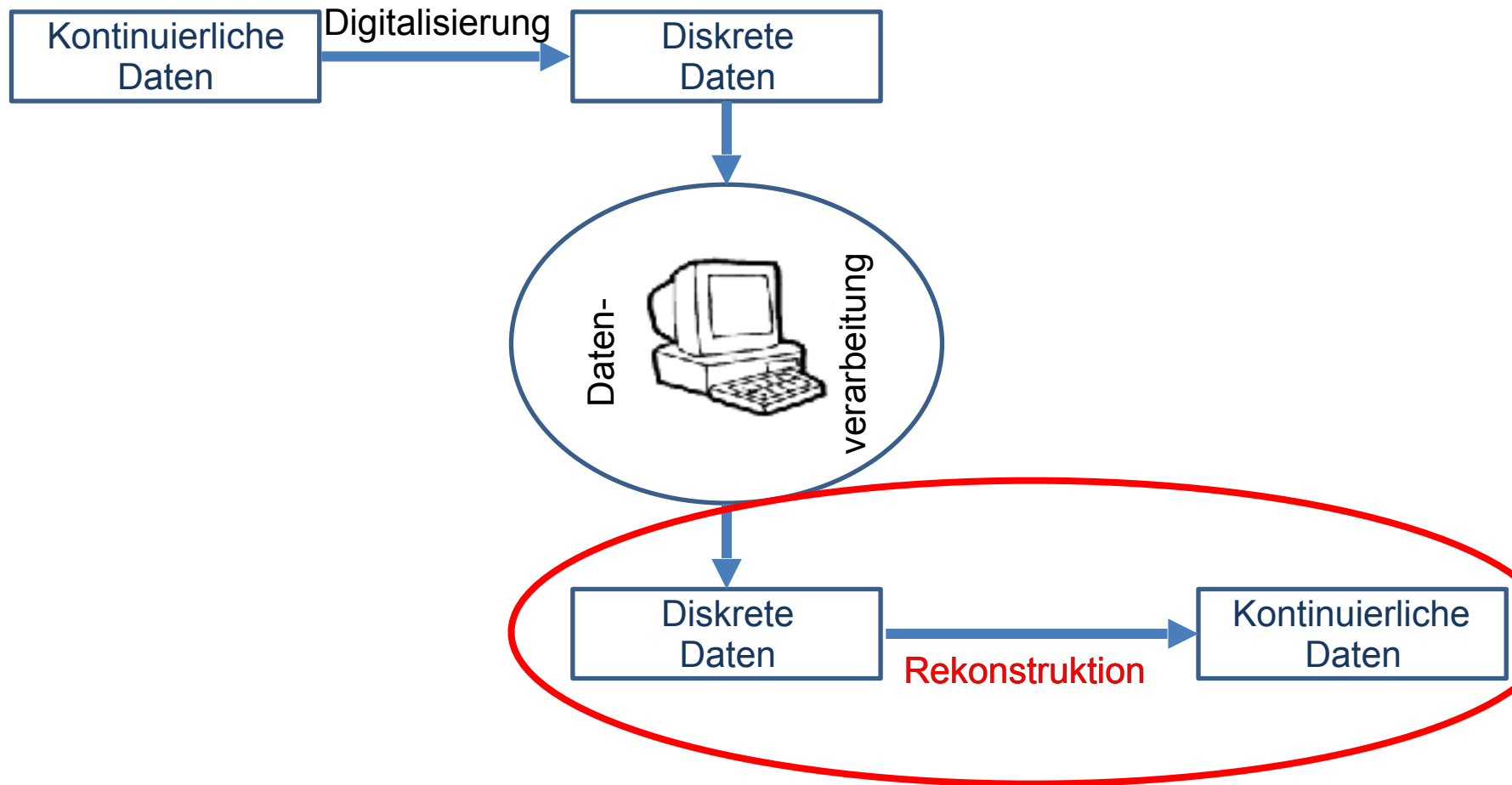


# Algorithmik kontinuierlicher Systeme

## *Rekonstruktion kontinuierlicher Daten – Multivariate Interpolation*



- Rekonstruktion kontinuierlicher Daten aus diskreten Daten



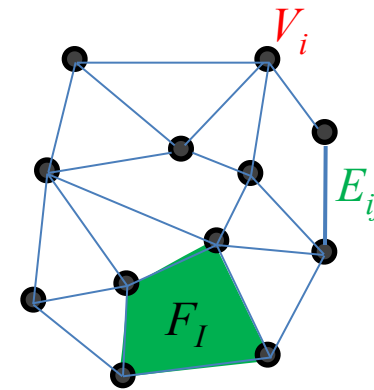
- Funktionen mehrerer Veränderlicher  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ 
  - bivariate:  $f(x,y)$
  - trivariate:  $f(x,y,z)$
- Veranschaulichung in 2D Funktionsgraph: „Gebirge“
- Gegeben *diskrete samples* einer unbekannten Funktion  
**Stützstellen**  $(x_i, y_i)$   
**Stützwerte**  $f_i = f(x_i, y_i)$
- Gesucht Näherungswerte von  $f$  an den Stellen „dazwischen“.
- Ansatz: Konstruiere eine interpolierende Funktion  $g$  :  

$$g(x_i, y_i) = f_i$$

- Erinnerung 1D : Punkte  $x_i$  bzw. Intervalle  $[x_i, x_{i+1}]$   
Geometrie (Lage der Punkte) und  
Topologie (welches sind die Nachbarn)
- 2D : Punkte mit Nachbarschaftsbeziehung  $\rightarrow$  Gitter

- Ein Gitter besteht aus

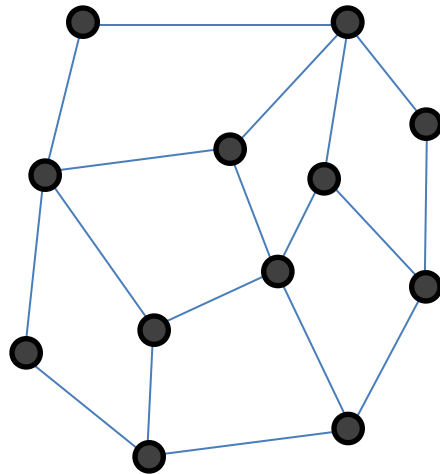
- ▶ Ecken (vertex),  $V = \{ V_i \}$
- ▶ Kanten (edge),  $E = \{ E_{ij} \}$
- ▶ Zellen (face),  $F = \{ F_I \}$



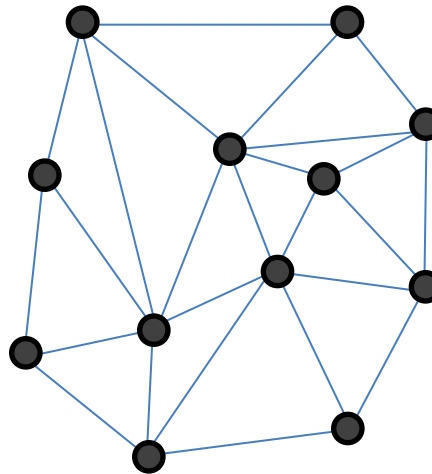
- Euler-Formel für planares Gitter ohne Löcher:

$$\underbrace{|F|}_{\text{Anzahl Zellen}} - \underbrace{|E|}_{\text{Anzahl Kanten}} + \underbrace{|V|}_{\text{Anzahl Ecken}} = 1$$

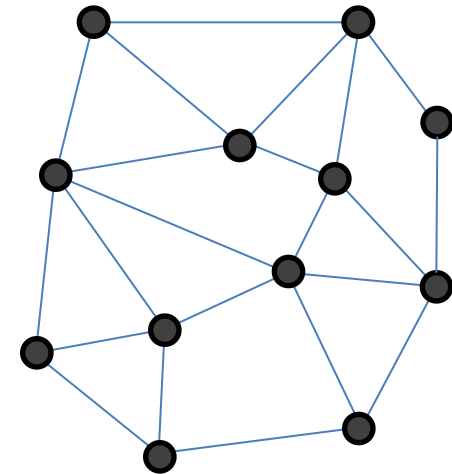
- Dreiecks-Gitter, Vierecks-G., beliebige Polygone, gemischt:  
(triangle mesh), (quad mesh), (unstructured mesh)
- Gleiche Geometrie (Lage der Punkte) aber unterschiedliche Topologie (Nachbarschaftsbeziehung)



Vierecke  
(quads)

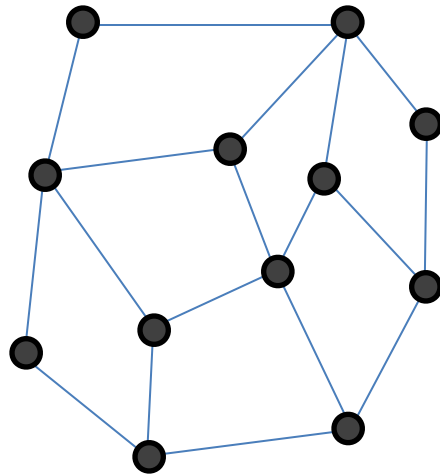


Dreiecke  
(triangles)

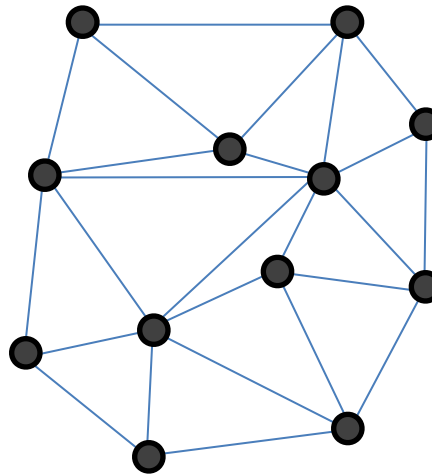


gemischt

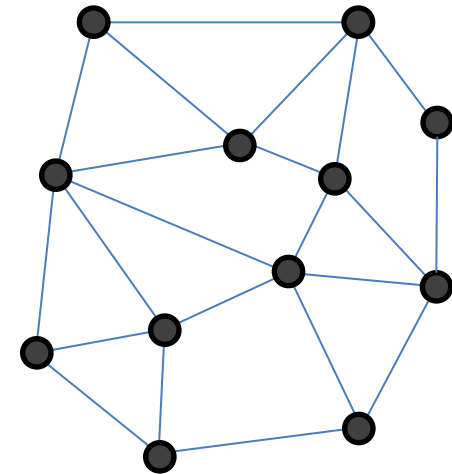
- Dreiecks-Gitter, Vierecks-G., beliebige Polygone, gemischt:  
(triangle mesh), (quad mesh), (unstructured mesh)
- Gleiche Geometrie (Lage der Punkte) aber unterschiedliche Topologie (Nachbarschaftsbeziehung)



Vierecke  
(quads)



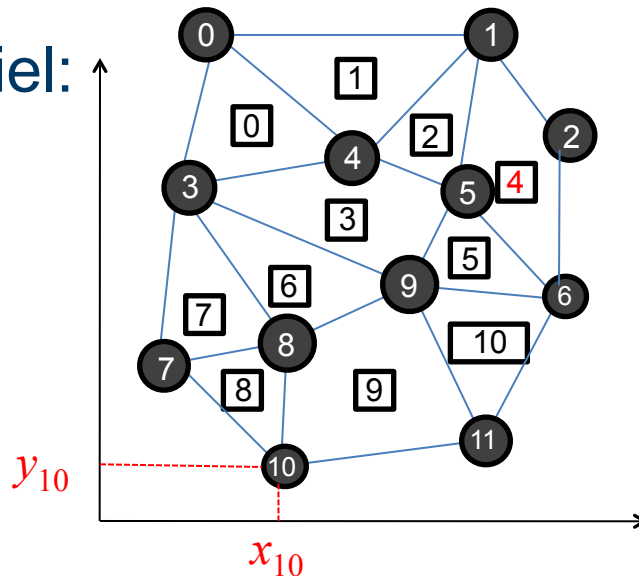
Dreiecke  
(triangles)



gemischt

- Datenstrukturen  
(shared vertex oder indexed face set) : 2 Listen
  - Liste der Punkte (vertex list)  
Koordinaten der Ecken
  - Liste der Zellen (face list)  
Verweise auf die vertex list

## Beispiel:

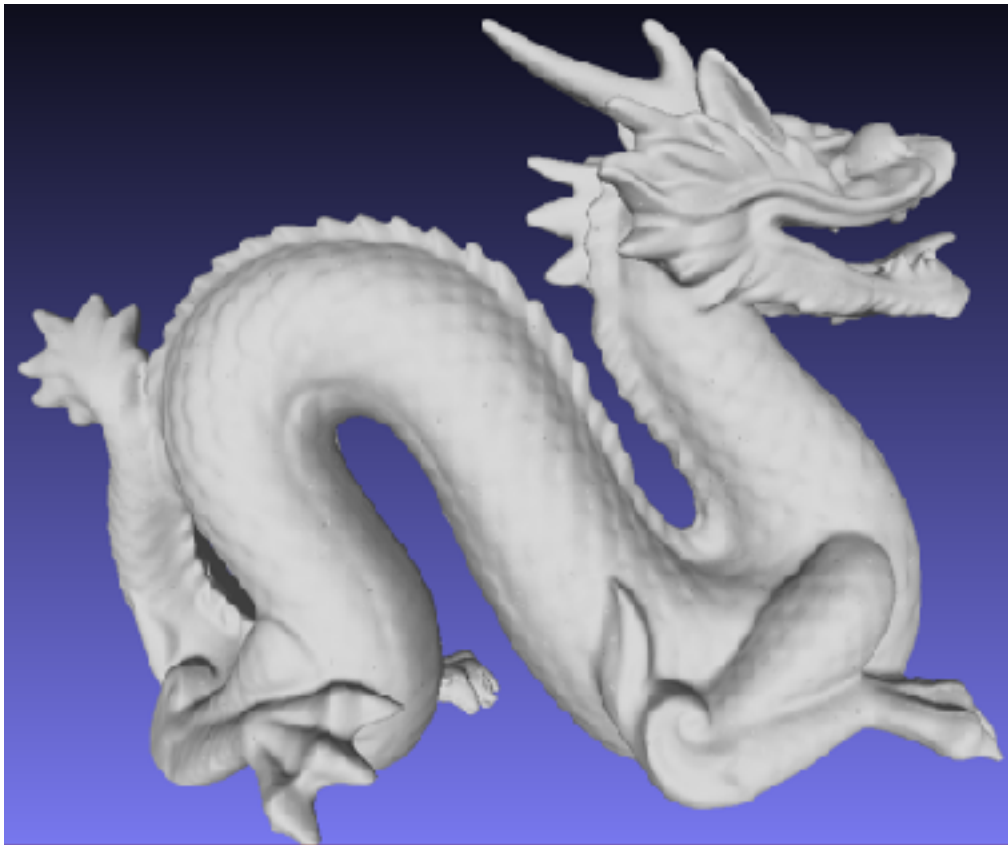


vertex list :	face list :
0 : $x_0, y_0$ ;	0 : 0,3,4;
1 : $x_1, y_1$ ;	1 : 0,4,1;
2 : $x_2, y_2$ ;	2 : 4,5,1;
3 : $x_3, y_3$ ;	3 : 3,9,5,4;
4 : $x_4, y_4$ ;	4 : 5,6,2,1;
5 : $x_5, y_5$ ;	5 : 9,6,5;
6 : $x_6, y_6$ ;	6 : 3,8,7;
7 : $x_7, y_7$ ;	7 : 7,8,3;
8 : $x_8, y_8$ ;	8 : 7,10,8;
9 : $x_9, y_9$ ;	9 : 10,11,9,8;
10 : $x_{10}, y_{10}$ ;	10 : 9,11,6;
11 : $x_{11}, y_{11}$ ;	

Achte auf die Orientierung

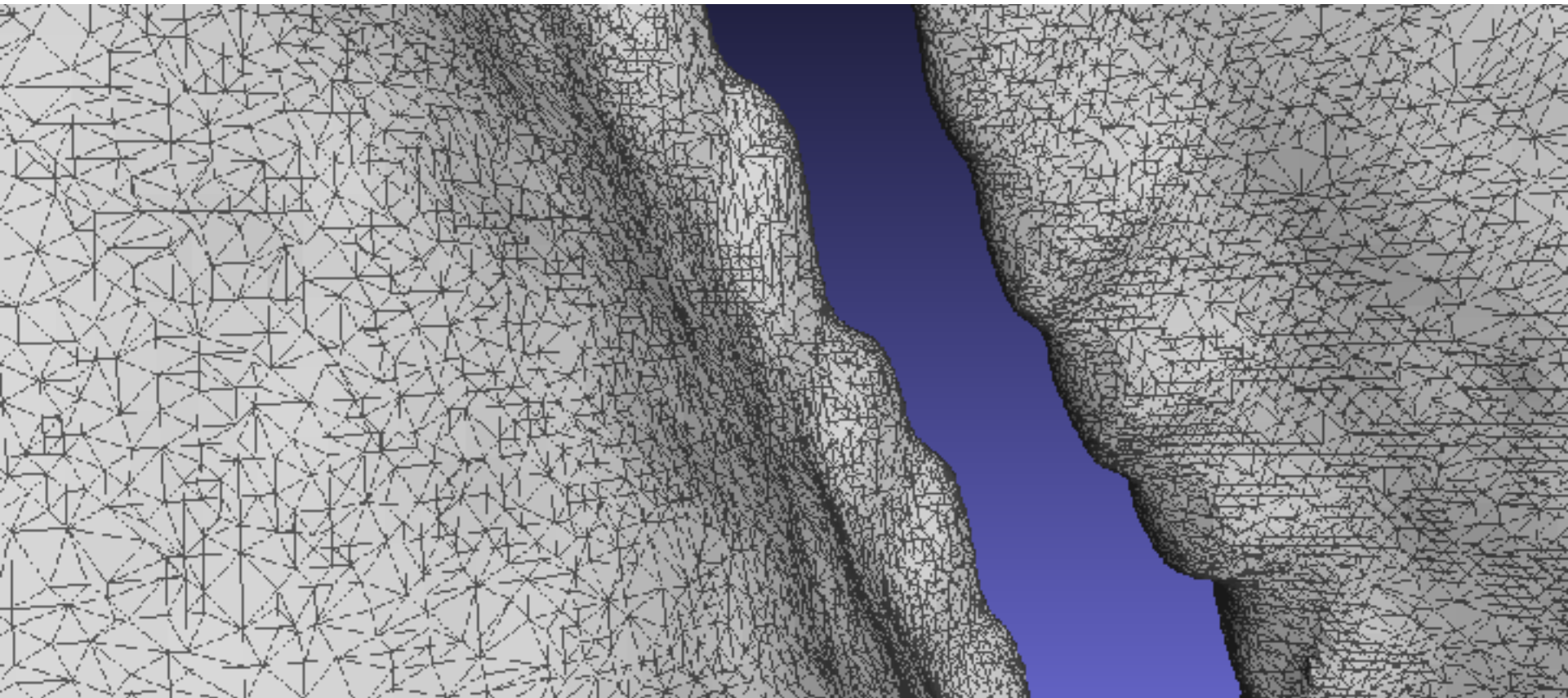
- Beispiel *shared vertex / indexed face set*
- Dragon:
  - # Vertices: 437645
  - # Faces: 871414

(MeshLab Demo)

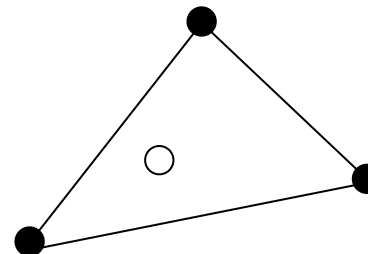
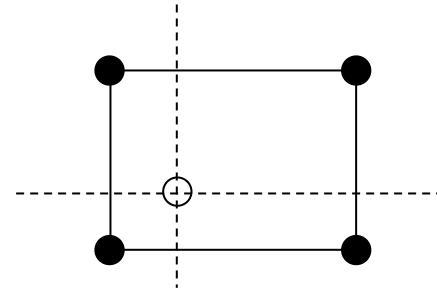




- Beispiel *shared vertex / indexed face set* (MeshLab Demo)
- Dragon:
  - # Vertices: 437645
  - # Faces: 871414



- Berücksichtige nur die Daten in der Umgebung des jeweiligen Punktes.
- Extrem **Zellenweise**: nur Werte in den Eckpunkten der Zelle
- Bilinear in 2D falls Zelle rechteckig
- Linear in 2D falls Zelle ein Dreieck



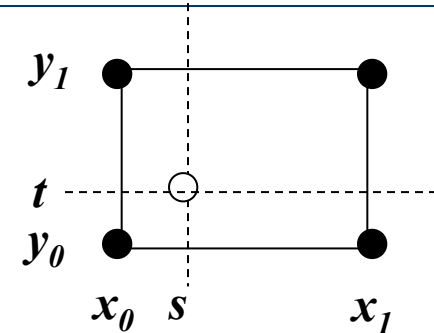
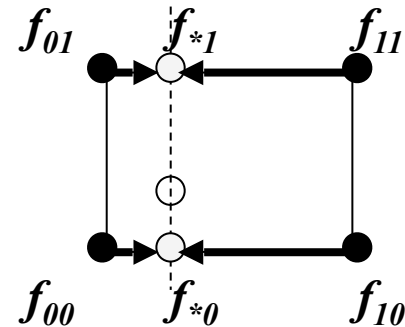
## Zweistufiges Verfahren:

### 1. Zwei lineare Interpolationen in $x$ -Richtung

$$f_{*0} = (1 - \alpha) \cdot f_{0,0} + \alpha \cdot f_{1,0}$$

$$f_{*1} = (1 - \alpha) \cdot f_{0,1} + \alpha \cdot f_{1,1}$$

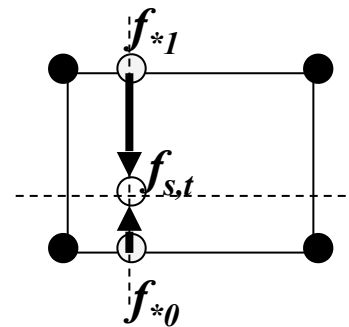
$$\text{wobei } \alpha = \frac{s - x_0}{x_1 - x_0}$$



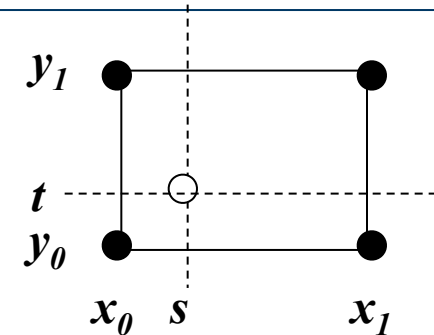
### 2. Die Ergebnisse von 1. linear in $y$ -Richtung interpolieren

$$f_{s,t} = (1 - \beta) \cdot f_{*0} + \beta \cdot f_{*1}$$

$$\text{wobei } \beta = \frac{t - y_0}{y_1 - y_0}$$

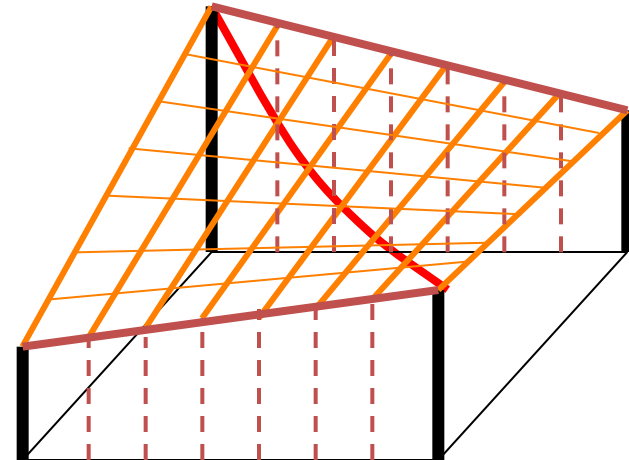
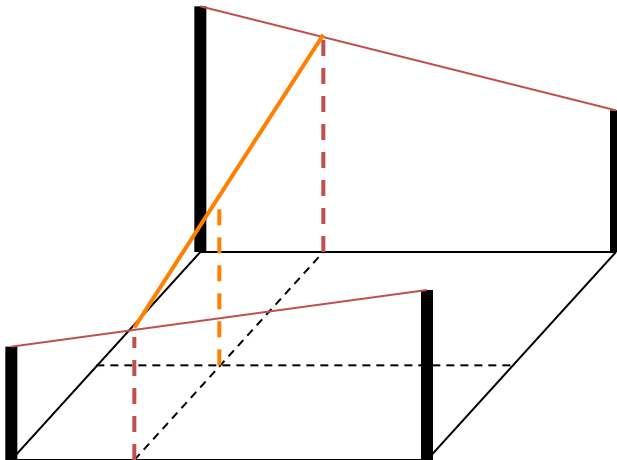


- Ergebnis der bilinearen Interpolation



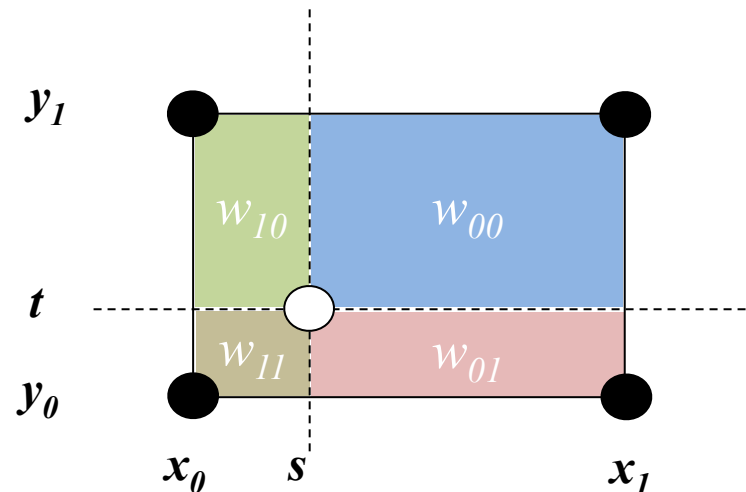
$$BL(s, t) = (1 - \beta) \left( (1 - \alpha) f_{00} + \alpha f_{10} \right) + \beta \left( (1 - \alpha) f_{01} + \alpha f_{11} \right)$$

wobei  $\alpha = \frac{s - x_0}{x_1 - x_0}$  und  $\beta = \frac{t - y_0}{y_1 - y_0}$



- Andere Sichtweise:

$$\begin{aligned}
 BL(s, t) &= \frac{y_1 - t}{y_1 - y_0} \cdot \frac{x_1 - s}{x_1 - x_0} \cdot f_{0,0} + \frac{y_1 - t}{y_1 - y_0} \cdot \frac{s - x_0}{x_1 - x_0} \cdot f_{1,0} \\
 &\quad + \frac{t - y_0}{y_1 - y_0} \cdot \frac{s - x_0}{x_1 - x_0} \cdot f_{1,1} + \frac{t - y_0}{y_1 - y_0} \cdot \frac{x_1 - s}{x_1 - x_0} \cdot f_{0,1} \\
 &= w_{00} \cdot f_{0,0} + w_{10} \cdot f_{1,0} + w_{11} \cdot f_{1,1} + w_{01} \cdot f_{0,1}
 \end{aligned}$$



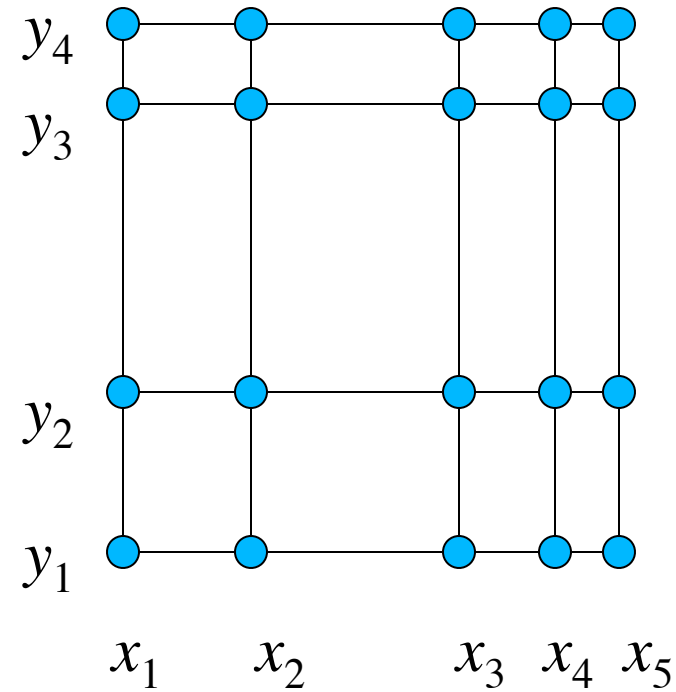
- Andere Reihenfolge - *erst zweimal in y-Richtung, dann einmal in x-Richtung* - liefert das gleiche Ergebnis.  
(Obige Formel ist unabhängig von der Reihenfolge!)
- Die bilineare Interpolierende ist i.a. keine ebene Fläche  
*Ausnahme: Die vier 3D-Punkte  $(x_i, y_j, f_{ij})$  liegen in einer Ebene*

Der bilineare Interpolant ist **keine** lineare Funktion über der Ebene, sondern i.A. ein Polynom vom Grad zwei (enthält einen Term der Form  $s \cdot t$ )

Liegen die Stützstellen auf einem „Tensorprodukt“-Gitter

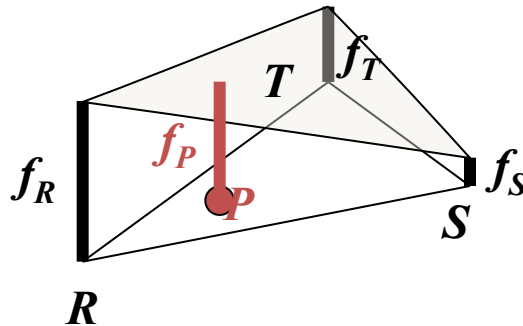
$$\{(x_i, y_j) : 1 \leq i \leq n, 1 \leq j \leq m\}$$

kann die Interpolation zuerst in  $x$ -Richtung und dann in  $y$ -Richtung (oder umgekehrt) jeweils als eindimensionale Interpolation ausgeführt werden, und dabei die bekannten Algorithmen zum Einsatz kommen. (Lösbarkeit ist gesichert)



Beachte, dass das resultierende bivariate Polynom in  $x$ -Richtung ein Polynom vom Grad  $n$  und in  $y$ -Richtung ein Polynom vom Grad  $m$  ist. In Richtung  $\xi = \alpha x + \beta y$  ist es ein Polynom vom Grad  $m+n$ .

- Interpolation in dreieckigen Zellen mit **linearer Interpolation**
- geometrisch



- Berechnung mit Hilfe **baryzentrischer Koordinaten**:



- Gegeben drei Punkte  $R, S, T$  in der Ebene, die **nicht** auf einer Geraden liegen.

Jeder Punkt  $P$  der Ebene lässt sich eindeutig wie folgt darstellen:

$$P = \rho R + \sigma S + \tau T$$

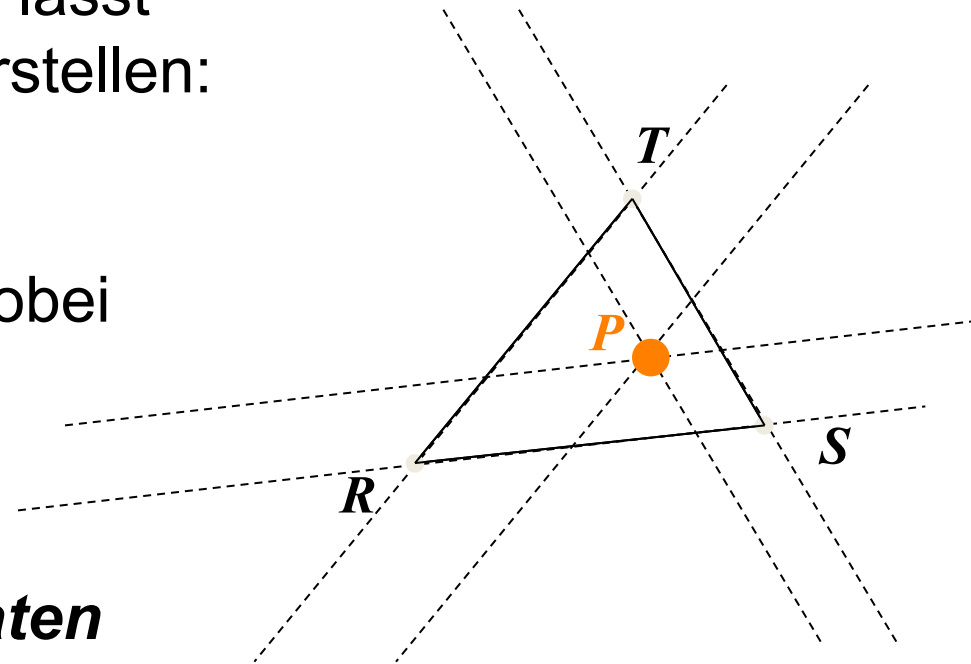
wobei

$$1 = \rho + \sigma + \tau$$

- $\rho, \sigma$  und  $\tau$  heißen

***baryzentrische Koordinaten***

von  $P$  bezüglich  $\Delta(R, S, T)$



- Drei (lineare) Gleichungen für 3 Unbekannte
- Berechnung der baryzentrischen Koordinaten:  
mit Cramerscher Regel:

$$\rho = \frac{\begin{vmatrix} P_x & S_x & T_x \\ P_y & S_y & T_y \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} R_x & S_x & T_x \\ R_y & S_y & T_y \\ 1 & 1 & 1 \end{vmatrix}} = \frac{\begin{vmatrix} P_x - T_x & S_x - T_x \\ P_y - T_y & S_y - T_y \end{vmatrix}}{\begin{vmatrix} R_x - T_x & S_x - T_x \\ R_y - T_y & S_y - T_y \end{vmatrix}}$$

$$\sigma = \frac{\begin{vmatrix} R_x - T_x & P_x - T_x \\ R_y - T_y & P_y - T_y \end{vmatrix}}{\begin{vmatrix} R_x - T_x & S_x - T_x \\ R_y - T_y & S_y - T_y \end{vmatrix}} \quad \tau = \frac{\begin{vmatrix} R_x - P_x & S_x - P_x \\ R_y - P_y & S_y - P_y \end{vmatrix}}{\begin{vmatrix} R_x - T_x & S_x - T_x \\ R_y - T_y & S_y - T_y \end{vmatrix}}$$

- Das ist in der Computergraphik Standard, aber wegen der Stabilitätsmängel der Cramer'schen Regel bei höheren Genauigkeitsanforderungen zweifelhaft.

Lösen des 3x3 Gleichungssystems:

$$\begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ y \end{bmatrix}$$

- bzw.:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & x_2 - x_1 & x_3 - x_1 \\ 0 & y_2 - y_1 & y_3 - y_1 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} 1 \\ x - x_1 \\ y - y_1 \end{bmatrix}$$

- kann (und sollte) ganz normal mit Gauss-Elimination + Pivot-Suche geschehen. Das ist insbesondere der richtige Weg für die Berechnung der baryzentrischen Koordinaten in 3D (oder noch höher-dimensional).

- Einfache Beispiele

$$P_i : (\rho, \sigma, \tau)$$

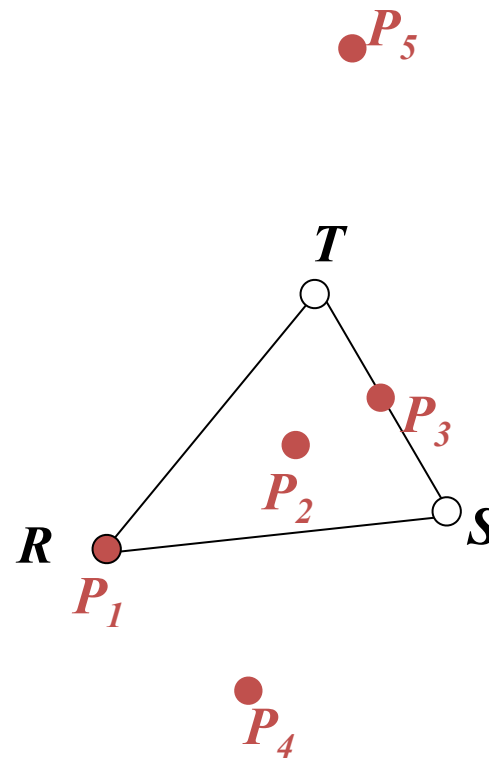
$$P_1 : (1, 0, 0) \quad (\text{Eckpunkt})$$

$$P_2 : (1/3, 1/3, 1/3) \quad (\text{Schwerpunkt})$$

$$P_3 : (0, 1/2, 1/2) \quad (\text{Kantenmitte})$$

$$P_4 : (1, 1, -1)$$

$$P_5 : (-1/2, -1/2, 2)$$



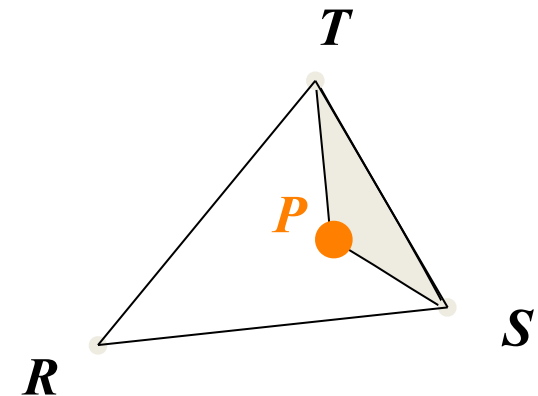
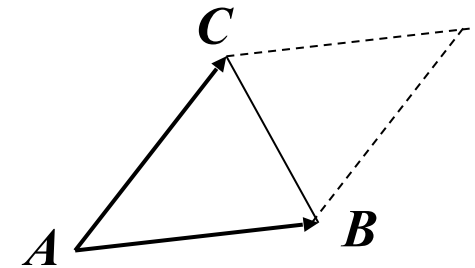
## Geometrische Interpretation (1)

- Erinnerung:  
 $\det(B-A, C-A) = \mathbf{area}(\text{Parallelogramm})$   
 $= 2 \cdot \mathbf{area}(\Delta(A, B, C))$
- $\rho = \mathbf{area}(\Delta(P, S, T)) / \mathbf{area}(\Delta(R, S, T))$

ebenso

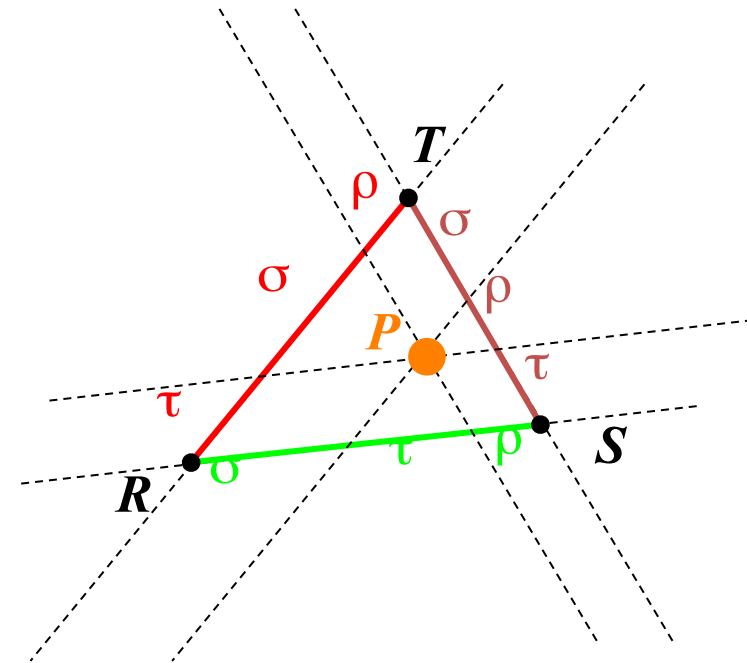
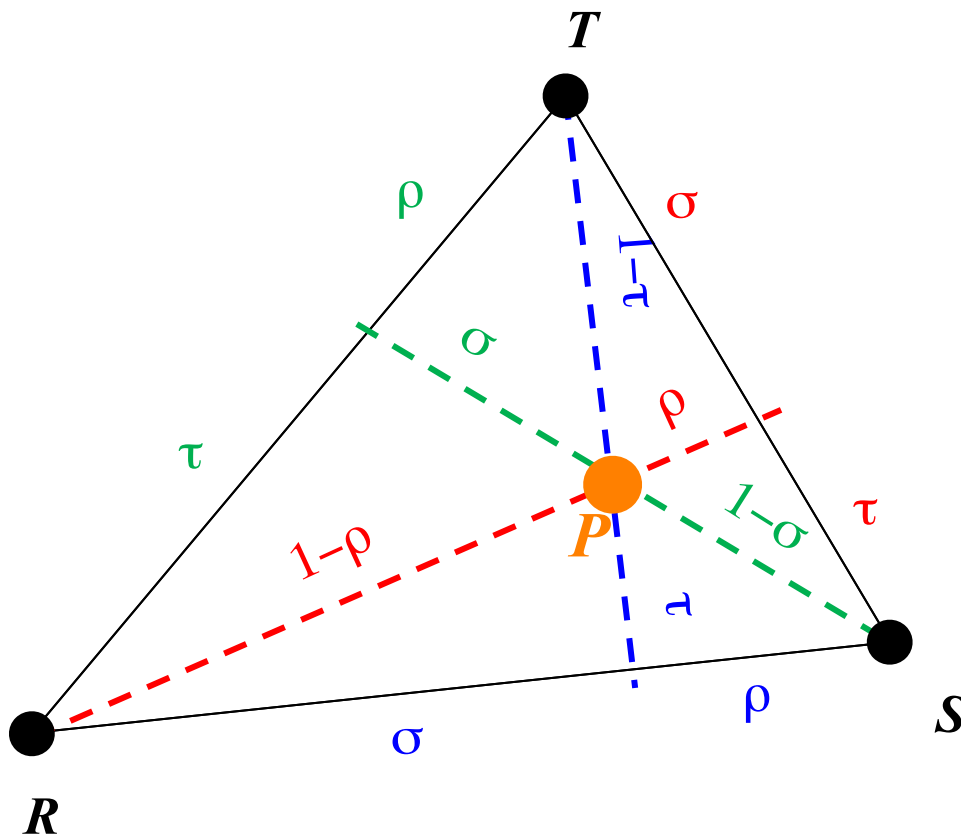
$$\sigma = \mathbf{area}(\Delta(R, P, T)) / \mathbf{area}(\Delta(R, S, T))$$

$$\tau = \mathbf{area}(\Delta(R, S, P)) / \mathbf{area}(\Delta(R, S, T))$$

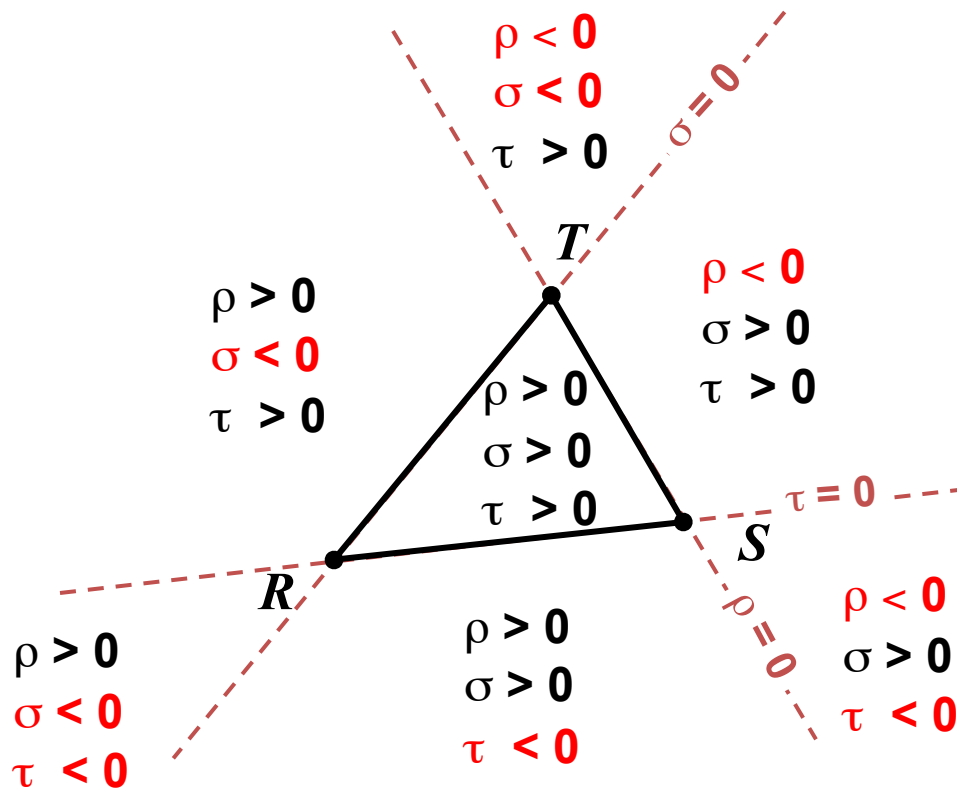


## Geometrische Interpretation (2)

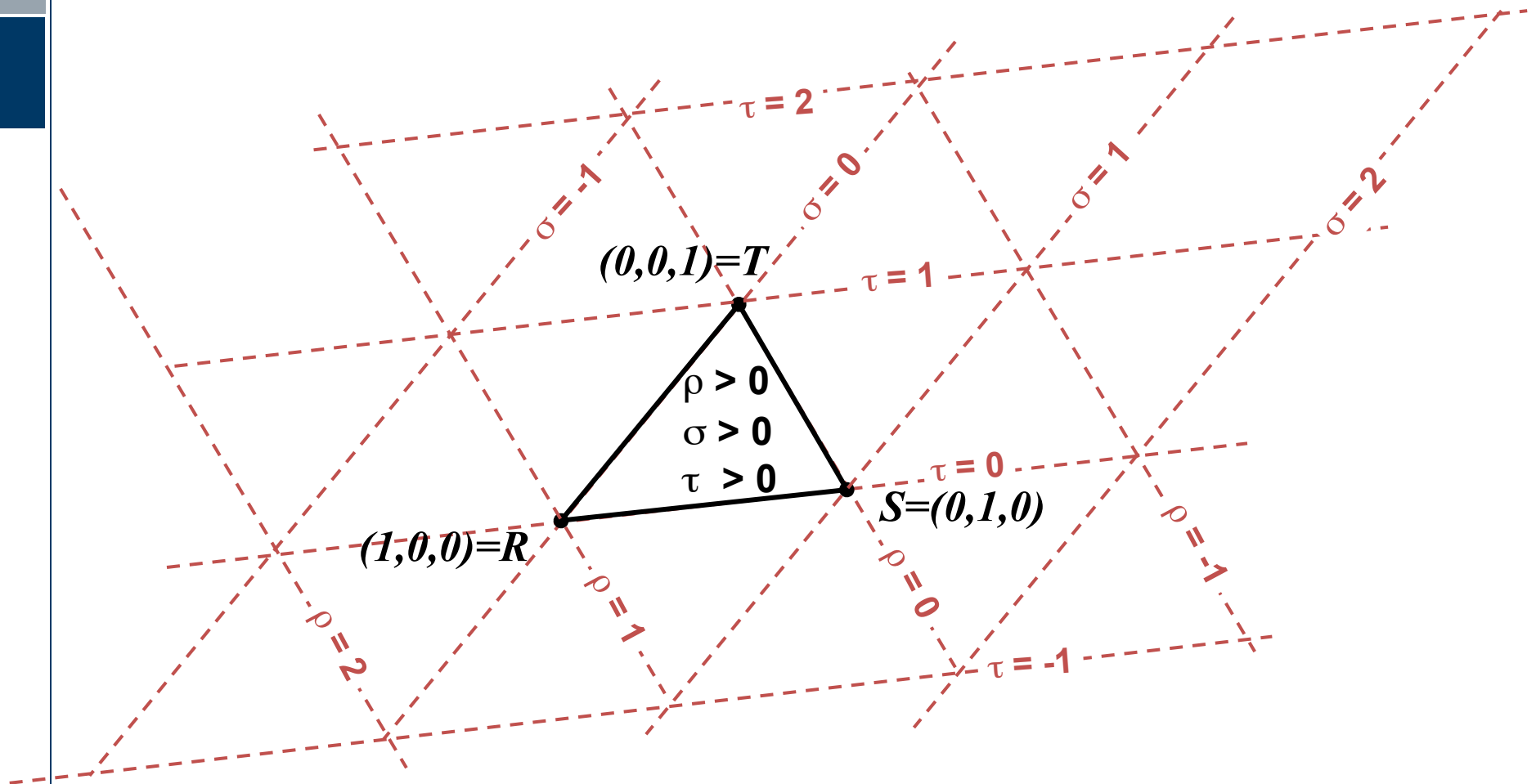
- Teilungsverhältnisse



- Vorzeichen der baryzentrischen Koordinaten

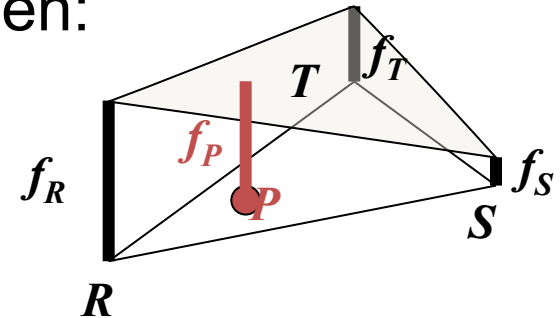


- Baryzentrische Koordinaten





- Sind nun in den Eckpunkten des Dreiecks  $\Delta(R, S, T)$  (skalare oder vektorielle) Daten gegeben:  $f_R, f_S$  und  $f_T$



- Den Wert des linearen Interpolanten an einer Stelle  $P \in \Delta(R, S, T)$  erhält man wie folgt:

$$f_P = \rho \cdot f_R + \sigma \cdot f_S + \tau \cdot f_T \quad \text{wobei } \rho, \sigma, \tau \text{ die baryzentrischen}$$

Koordinaten von  $P$  bzgl.  $\Delta(R, S, T)$  sind

- Analogie zur linearen Interpolation in 1D

- **trilinear in 3D**

falls lokale Umgebung ein Quader

4-mal in  $x$ -Richtung linear interpolieren

2-mal in  $y$ -Richtung -- “ --

1-mal in  $z$ -Richtung -- “ --

- **multi-linear (in nD):**  $2^{n-1} + 2^{n-2} + \dots + 2^2 + 2 + 1$  lineare Interpolationen

- **linear**

falls lokale Umgebung ein Simplex (ein Tetraeder in 3D)

= konvexe Hülle von  $n+1$  Punkten *in allgemeiner Lage*

Baryzentrische Koordinaten eines Punktes  $P$  (im 3D-Fall):

$$P = \rho R + \sigma S + \tau T + \upsilon U \quad \text{mit} \quad \rho + \sigma + \tau + \upsilon = 1$$

Alles Weitere wie in 2D !

Aber Achtung, dass die baryzentrischen Koordinaten stabil berechnet werden. Computergraphikbücher sind da manchmal etwas „uninformiert“

- Quader in 3D :  $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}]$

- 4-mal in x-Richtung

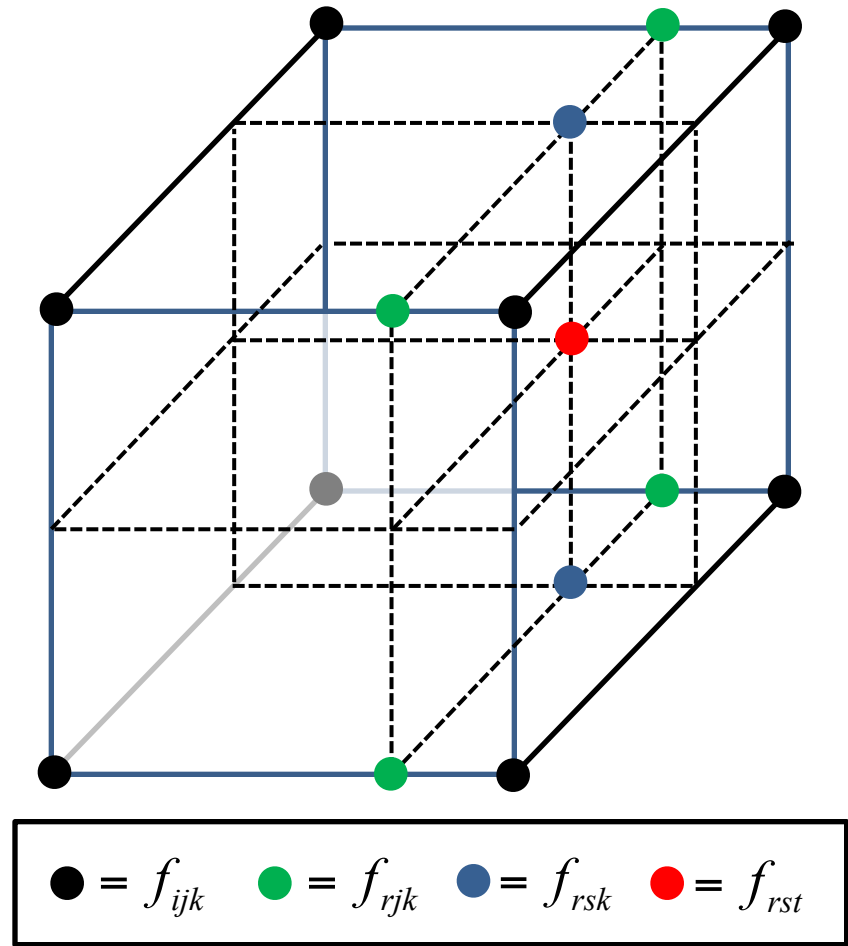
$$\bullet \quad f_{rjk} = \frac{x_{\max} - r}{x_{\max} - x_{\min}} \cdot f_{0jk} + \frac{r - x_{\min}}{x_{\max} - x_{\min}} \cdot f_{1jk}$$

- 2-mal in y-Richtung

$$\bullet \quad f_{rsk} = \frac{y_{\max} - s}{y_{\max} - y_{\min}} \cdot f_{r0k} + \frac{s - y_{\min}}{y_{\max} - y_{\min}} \cdot f_{r1k}$$

- 1-mal in z-Richtung

$$\bullet \quad f_{rst} = \frac{z_{\max} - t}{z_{\max} - z_{\min}} \cdot f_{rs0} + \frac{t - z_{\min}}{z_{\max} - z_{\min}} \cdot f_{rs1}$$

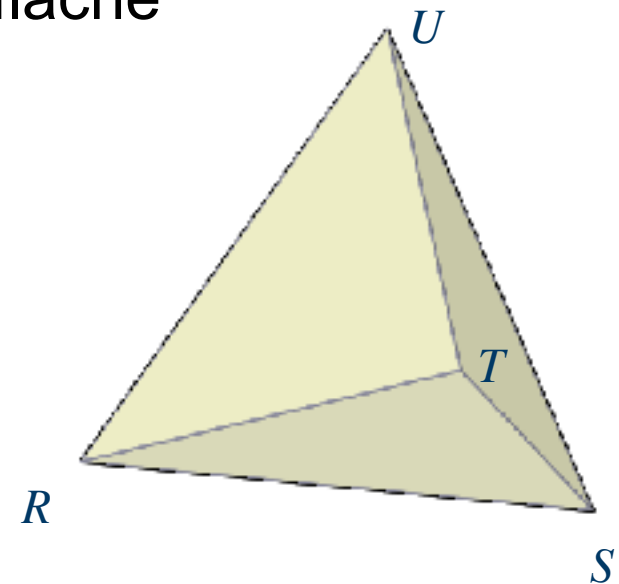


- **linear**  
falls lokale Umgebung ein **Simplex**  
(Verallgemeinerung von Dreieck nach nD)  
= konvexe Hülle von  $n+1$  Punkten *in allgemeiner Lage*.
- Simplex in 3D: konvexe Hülle von 4 Punkten die nicht in einer Ebene liegen → **Tetraeder**  
= Pyramide über dreieckiger Grundfläche
- Baryzentrische Koordinaten eines Punktes  $P$  (im 3D-Fall):

$$P = \rho R + \sigma S + \tau T + \upsilon U$$

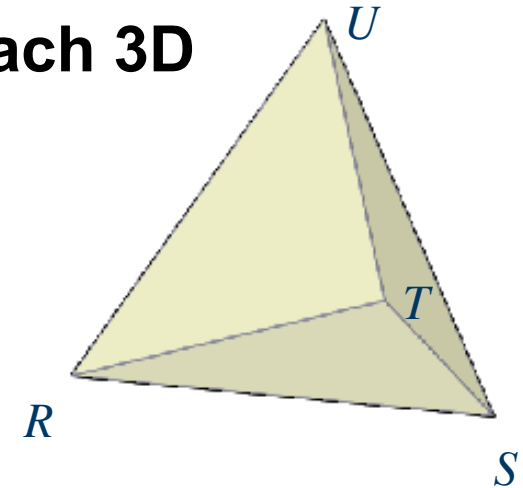
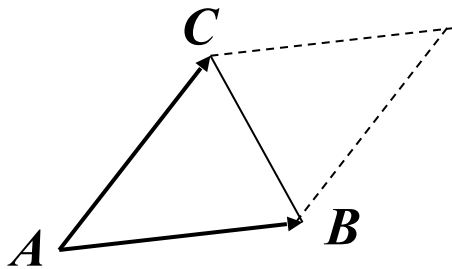
mit

$$\rho + \sigma + \tau + \upsilon = 1$$

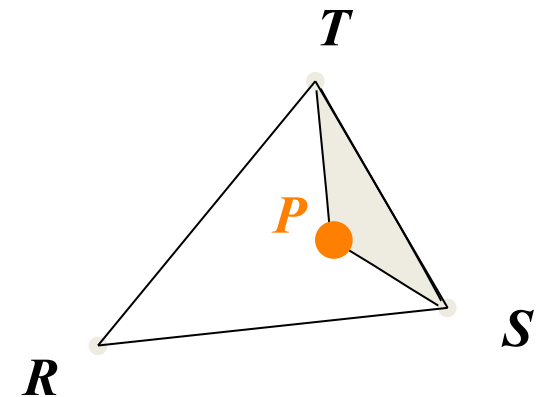


## Geometrische Interpretation: Übertrage nach 3D

- Erinnerung:  
 $\det(B-A, C-A) = \mathbf{area}(\text{Parallelogramm})$   
 $= 2 \cdot \mathbf{area}(\Delta(A, B, C))$

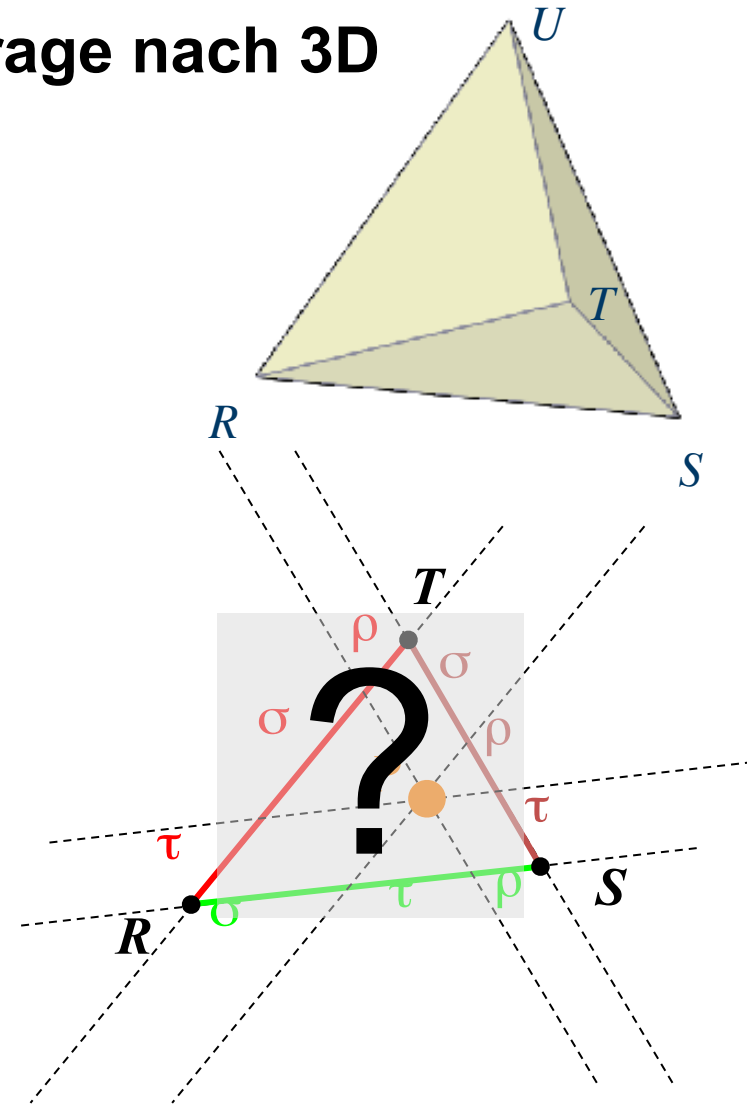
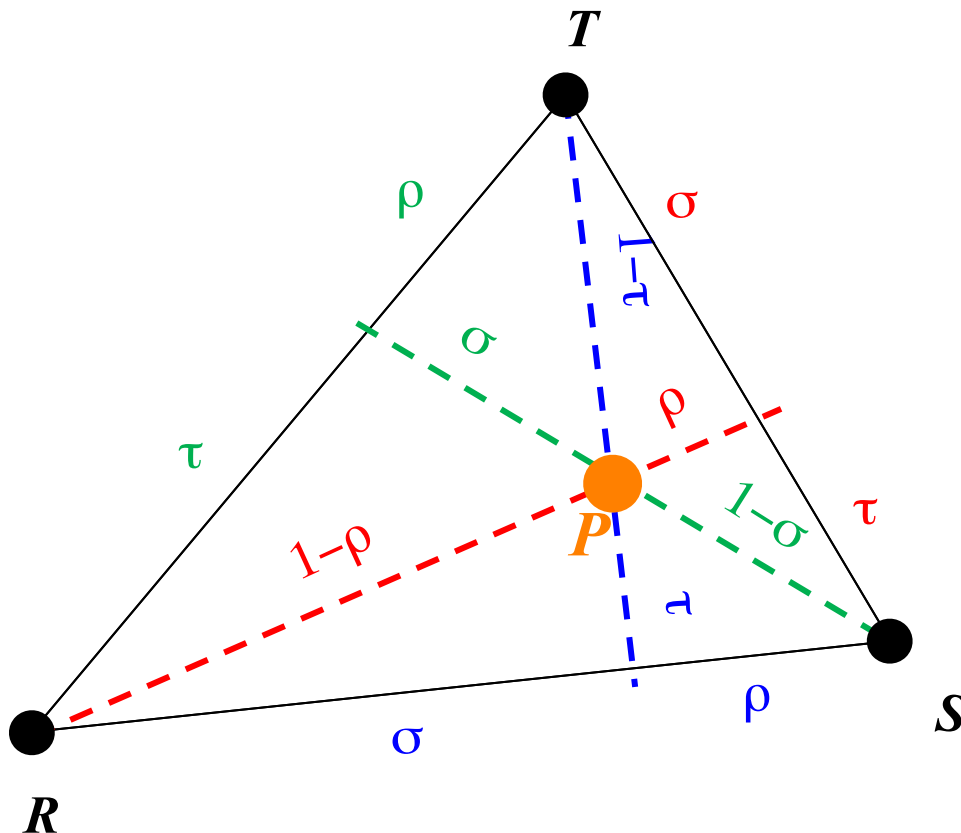


- $\rho = \mathbf{area}(\Delta(P, S, T)) / \mathbf{area}(\Delta(R, S, T))$   
 ebenso  
 $\sigma = \mathbf{area}(\Delta(R, P, T)) / \mathbf{area}(\Delta(R, S, T))$   
 $\tau = \mathbf{area}(\Delta(R, S, P)) / \mathbf{area}(\Delta(R, S, T))$



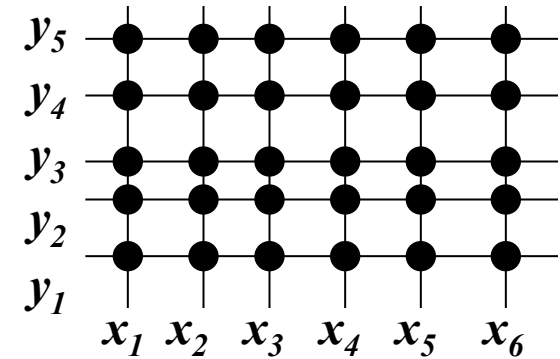
## Geometrische Interpretation: Übertrage nach 3D

- Teilungsverhältnisse



- Tensor-Produkt-Ansatz der 1D-Verfahren
  - nur für Spezialfälle (gridded data)
- Radiale Basis-Funktion (RBF)
  - geht immer

- Tensorprodukt-Ansatz:  
Voraussetzung: Stützstellen  
bilden rechteckiges Array:  
 $\{ (x_i, y_j) : 1 \leq i \leq n, 1 \leq j \leq m \}$



- zB Polynominterpolation: Gesucht ein Polynom

$$p(x, y) = \sum_{p=0}^{n-1} \sum_{q=0}^{m-1} c_{pq} x^p y^q$$

das die Interpolationbedingungen

$$p(x_i, y_j) = f_{ij}$$

erfüllt.



- Tensorprodukt-Ansatz:

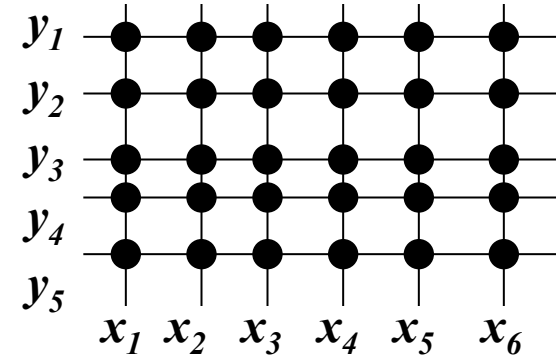
$$p(x, y) = \sum_p \sum_q c_{pq} x^p y^q$$

- Die Interpolationsbedingungen in Matrixschreibweise:

$$\left( \sum_{p=0}^{n-1} \sum_{q=0}^{m-1} c_{pq} x_i^p y_j^q \right)_{i,j} =$$

$$= \begin{pmatrix} x_1^0 & \cdots & x_1^{n-1} \\ \vdots & \ddots & \vdots \\ x_n^0 & \cdots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} c_{00} & \cdots & c_{0m-1} \\ \vdots & \ddots & \vdots \\ c_{n-10} & \cdots & c_{n-1m-1} \end{pmatrix} \begin{pmatrix} y_1^0 & \cdots & y_m^0 \\ \vdots & \ddots & \vdots \\ y_1^{m-1} & \cdots & y_m^{m-1} \end{pmatrix} =$$

$$= \begin{pmatrix} f_{11} & \cdots & f_{1m} \\ \vdots & \ddots & \vdots \\ f_{n1} & \cdots & f_{nm} \end{pmatrix}$$

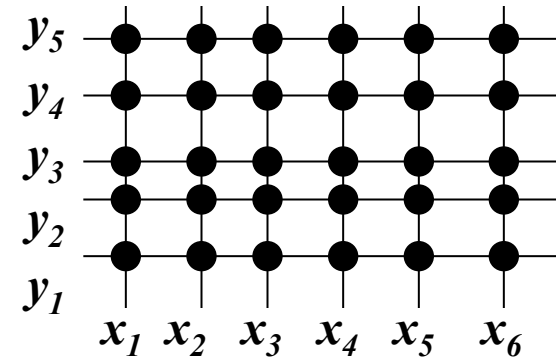


- Matrixschreibweise in Kurzform  $\mathbf{XCY}^T = \mathbf{F}$   
dabei sind  $\mathbf{X}$  und  $\mathbf{Y}$  die Vandermonde-Matrizen zu den Stützstellen  $\{x_i\}$  bzw.  $\{y_j\}$ .
- Lösungsstrategie:
  - Löse  $\mathbf{XD} = \mathbf{F}$   
das sind  $m$  1D-Probleme  
(eines für jede Spalte von  $\mathbf{F}$  )!
  - Löse  $\mathbf{CY}^T = \mathbf{D}$  bzw.  $\mathbf{YC}^T = \mathbf{D}^T$   
das sind  $n$  1D-Probleme (für jede Spalte von  $\mathbf{D}^T$  )

- Tensorprodukt-Ansatz (mit Newton-Polynomen)

$$p(x, y) = \sum_{p=0}^{n-1} \sum_{q=0}^{m-1} d_{pq} N_p(x) N_q(y)$$

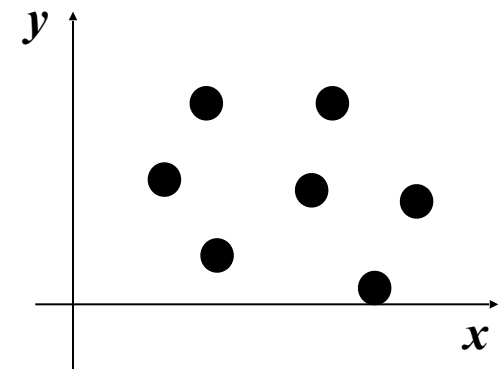
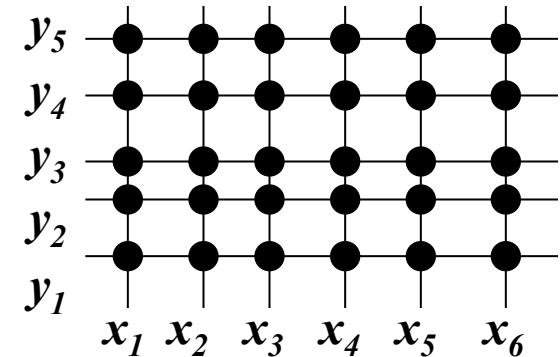
- Die Interpolationsbedingungen in Matrixschreibweise:



$$\begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 1 & x_2 - x_1 & & & 0 \\ \vdots & & \ddots & & \vdots \\ 1 & x_{n-1} - x_1 & \cdots & N_{n-1}(x_{n-1}) & 0 \\ 1 & x_n - x_1 & \cdots & N_{n-1}(x_n) & N_n(x_n) \end{pmatrix} \begin{pmatrix} d_{00} & \cdots & d_{0m} \\ \vdots & \ddots & \vdots \\ d_{n-10} & \cdots & d_{n-1m-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 & 1 \\ 0 & y_2 - y_1 & & y_{n-1} - y_1 & y_n - y_1 \\ \vdots & & \ddots & & \vdots \\ 0 & & & N_{n-1}(y_{n-1}) & N_{n-1}(y_n) \\ 0 & 0 & \cdots & 0 & N_n(y_n) \end{pmatrix} = \begin{pmatrix} f_{11} & \cdots & f_{1m} \\ \vdots & \ddots & \vdots \\ f_{n1} & \cdots & f_{nm} \end{pmatrix}$$

- Matrixschreibweise in Kurzform  $\mathbf{XDY}^T = \mathbf{F}$ .  
Dabei sind  $\mathbf{X}$  und  $\mathbf{Y}$  Dreiecksmatrizen (die Newton-Polynome ausgewertet in den Stützstellen  $\{x_i\}$  bzw.  $\{y_j\}$  ).
- Lösungsstrategie:
  - Löse  $\mathbf{XE} = \mathbf{F}$   
das sind  $m$  1D-Probleme  
(eines für jede Spalte von  $\mathbf{F}$  )!
  - Löse  $\mathbf{DY}^T = \mathbf{E}$  bzw.  $\mathbf{YD}^T = \mathbf{E}^T$   
das sind  $n$  1D-Probleme (für jede Spalte von  $\mathbf{E}^T$  )
- Diese  $n+m$  1D Interpolationsprobleme kann man mit Aitken-Neville effizient lösen!

- Tensorprodukt-Ansatz:  
Voraussetzung: Stützstellen  
bilden rechteckiges Array:  
 $\{ (x_i, y_j) : 1 \leq i \leq n, 1 \leq j \leq m \}$
- Effizientes Lösungsverfahren durch Reduktion auf 1D-Probleme
- Im allgemeinen Fall ist Polynom-Interpolation nicht möglich
  - Zahl der Freiheitsgrade ist problematisch (zB welchen Polynomgrad für 8 Punkte)
  - (Theoretische) Lösbarkeit nicht gesichert (zB 6 symmetrisch angeordnete Punkte)
  - Mehrdeutigkeit  
→ Schwieriges mathematisches Problem: (Algebraische Geometrie)



## *Radiale Basis-Funktionen* (RBF)

- Idee: 1D-Funktion  $\tilde{h}(r)$  radial-symmetrisch fortsetzen

- Beispiele

- ▶ Multi-Quadriken

$$\tilde{h}(r) = \sqrt{R^2 + r^2} \quad (R > 0)$$

- ▶ Inverse Multi-Quadriken

$$\tilde{h}(r) = \frac{1}{\sqrt{R^2 + r^2}} \quad (R > 0)$$

- ▶ Thin-Plate-Spline

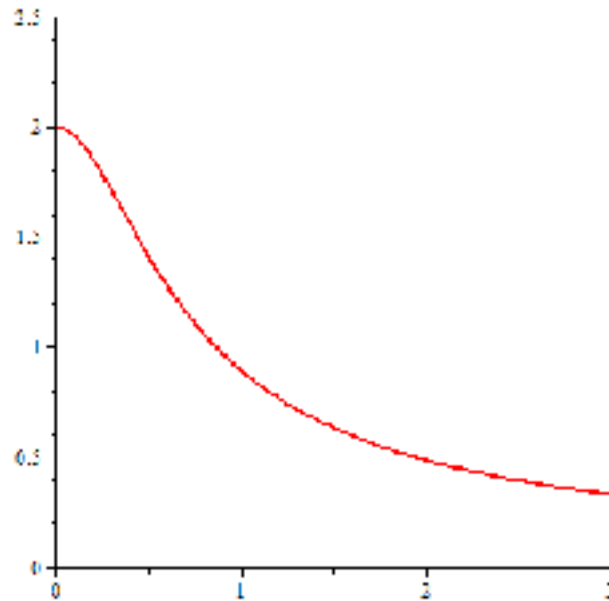
$$\tilde{h}(r) = r^2 \cdot \log(r)$$

- Radial symmetrische Fortsetzung (in 2D):

$$h_0(x, y) = \tilde{h}\left(\sqrt{x^2 + y^2}\right)$$

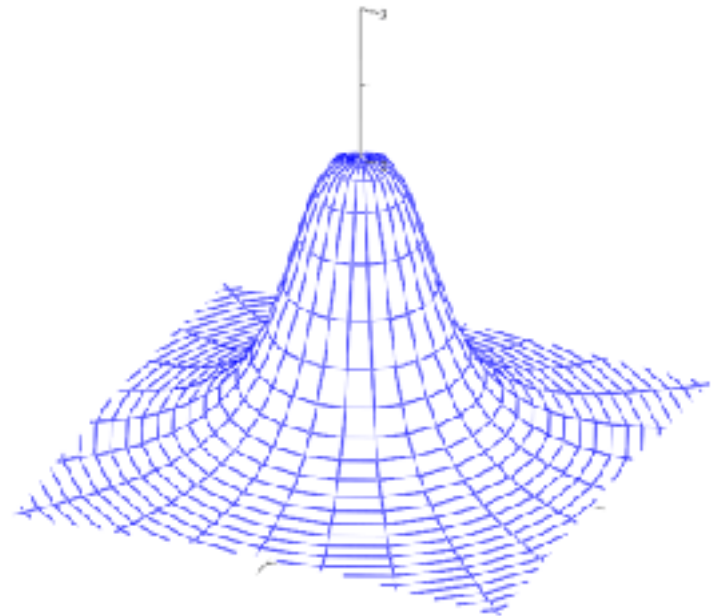
- Beispiel **Inverse Multi-Quadrik** (mit Konstante  $R = 0.5$ )

1D-Funktion



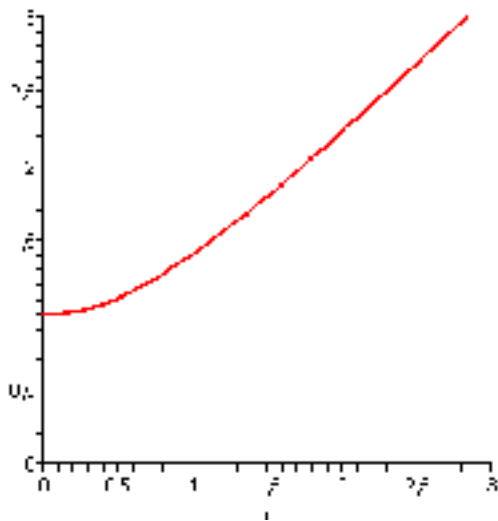
$$\tilde{h}(r) = \frac{1}{\sqrt{0.25 + r^2}}$$

radial-symmetrische Fortsetzung

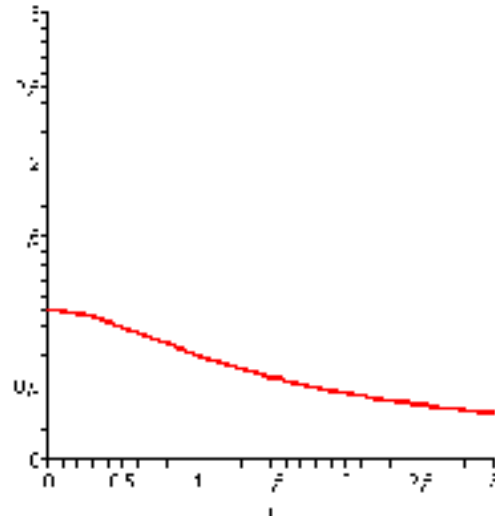


$$h_0(x, y) = \tilde{h}\left(\sqrt{x^2 + y^2}\right)$$

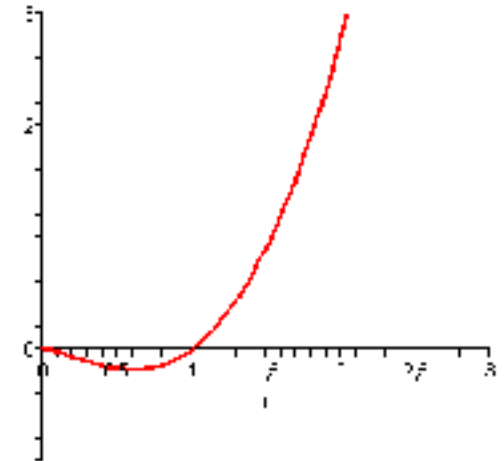
- Beispiele der 1D-Funktionen  $\tilde{h}$



Multiquadrik (R=1)



Inverse Multiquadrik (R=1)



Thin Plate Spline



- Ansatz für den Interpolanten:

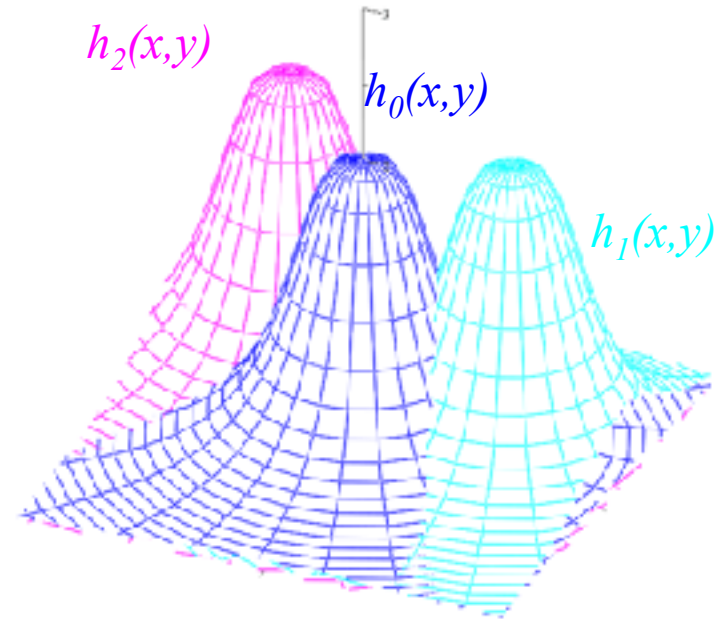
$$g(x, y) = \sum_{i=1}^n d_i h_i(x, y)$$

wobei die Funktionen  $h_i$  die um die Stützstellen  $(x_i, y_i)$  zentrierten

**radialen Basis-Funktionen** sind:

$$h_i(x, y) = h_0(x - x_i, y - y_i)$$

- Die Koeffizienten  $d_i$  müssen so bestimmt werden, dass die Interpolationsbedingungen erfüllt sind.



- Ansatz für den Interpolanten:

$$g(x, y) = \sum_{i=1}^n d_i h_i(x, y)$$

mit noch  $n$  unbekannten Koeffizienten  $d_i$

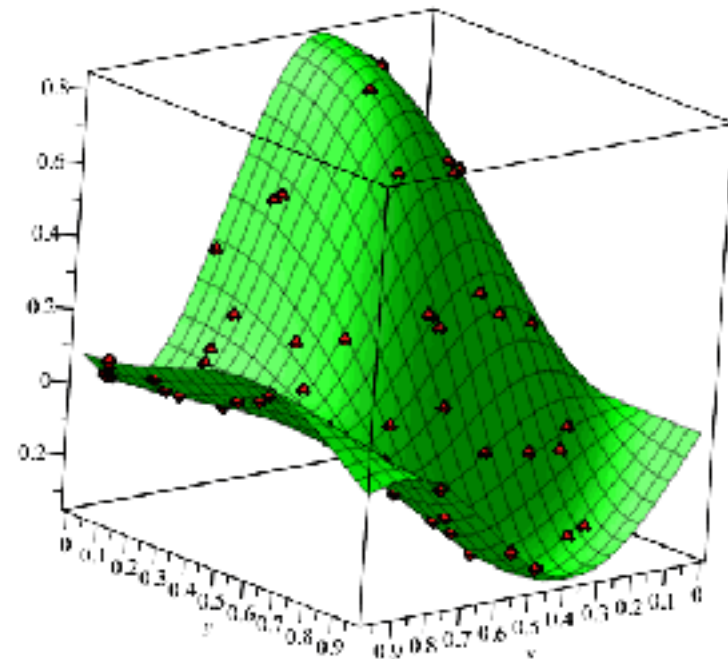
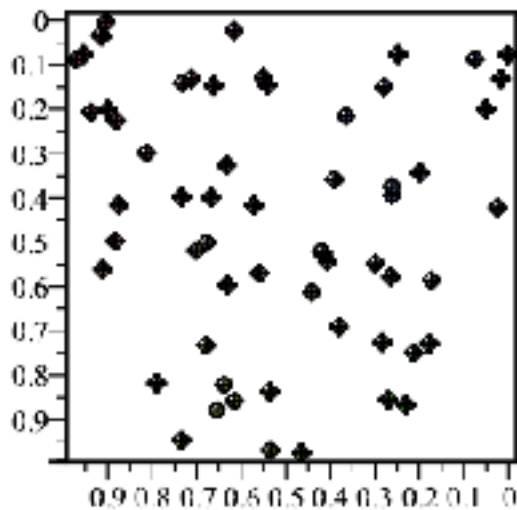
- Diese werden festgelegt durch die  $n$  Interpolationsbedingungen 
$$g(x_j, y_j) = \sum_{i=1}^n d_i h_i(x_j, y_j) = f_j$$

- Lineares System

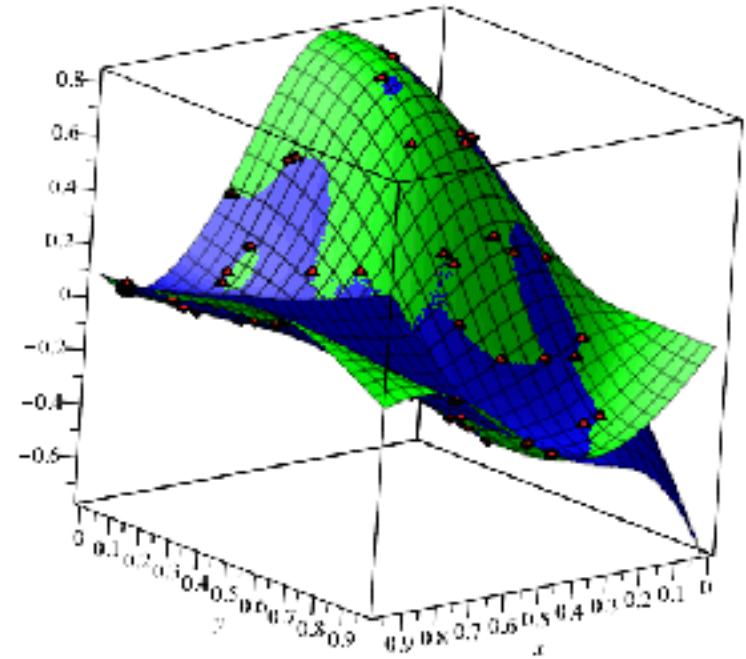
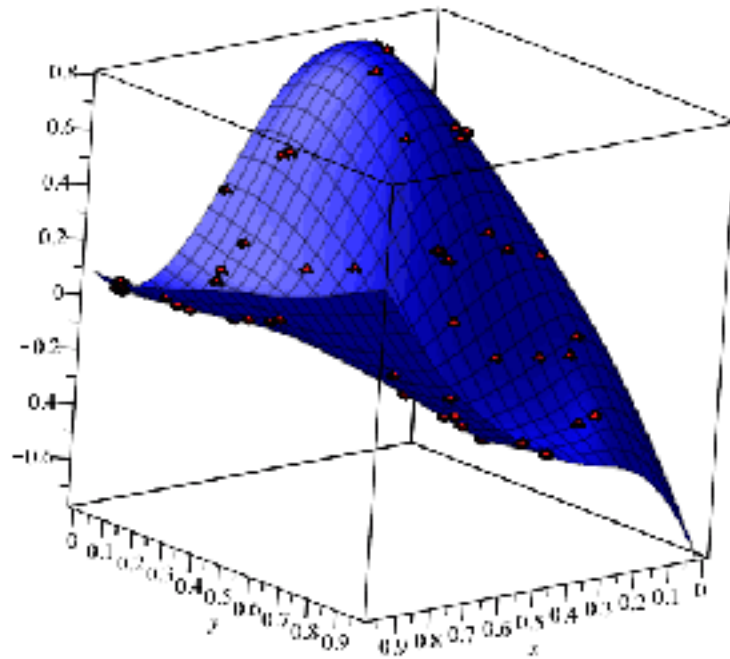
$$\begin{pmatrix} h_1(x_1, y_1) & h_2(x_1, y_1) & \cdots & h_n(x_1, y_1) \\ h_1(x_2, y_2) & h_2(x_2, y_2) & \cdots & h_n(x_2, y_2) \\ \vdots & \vdots & \ddots & \vdots \\ h_1(x_n, y_n) & h_2(x_n, y_n) & \cdots & h_n(x_n, y_n) \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$$

- Lösbarkeit des LGS ist gesichert (für MQ, InvMQ und TPS, bei beliebiger Lage der Stützstellen)
- Bei vielen Punkten ( $> 1000$ ) schlecht konditioniert  
→ sorgfältig lösen.
- Liefert auch im Fall ungleichmäßig verteilter Daten gute Ergebnisse.
- Bei MQ und InvMQ muss die Konstante  $R$  ermittelt werden, abhängig von der mittleren Punktedichte  
Beachte: bei TPS ist dies nicht notwendig!
- Lineare Funktionen werden nicht exakt rekonstruiert  
→ *RBF mit linearer Präzision* (s.u.)
- Übertragung auf 3D und  $nD$  ist möglich (andere  $\tilde{h}$ !)

- 60 zufällige Samples - Originalfunktion (grün)

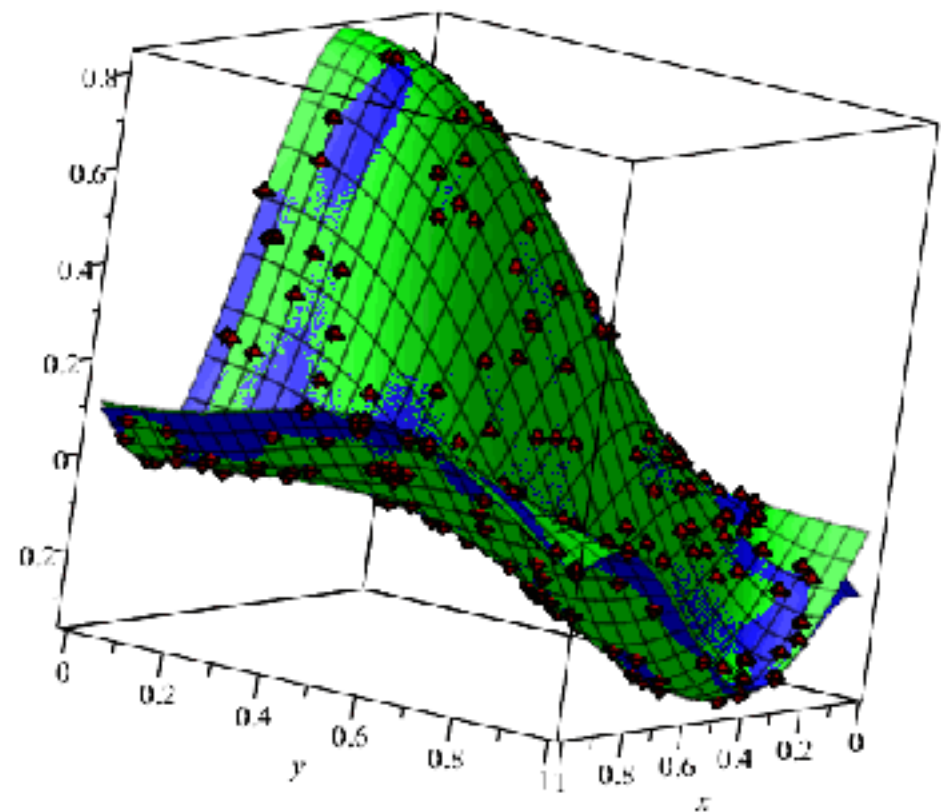
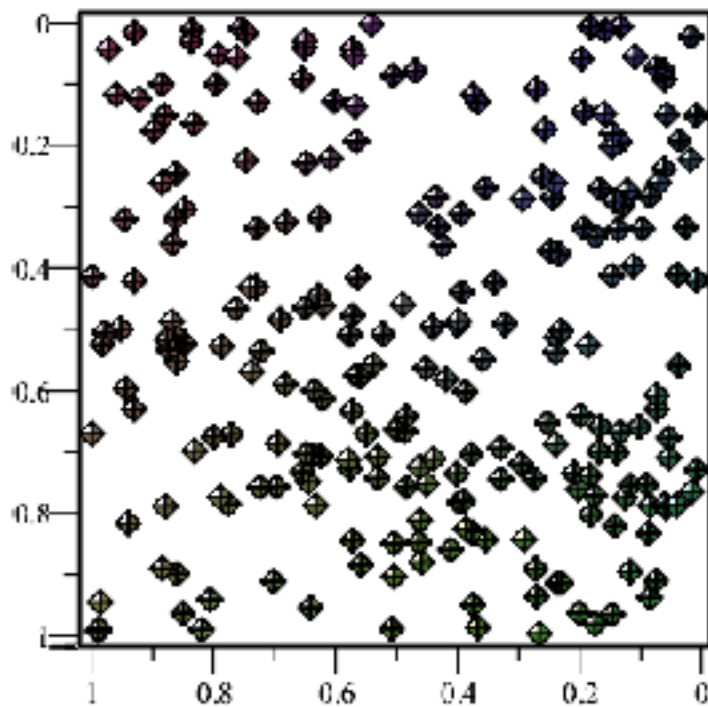


- RBF-Interpolant mit Thin-Plate-Spline (blau)



- 240 zufällige Samples

*240 Sample – Punkte, Original, RBF – Interpolant,*



# *Radiale Basis Funktionen* mit **(linearer) Präzision**

- Ziel: Lineare Funktionen werden exakt rekonstruiert

- Ansatz: 
$$f(x, y) = a + bx + cy + \sum_{i=1}^n d_i h_i(x, y)$$

- Interpolationsbedingungen + verschwindende Momente

$$\sum_j \left( \sum_{i=1}^n d_i h_i(x_j, y_j) \right) = 0$$

Moment 0.-ter Ordnung

$$\sum_j \left( \sum_{i=1}^n d_i h_i(x_j, y_j) \right) x_j = 0$$

Moment 1.-ter Ordnung

$$\sum_j \left( \sum_{i=1}^n d_i h_i(x_j, y_j) \right) y_j = 0$$

- „ -

- Verallgemeinerung: polynomielle Präzision

## Radiale Basis Funktionen mit (linearer) Präzision

$$f(x, y) = a + bx + cy + \sum_{i=1}^n d_i h_i(x, y)$$

- das resultierende  $(n+3) \times (n+3)$  – Gleichungssystem:

$$\begin{pmatrix} 0 & 0 & 0 & \sum_j h_1(x_j, y_j) & \sum_j h_2(x_j, y_j) & \cdots & \sum_j h_n(x_j, y_j) \\ 0 & 0 & 0 & \sum_j x_j h_1(x_j, y_j) & \sum_j x_j h_2(x_j, y_j) & \cdots & \sum_j x_j h_n(x_j, y_j) \\ 0 & 0 & 0 & \sum_j y_j h_1(x_j, y_j) & \sum_j y_j h_2(x_j, y_j) & \cdots & \sum_j y_j h_n(x_j, y_j) \\ 1 & x_1 & y_1 & h_1(x_1, y_1) & h_2(x_1, y_1) & \cdots & h_n(x_1, y_1) \\ 1 & x_2 & y_2 & h_1(x_2, y_2) & h_2(x_2, y_2) & \cdots & h_n(x_2, y_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & y_n & h_1(x_n, y_n) & h_2(x_n, y_n) & \cdots & h_n(x_n, y_n) \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$$



- diskrete Strukturen beinhalten **Geometrie** und **Topologie**
  - Gitter, Datenstruktur: shared vertex
- Lokale Verfahren (zellenweise interpolieren)
  - im Rechteck: bilineare Interpolation
  - im Dreieck: lineare Interpolation (mit baryzentrischen Koordinaten)
- Baryzentrische Koordinaten
  - Definition
  - geometrische Charakterisierungen
- Globale Interpolation
  - Polynominterpolation ist problematisch  
(außer rechteckiges Array von Stützstellen → Tensorprodukt)
  - RBF: Radiale Basis-Funktionen  
(Multi-Quadrik, inv. Multi-Quadrik, Thin-Plate-Spline)
  - RBF mit linearer Präzision