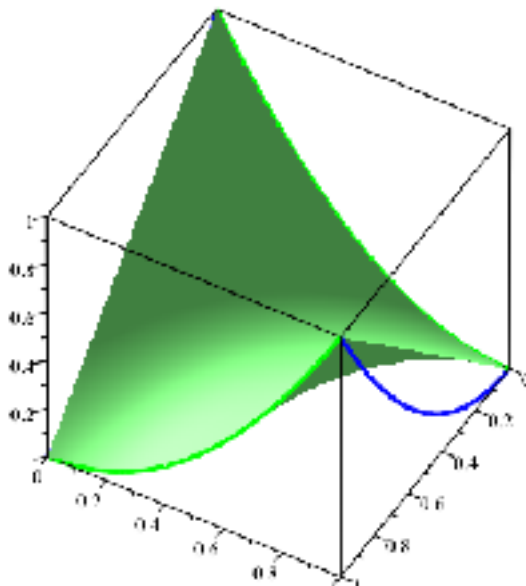
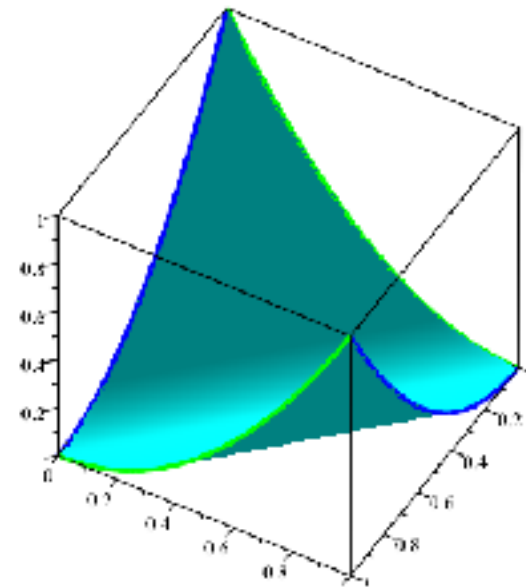
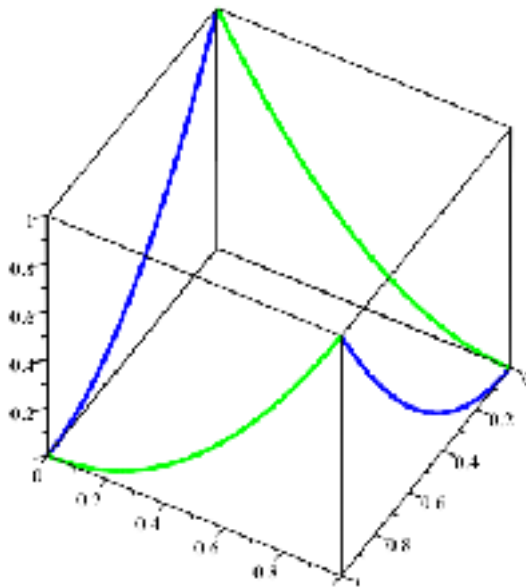
$$F_{st}(t, s)$$

=

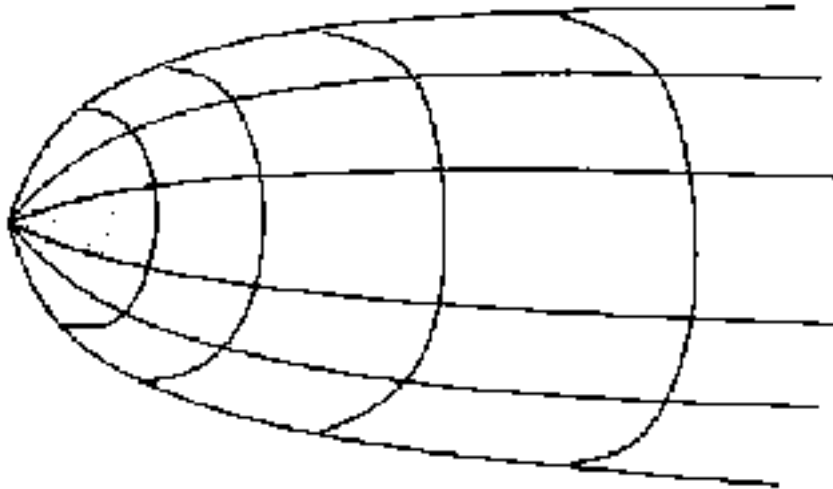
$$\begin{bmatrix} s \\ t \\ (1-s)(1-t) + st \end{bmatrix}$$

$$CP(t, s) = F_s(t, s) + F_t(t, s) - F_{st}(t, s)$$

$$= \begin{bmatrix} s \\ t \\ (1-t)(1-s)^2 + ts^2 \end{bmatrix} + \begin{bmatrix} s \\ t \\ (1-s)(1-t)^2 + st^2 \end{bmatrix} - \begin{bmatrix} s \\ t \\ (1-s)(1-t) + st \end{bmatrix} = \begin{bmatrix} s \\ t \\ (1-s-t)^2 \end{bmatrix}$$



- Für die Flächenmodellierung sind in der Regel zusätzlich dreieckige Flächen erforderlich



- Man parametrisiert diese Flächenstücke über einem dreieckigen Parametergebiet
- Und verwendet dazu beispielsweise „Bernsteinpolynome in baryzentrischen Koordinaten“

- Regelgeometrie vs. Freiformgeometrie
- Polynomkurven
 - Monom-Basis ungeeignet, Lagrange-Basis besser (Interpolationspunkte)
 - Bernstein-Basis viel besser → Bézier-Kurven
- Vorteile der Bézier-Kurven
 - Formeigenschaften (.....)
 - Auswertung mit deCasteljau (analytisch und geometrisch)
 - midpoint-subdivision (→ Zeichnen mit Hilfe von Bézier-Kurven)
- Darstellung von (Freiform-)Flächen
 - Tensorproduktansatz: bilinear und allgemeiner (TP-Bézier)
 - Transfinite Interpolation: (Lineares) Coons-Patch