RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT BONN

INSTITUT FÜR INFORMATIK DER UNIVERSITÄT BONN

Einführung in die Computergrafik

Kapitel 18: Parametrische Flächen

Prof. Dr. Matthias Hullin

Institut für Informatik
Abteilung 2: Visual Computing
Universität Bonn

10. Juli 2020

Analog zu den parametrischen Kurven kann man parametrische Flächen definieren:

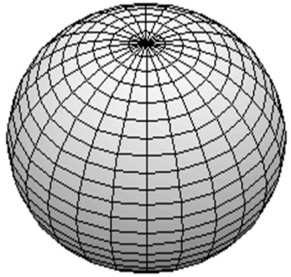
Definition (Parametrische Fläche)

Eine *parametrische Fläche* ist eine *glatte Abbildung*

$$q : D \to \mathbb{R}^n$$

eines zweidimensionalen Intervalls $D \subseteq \mathbb{R}^2$ in den $\mathbb{R}^n$. Die Variablen $(u, v) \mapsto q(u, v)$ heißen *Parameter* der Fläche.

Beispiel: Kugel



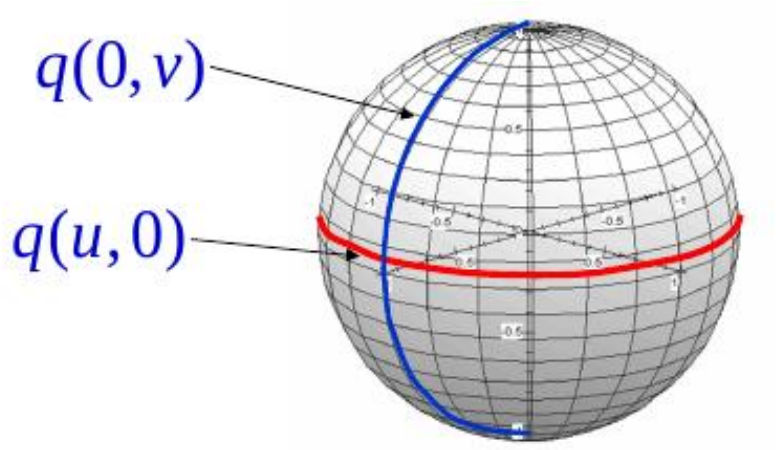
$$q(u,v) = R \cdot \begin{pmatrix} \cos(u) \cdot \cos(v) \\ \sin(u) \cdot \cos(v) \\ \sin(v) \end{pmatrix} \qquad (u,v) \in [-\pi, \pi] \times [-\pi/2, \pi/2]$$

Parametrische Flächen bestehen aus parametrischen Kurven:
Die Kurven

- $p(u) := q(u, v_0)$ für eine beliebiges aber **festes** v_0
- $p(v) := q(u_0, v)$ für eine beliebiges aber **festes** u_0

heißen *Parameterkurven der Fläche*.

Beispiel: Kugel



$$q(u, v) = R \cdot \begin{pmatrix} \cos(u) \cdot \cos(v) \\ \sin(u) \cdot \cos(v) \\ \sin(v) \end{pmatrix}$$

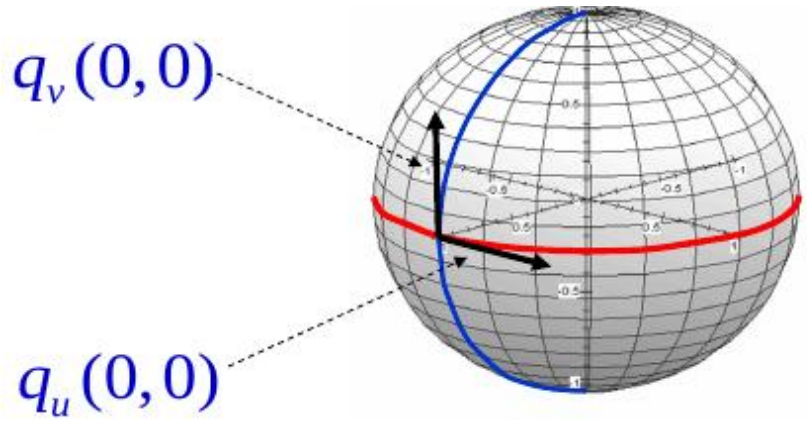
Sowohl die Stetigkeit als auch die Regularität kann analog zu den parametrischen Kurven definiert werden:

Definition (Differenzierbarkeit von parametrisierten Flächen)

Eine Fläche heißt n-*mal stetig differenzierbar*, falls die Abbildung q n-mal stetig differenzierbar ist, d.h. q n-mal stetige partielle Ableitungen besitzt.

Die Vektoren $q_u(u,v) := \frac{\partial q(u,v)}{\partial u}$ und $q_v(u,v) := \frac{\partial q(u,v)}{\partial v}$ heißen u-Tangente bzw. v-Tangente an der Stelle (u,v).

Beispiel:



Definition (Regularität von Flächen)

Eine Fläche heißt *regulär*, falls die Abbildung q einmal stetig differenzierbar ist und die Vektoren $q_u(u,v)$ und $q_v(u,v)$ für alle $(u,v) \in D$ linear unabhängig sind.

Analog zu den Kurven ist insbesondere eine Fläche mit $q_{\{u|v\}}(u,v) = 0$ nicht regulär. Für unsere Kugel:

$$q_u(u,v) = \frac{\partial}{\partial u} R \cdot \begin{pmatrix} \cos(u)\cos(v) \\ \sin(u)\cos(v) \\ \sin(v) \end{pmatrix} = R \cdot \begin{pmatrix} -\sin(u)\cos(v) \\ \cos(u)\cos(v) \\ 0 \end{pmatrix}$$

Am Nord- und Südpol ($v = \pm\pi/2$) ist $\cos(v) = 0$ und daher $q_u(u,v) = \mathbf{0}$. Die Fläche ist dort also irregulär.

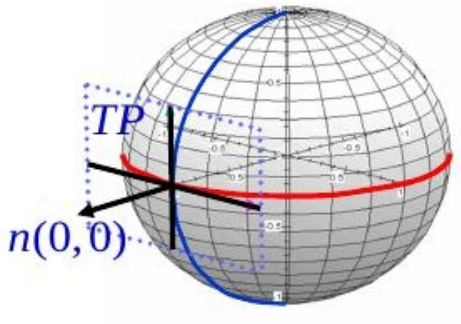
Definition (Tangentialebene)

Ist $q : D \to \mathbb{R}^n$ eine reguläre parametrisierte Fläche, so heißt die von den Vektoren $q_u(u_0, v_0)$ und $q_v(u_0, v_0)$ aufgespannte Ebene *Tangentialebene im Punkt* $q(u_0, v_0)$.

$$n(u, v) := \frac{q_u(u, v) \times q_v(u, v)}{\|q_u(u, v) \times q_v(u, v)\|_2}$$

heißt dabei *Normalen(einheits)vektor im Punkt* $q(u_0, v_0)$. Er steht senkrecht auf der Tangentialebene und ist unabhängig von der Parametrisierung.

Beispiel:



Wie bei den parametrischen Kurven suchen wir auch hier geeignete Basisfunktionen:

Definition (Tensorproduktraum)

Sind $\{F_i^m, i = 0, \ldots, m\}$ und $\{G_j^n, j = 0, \ldots, n\}$ Basen zweier Funktionsräume R_1 und R_2 mit reellwertigen univariaten Funktionen über den Intervallen I bzw. J, so bilden die bivariaten Funktionen

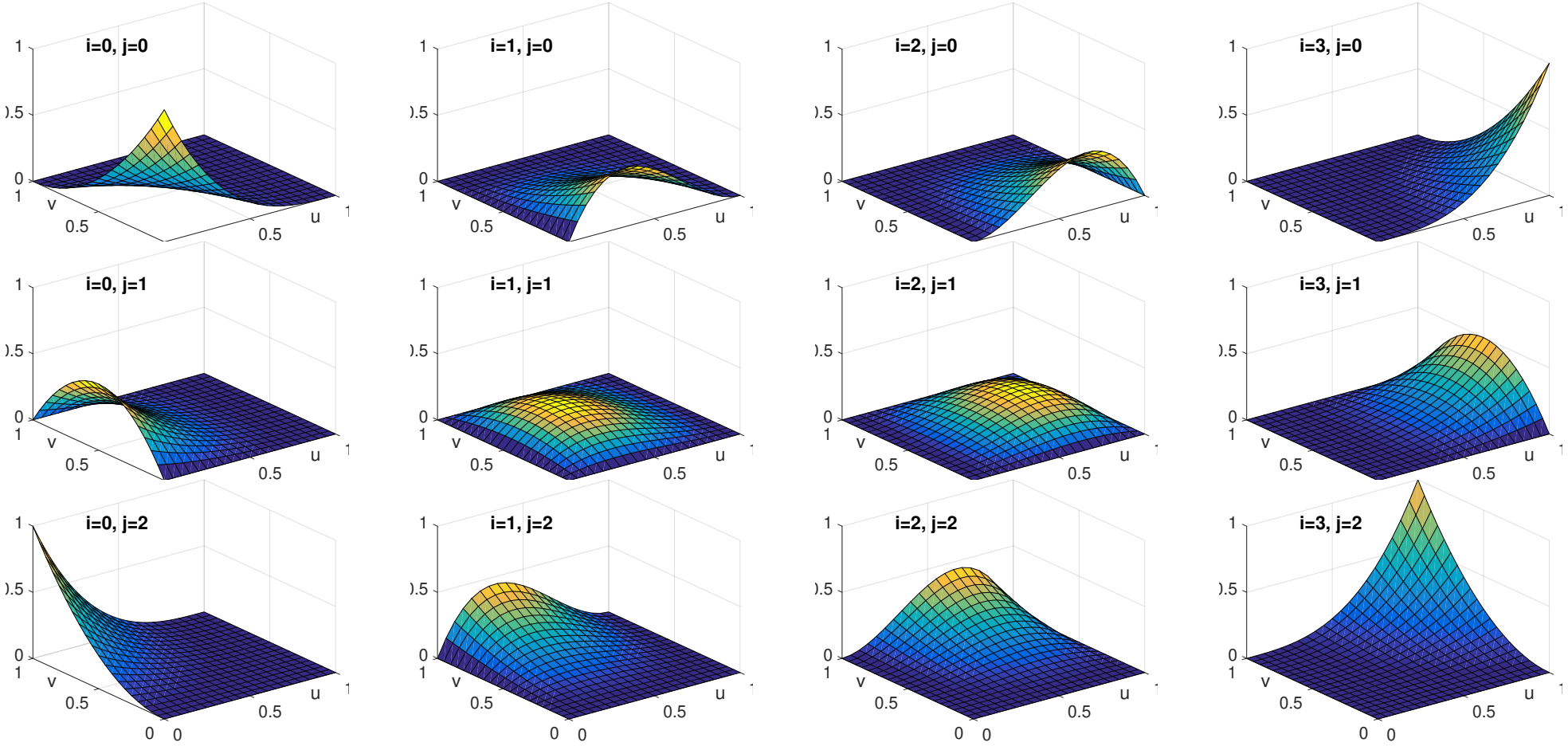
$$F_i^m G_j^n(u, v) := F_i^m(u) \cdot G_j^n(v) \qquad i = 0, \ldots m; j = 0, \ldots n \qquad (u, v) \in I \times J$$

eine Basis des *Tensorproduktraumes* $R_1 \otimes R_2$ der Dimension $(m + 1)(n + 1)$.

Beispiel: Bernsteinpolynome vom Grad m und n bilden eine Basis des Tensorproduktraumes $P^m \otimes P^n$:

$$B_i^m B_j^n(u,v) := B_i^m(u) \cdot B_j^n(v) \qquad i = 0, \ldots m; j = 0, \ldots n$$

Beispiel: $m = 3, n = 2$



Definition (Tensorproduktfläche)

Seien $F_i^m G_j^n(u,v) := F_i^m(u) \cdot G_j^n(v)\ ,\ i = 0,\dots m; j = 0,\dots n\ ,\ (u,v) \in I \times J$ eine Basis des Tensorproduktraumes $R_1 \otimes R_2$ von Funktionen über den Intervallen I bzw. J und $c_{ij} \in \mathbb{R}^d$ Koeffizienten. Dann heißt die bezüglich der Tensorproduktbasis dargestellte Funktion *Tensorproduktfläche*:

$$q(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} c_{ij} \cdot F_i^m(u) \cdot G_j^n(v)$$

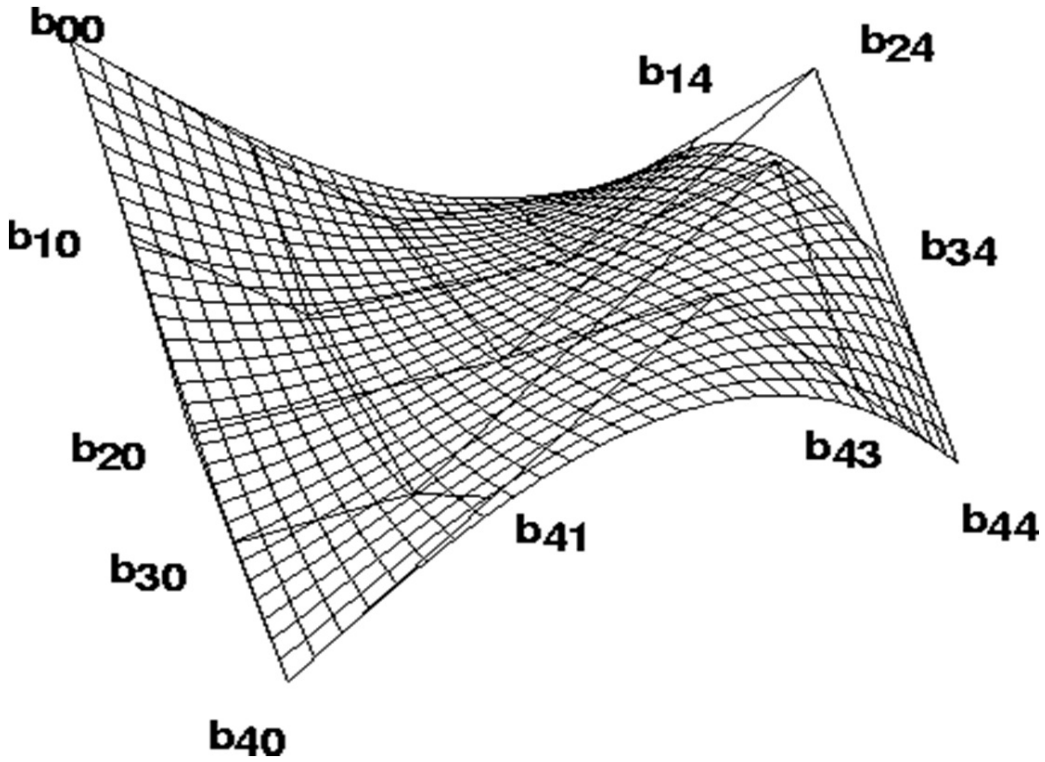
Die Tensorproduktfläche kann auch in Matrixschreibweise geschrieben werden:

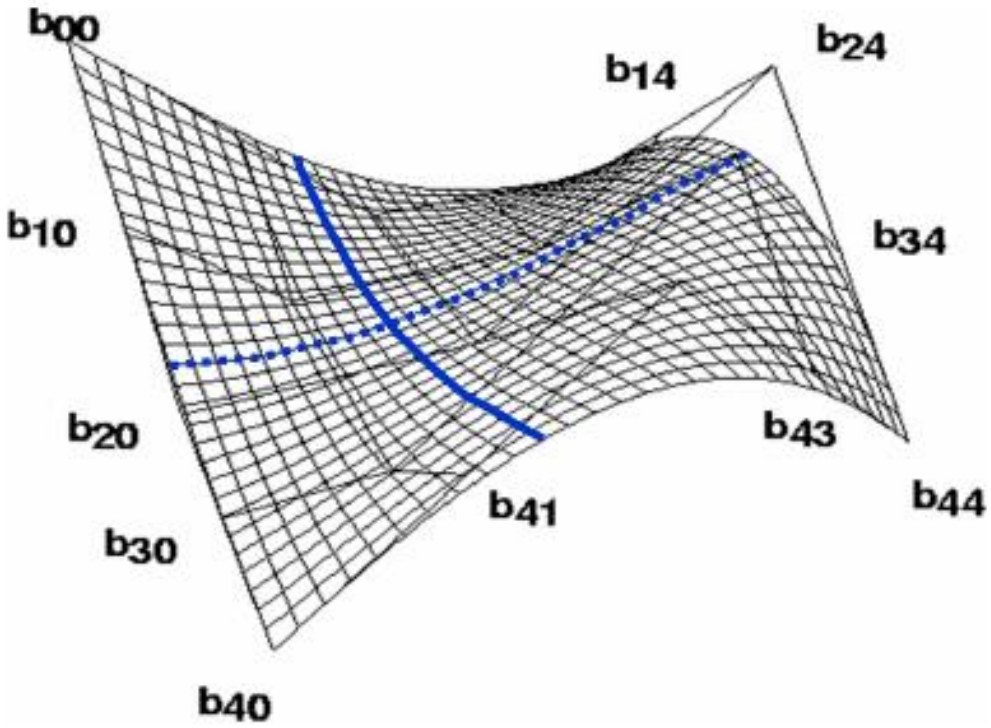
$$q(u,v) = \begin{pmatrix} F_0^m & \dots & F_m^m \end{pmatrix} \cdot \begin{pmatrix} c_{00} & \dots & c_{0n} \\ \vdots & \ddots & \vdots \\ c_{m0} & \dots & c_{mn} \end{pmatrix} \cdot \begin{pmatrix} G_0^n \\ \vdots \\ G_n^n \end{pmatrix}$$

Als Tensorproduktbasis können u. a. die Bernsteinpolynome verwendet werden:

$$q(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{ij} \cdot B_i^m(u) \cdot B_j^n(v) \qquad b_{ij} \in \mathbb{R}^d$$

Dabei sind die b_{ij} die Bézierpunkte und spannen das Kontrollnetz auf:



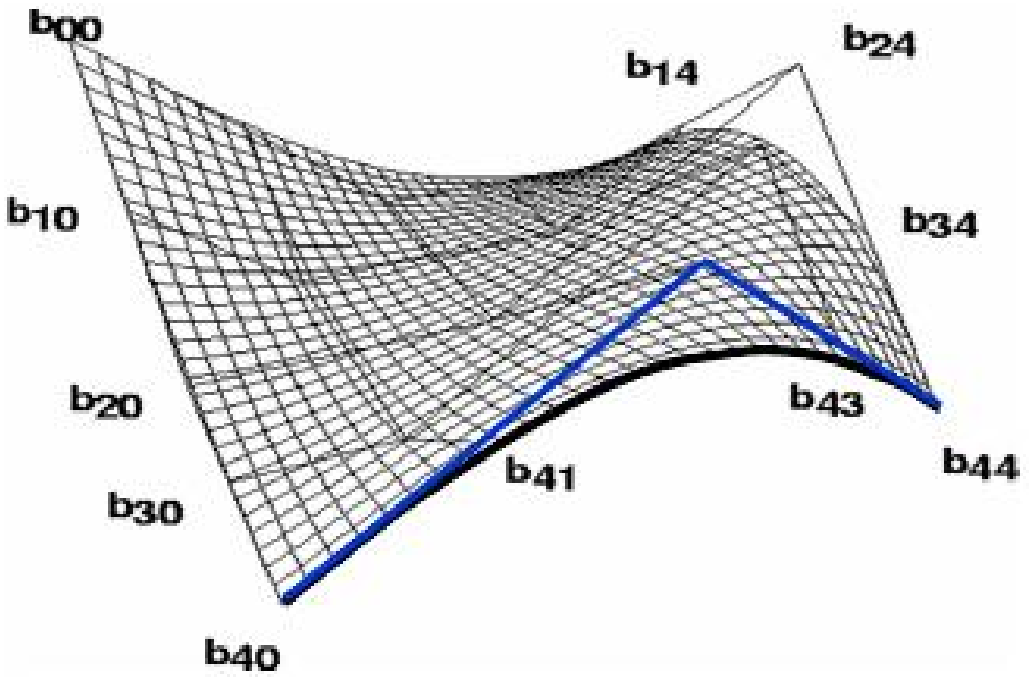


Hält man eine Dimension der Fläche fest, so liefert dies eine Bézierkurve:

$$q(u_0, v) = \sum_{i=0}^{m} \left(\sum_{j=0}^{n} b_{ij} \cdot B_i^m(u_0) \right) \cdot B_j^n(v) = \sum_{i=0}^{m} b_j(u_0) \cdot B_j^n(v)$$

Die Fläche liegt für $(u, v) \in [0, 1]^2$ in der konvexen Hülle des Kontrollpunktenetzes:

$$\sum_{i=0}^{m} \underbrace{\left(\sum_{j=0}^{n} B_i^m(u_0) \right)}_{=1} B_j^n(v) = 1$$



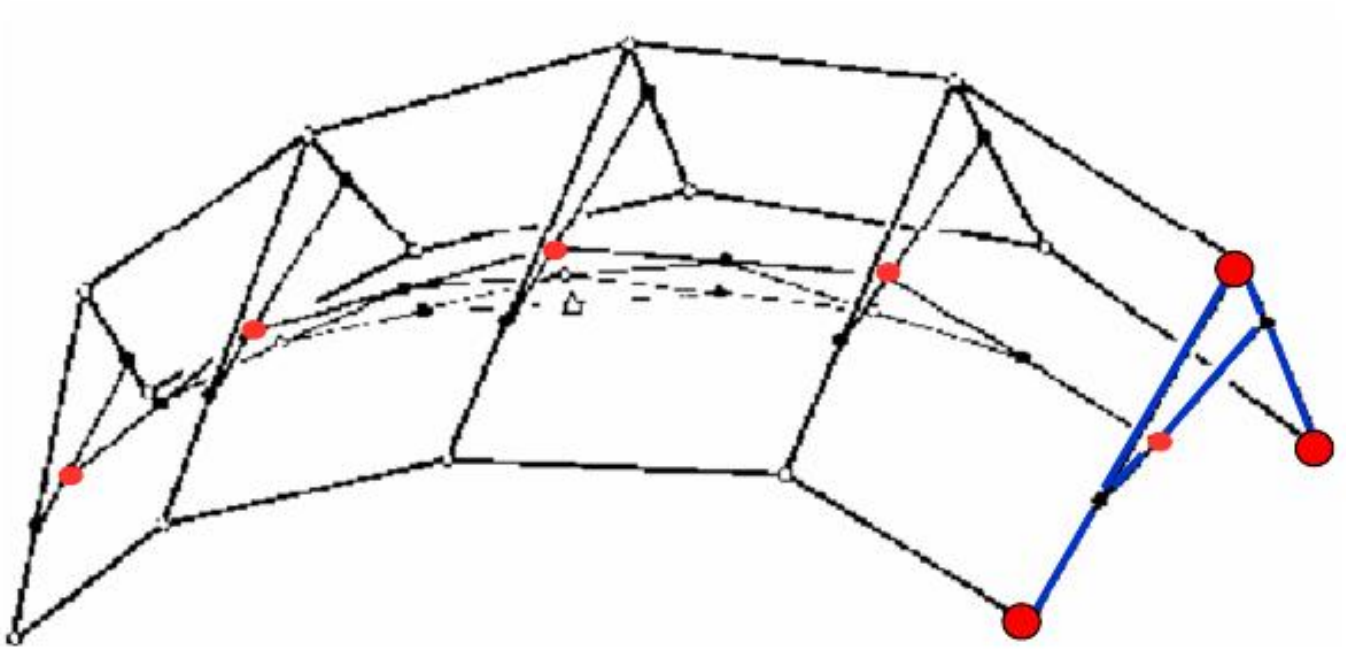
Ein Punkt $q(U_0, v_0)$ kann durch doppelte Anwendung des Algorithmus von de Casteljau rekursiv berechnet werden:

$$q(u_0, v_0) = \sum_{i=0}^{m} \underbrace{\left(\sum_{j=0}^{n} b_{ij} \cdot B_i^m(u_0) \right)}_{\text{mit de Casteljau berechnen}} \cdot B_j^n(v_0) = \underbrace{\sum_{i=0}^{m} b_j(u_0) \cdot B_j^n(v_0)}_{\text{mit de Casteljau berechnen}}$$

Ableitungen:

- Analog zu den Bézierkurven liefern die vorletzten Elemente des de Casteljau-Schemas die Richtungen der partiellen Ableitungen q_u bzw. q_v

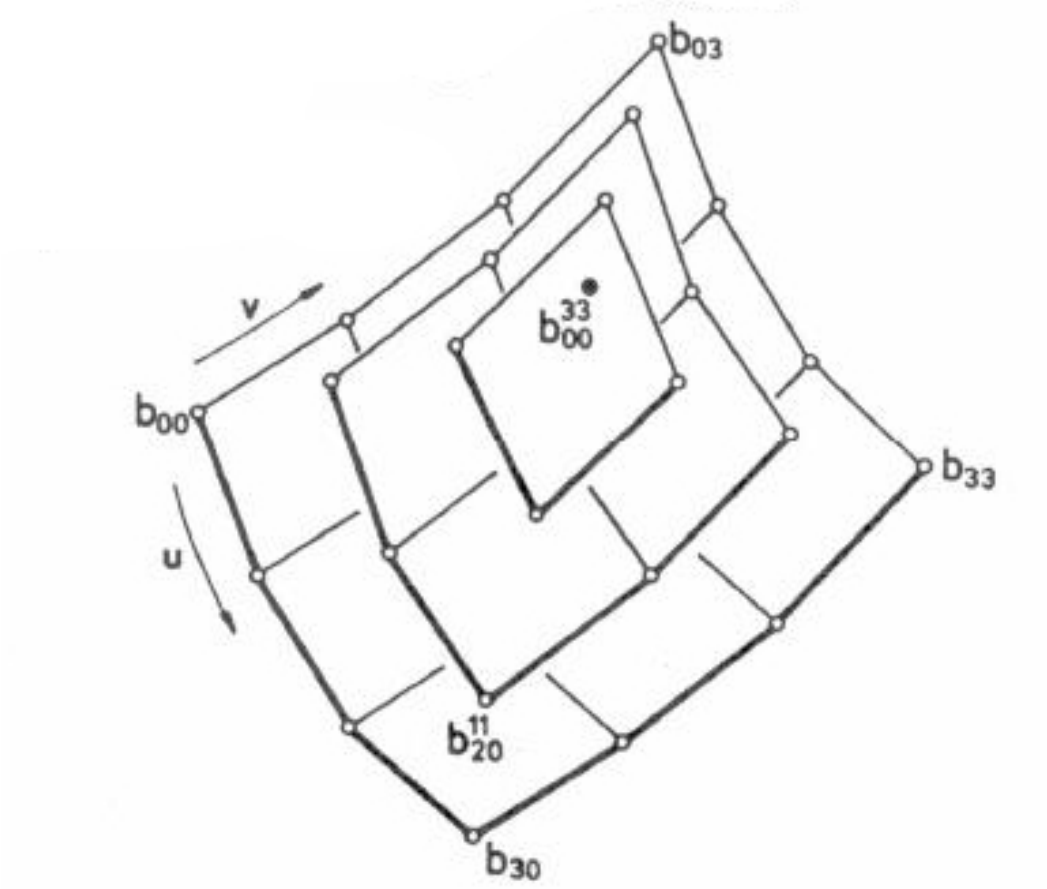
Beispiel:



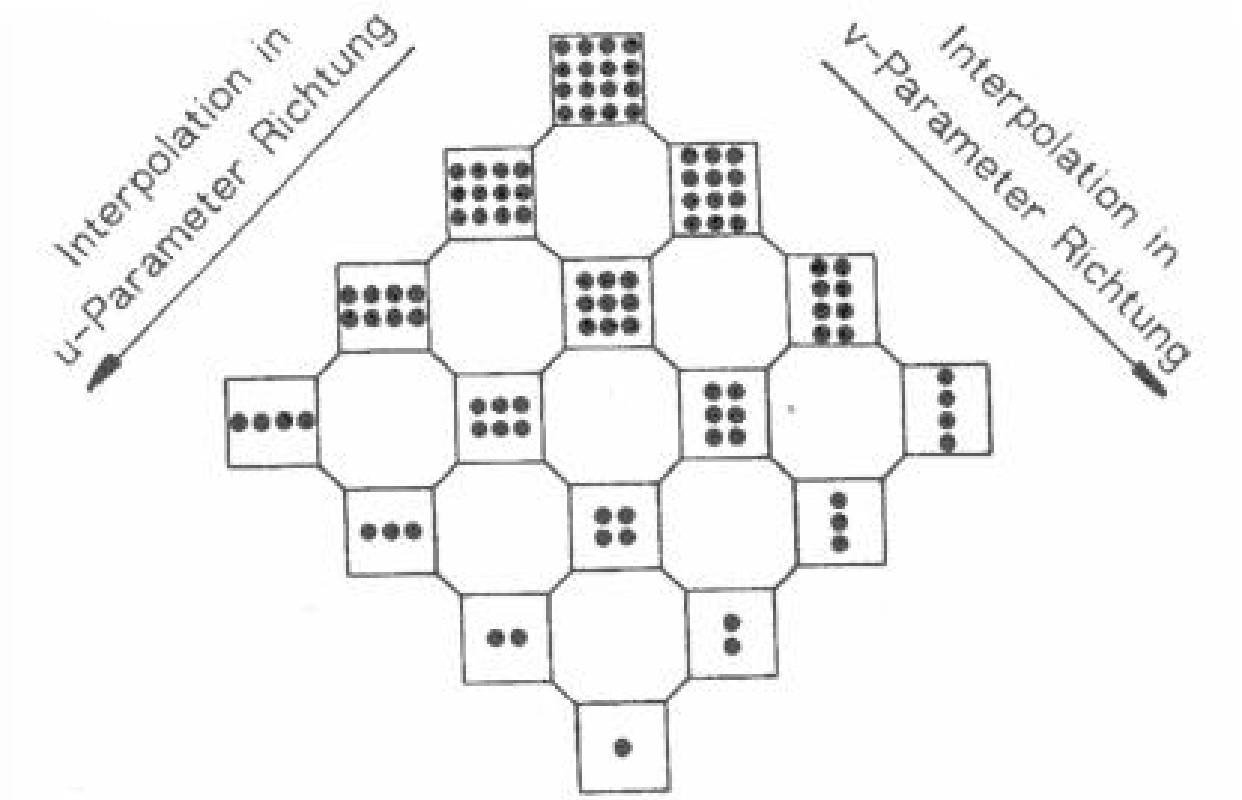
Interpolation erst in u- und dann in v-Richtung

Neben der Möglichkeit durch Hintereinanderschaltung einer Interpolation erst in u- und dann in v-Richtung besteht auch die Möglichkeit des abwechselnden Arbeitens in u- und v-Richtung.

Beispiel:



Somit lässt sich ein Berechnungsbaum aufspannen:



Anzahl der Berechnungsmöglichkeiten:

$$\frac{(m+n)!}{m! \cdot n!}$$

Ähnlich wie bei den Bézierkurven können auch Bézierflächen mit Gewichten versehen werden:

Definition (Rationale Bézierflächen)

Eine *rationale Bézierfläche* ist definiert durch

$$R(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} \tilde{b}_{ij} \cdot B_i^m(u) \cdot B_j^n(v)}{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{ij} \cdot B_i^m(u) \cdot B_j^n(v)}$$

mit Gewichten $w_{ij} \in \mathbb{R}, w_{ij} > 0, w_{n0} = 1 = w_{0m}$.

Bemerkung:

- Wegen $w_{ij} > 0$ ist der Nenner ungleich Null und R somit wohldefiniert
- Wählt man $w_i = 1 \; \forall i = 0, \ldots, n$ ist der Nenner gleich Eins und R ist eine Bézierfläche

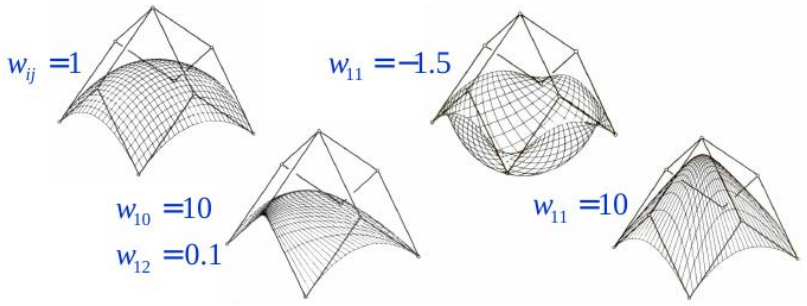
Es ist üblich, die Darstellung als Tensorproduktfläche zu bezeichnen, was aber nicht ganz korrekt ist. Die Basisfunktionen

$$B(u,v) = \frac{w_{ij} \cdot B_i^m(u) \cdot B_j^n(v)}{\sum_{i=0}^{m} \sum_{j=0}^{n} w_{ij} \cdot B_i^m(u) \cdot B_j^n(v)}$$

können aufgrund des Nenners in der Regel nicht in zwei Faktoren zerlegt werden!

Die Bezeichnung hat ihre Begründung in der Deutung als Projektion einer Tensorproduktfläche im $\mathbb{R}^4$. Daher besitzen auch rationale Bézierflächen viele Eigenschaften von Tensorproduktflächen.

Beispiele:



Die für rationale Bézierkurven besprochenen Eigenschaften übertragen sich analog zu den Standard Bézier-Tensorproduktflächen auch auf rationale Bézierflächen.

Wie bei Splines können auch Flächen stetig aneinander angeschlossen werden. Dabei gelten nahezu identische Bedingungen:

Parametrisch stetiger Anschluss (C^d-stetiger Übergang):

- Seien

$$q_1(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_{ij} \cdot B_i^m(u) \cdot B_j^n(v)$$

$$q_2(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} a_{ij} \cdot B_i^m(u) \cdot B_j^n(v)$$

  zwei stetig-differenzierbare Bézier-Tensorproduktflächen

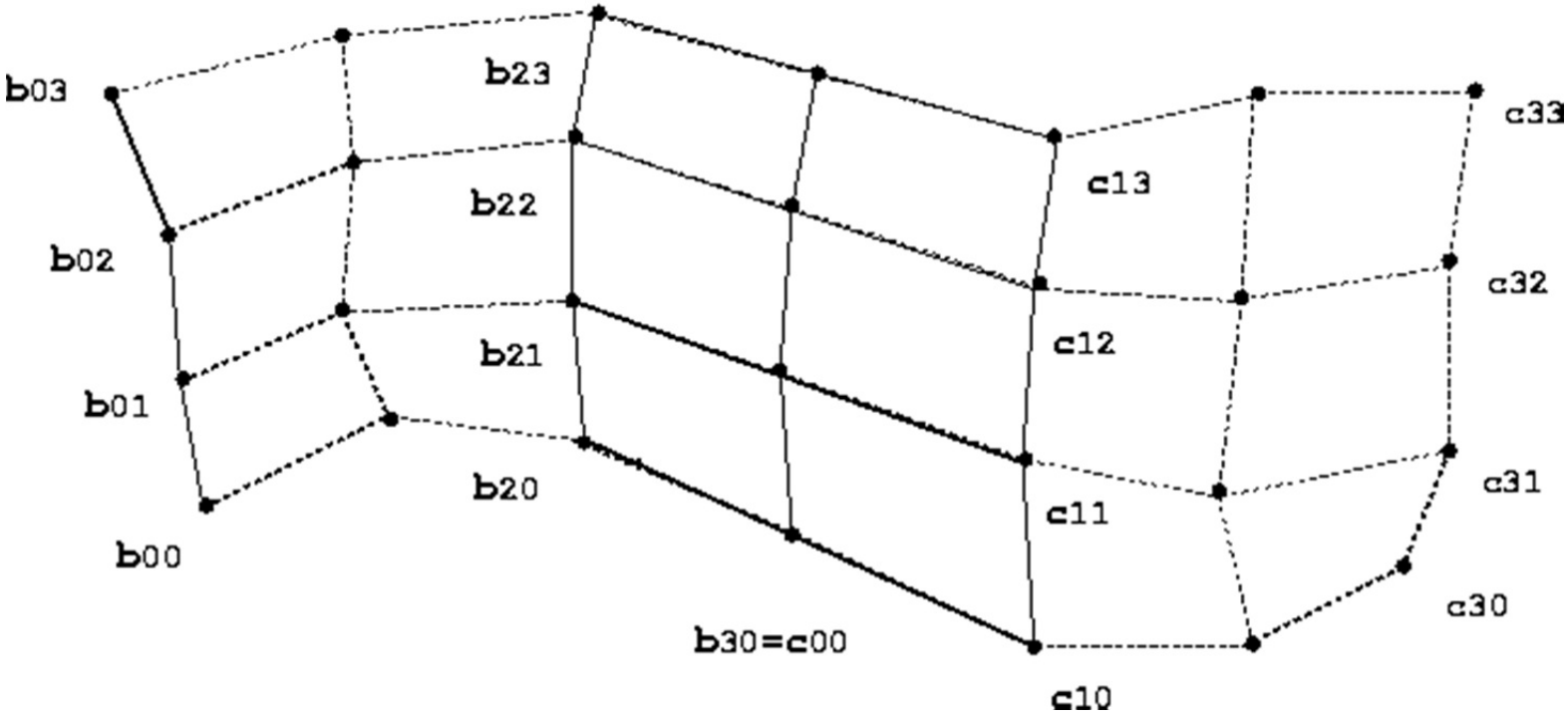
- q_1 und q_2 *schließen entlang der Randkurve C^n-stetig aneinander*, falls

$$q_1^{(k)}(b_{mj}) = q_2^{(k)}(a_{0j}) \qquad \forall k = 0,\ldots,d\ ,\ \forall j = 0,\ldots,n$$

  d.h. die Richtungen und die Länge der Ableitungen bis Ordnung d stimmen überein.

- Dies betrifft auch gemischte Ableitungen, d.h. etwa für C^2 müssen $\{\partial^2/\partial u^2, \partial^2/\partial v^2, \partial^2/\partial u \partial v\} q_{\{1,2\}}(u,v)$ zwischen den beiden Flächen übereinstimmen.

Beispiel:



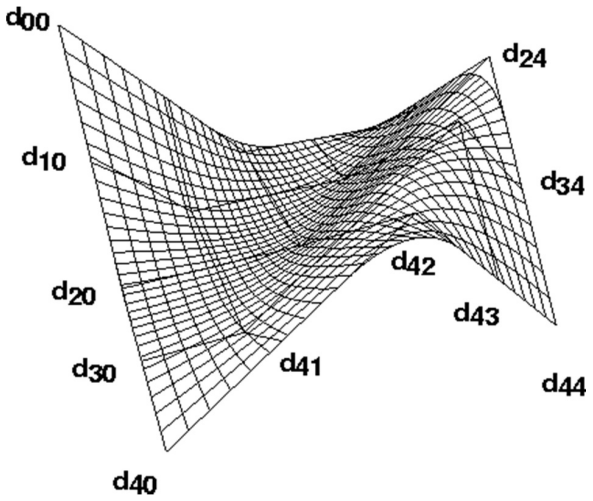
Die Ableitungen lassen sich genauso wie bei den Bézierkurven berechnen.

Auch die Idee der B-Splines kann übernommen werden:

Verwendet die B-Spline-Tensorproduktbasis über $[s_0, \ldots, s_{m+w+1}] \times [t_0, \ldots, t_{m+p+1}]$:

$$q(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} d_{ij} \cdot N_i^w(u) \cdot M_j^p(v)$$

mit *de Boor-Punkten* $d_{ij} \in \mathbb{R}^d$ und normalisierten B-Splines $N_i^n(u)$ und $M_j^p(v)$.



Normalisierte B-Splines (zur Erinnerung)
Sei $n \leq m$ und $T = (t_0 = \cdots = t_n, t_{n+1}, \ldots, t_m, t_{m+1} = \cdots = t_{m+n+1})$ eine schwach monoton wachsende Folge von Knoten mit $t_i < t_{i+n+1}, 0 \leq i \leq m$. Die rekursiv definierten Funktionen

$$N_i^0(t) := \begin{cases} 1 & \text{, falls } t_i \leq t < t_{i+1} \\ 0 & \text{, sonst} \end{cases}$$

$$N_i^k(t) := \frac{t - t_i}{t_{i+k} - t_i} \cdot N_i^{k-1}(t) + \frac{t_{i+1+k} - t}{t_{i+1+k} - t_{i+1}} \cdot N_{i+1}^{k-1}(t) \qquad 1 \leq k \leq n$$

heißen *normalisierte B-Splines* vom Grad n über T.

Wie bei Bézier-Tensorproduktflächen übertragen sich die Eigenschaften der Kurven entsprechend auf die B-Spline-Tensorproduktflächen.

Definition (Non-Uniform Rational B-Spline)

Ein *Non-Uniform Rational B-Spline* ist definiert durch

$$R(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,j}(u,v) P_{i,j}$$

mit Kontrollgitter $P_{i,j}$ und der rationalen Basisfunktion

$$R_{i,j}(u,v) = \frac{N_{i,p}(u) N_{j,q}(v) w_{i,j}}{\sum_{k=0}^{n} \sum_{l=0}^{m} N_{k,p}(u) N_{l,q}(v) w_{k,l}}$$

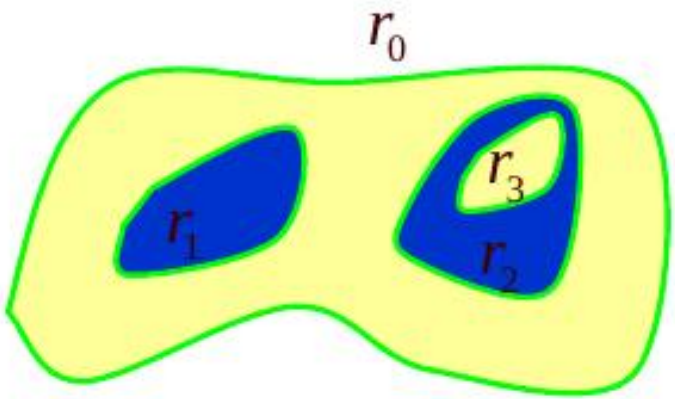
mit einem Gewicht $w_{i,j}$ für jeden Kontrollpunkt.

NURBS sind sozusagen das “volle Paket”:

- non-uniform, d.h. Knoten je Dimension beliebig positionierbar
- rational, d.h. Kugeln, Ellipsen, Tori, ... darstellbar
- B-Spline, d.h. lokale Kontrolle über Glattheit; Grad von Komplexität der Fläche entkoppelt.

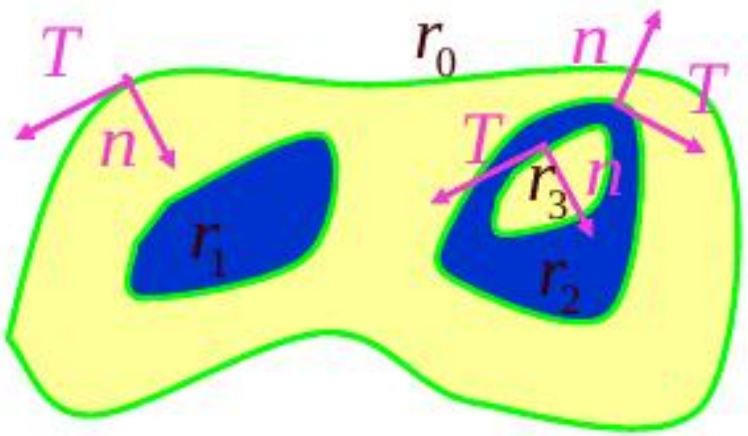
Einschränkung: bisher nur rechteckiges Parametergebiet
Lösung: Definiere getrimmtes ebenes Parametergebiet durch eine Menge von Randkurven:

$$R = \{r_0, \ldots, r_n\}$$



Dabei ist r_0 der äußere Rand und $r_1, \ldots, r_n$ sind die inneren Ränder.

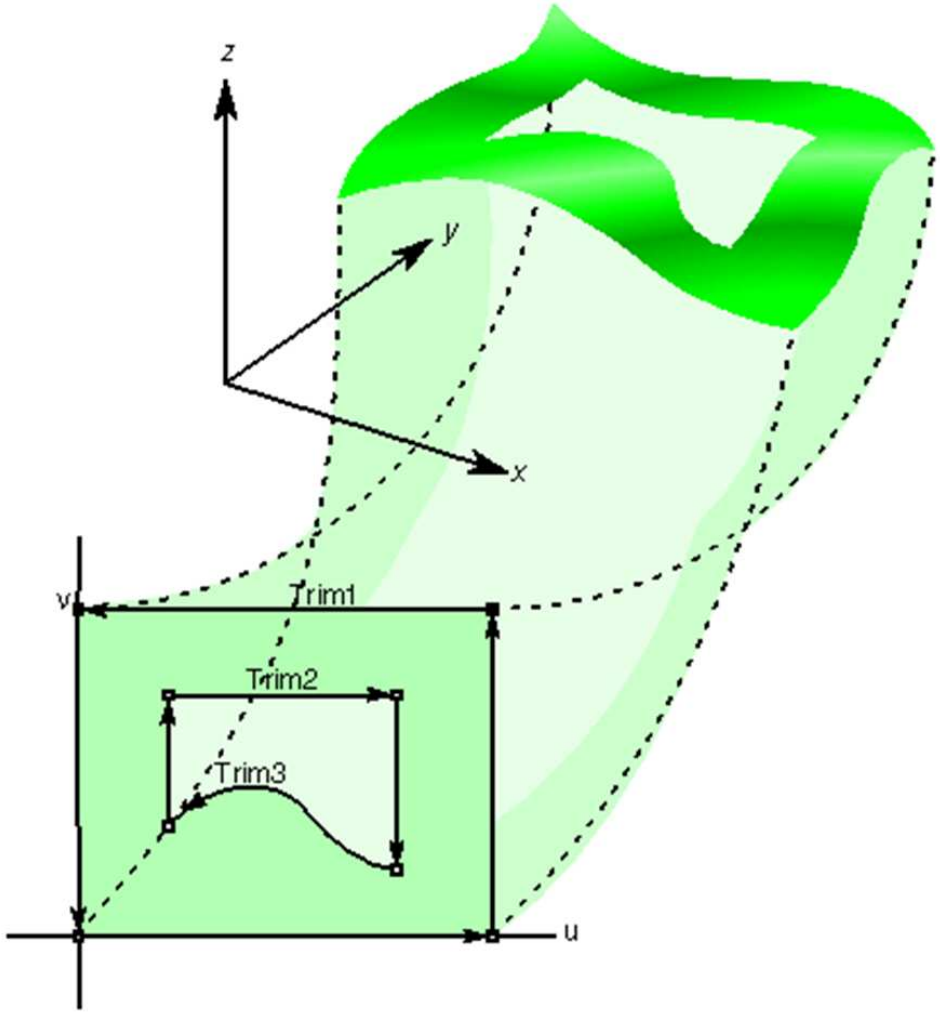
Jede Randkurve r_i besitzt ein Gebiet Ω_i. Der positive Normaleneinheitsvektor n der Randkurve r_i zeigt stets in das Innere von Ω_i:



Bedingungen der Randkurven:

- jede Randkurve ist geschlossen
- Randkurven schneiden sich nicht
- Geschachtelte Randkurven sind konsistent orientiert, d.h. ist r_i Vater von r_j, so sind r_i und r_j gegenläufig orientiert

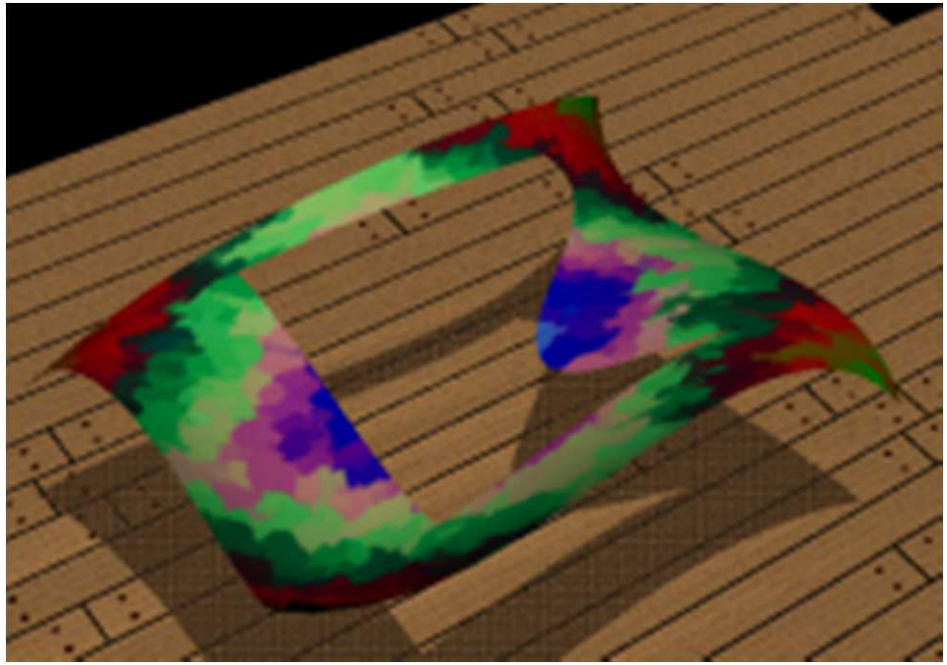
Die Trimming-Kurven im Parametergebiet werden auf die Fläche abgebildet:



Solche Kurven konnten einst auch direkt in OpenGL erzeugt werden

(letzte auffindbare Erwähnung in einem Referenzhandbuch zu OpenGL 2.1):

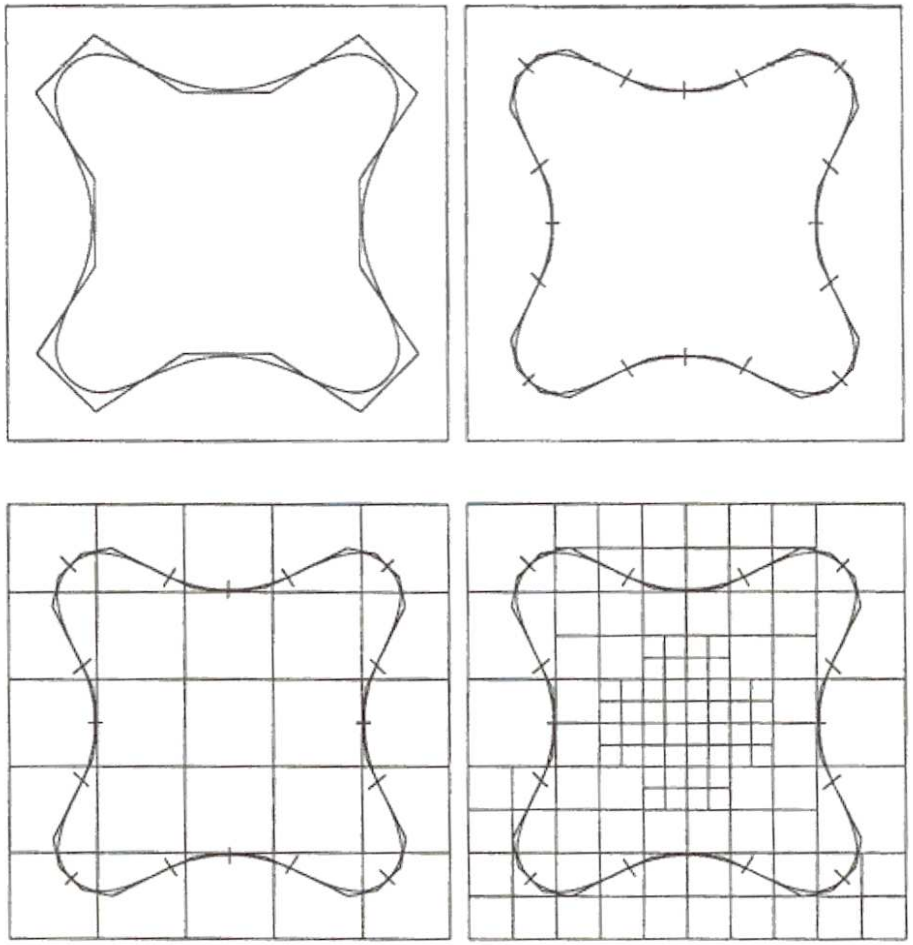
```
1: gluBeginSurface();
2:     gluNurbsSurface(...);
3:     gluBeginTrim();
4:         gluPwlCurve(...);
5:     gluEndTrim();
6:     gluBeginTrim();
7:         gluNurbsCurve(...);
8:     gluEndTrim();
9: gluEndSurface();
```



Problem: Wie approximiert man die Fläche durch Dreiecke?

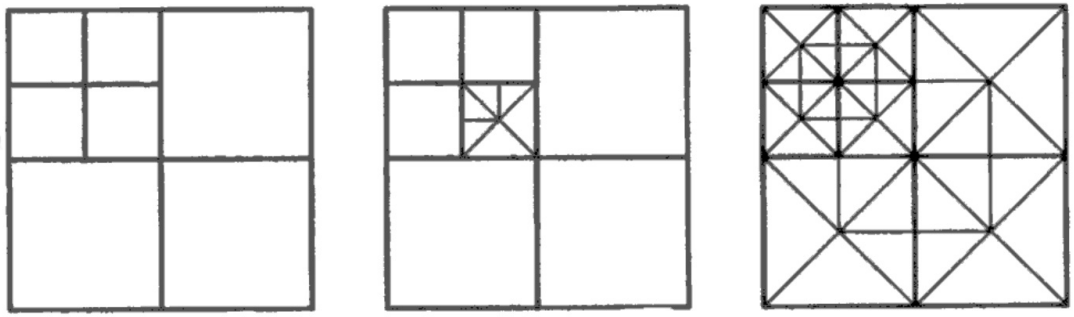
Übersicht:

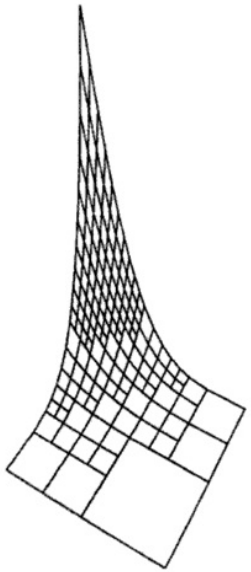
- Konvertierung der NURBS-Trimmingkurven und NURBS-Flächen in Bézierdarstellung
- Adaptive Unterteilung der Bézierflächen mit kontrolliertem parametrischen Fehler
- Fehlerkontrollierte Approximation der Trimmingkurven im 3D



Adaptive Unterteilung der Bézierflächen mit kontrolliertem parametrischen Fehler:

- Zur Unterteilung wird ein eingeschränkter Quadtree erzeugt: Die Unterteilungstiefe benachbarter Zellen unterscheidet sich um nicht mehr als eins

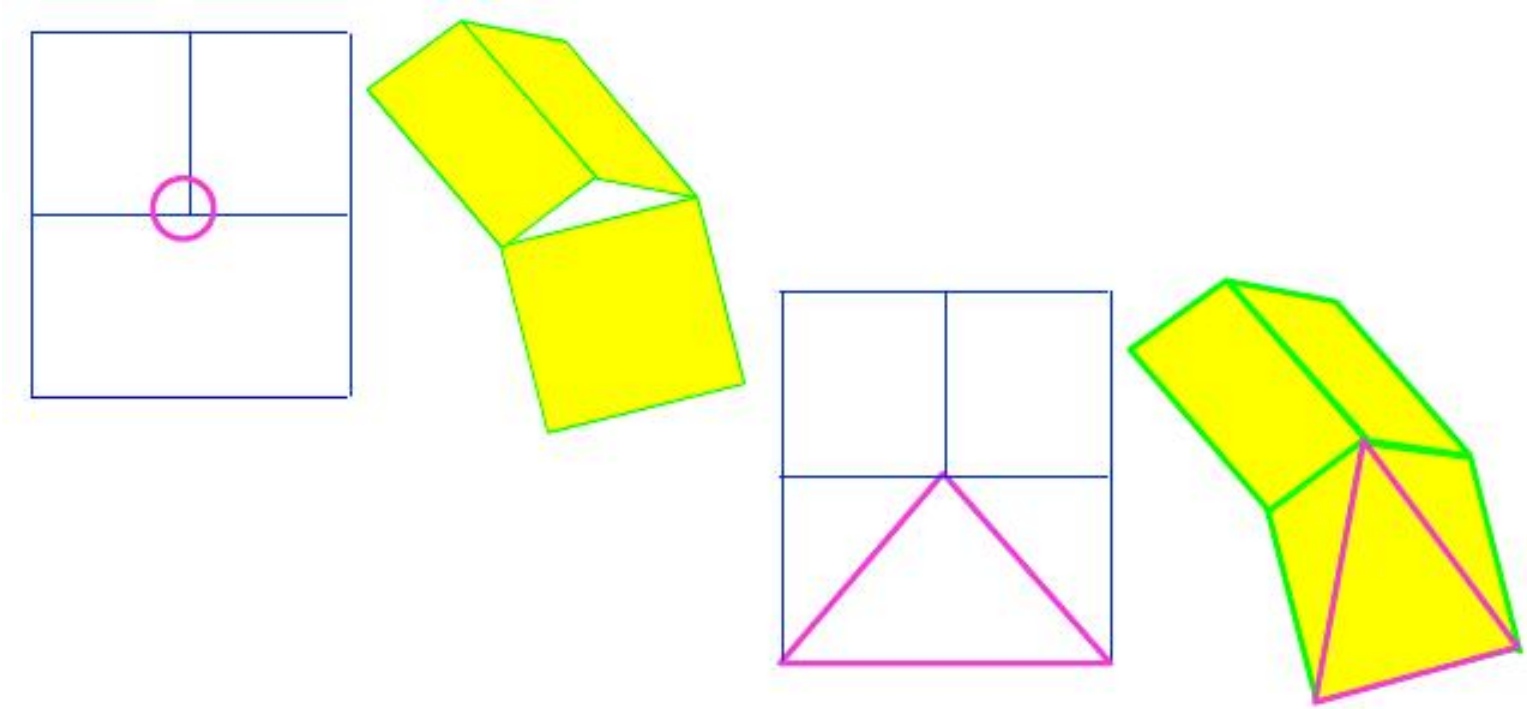




- **Start** : In Bézierflächen konvertierte NURBS-Flächen
- **Triangulierung (Lookup-Tabelle)** : Jede Zelle wird in 4–8 Dreiecke zerlegt. Jede Kante hat 2 Nachbardreiecke, außer die Nachbarzelle ist nicht so oft unterteilt – dann wird nur ein Dreieck erzeugt
- **Rekursion** : Solange der Approximationsfehler größer als ein Schwellwert ist, führe Mittelpunktsunterteilung der Bézierfläche durch

Adaptive Unterteilung der Bézierflächen mit kontrolliertem parametrischen Fehler:

Achtung bei Quadtrees : T-Vertices führen zu Artefakten bei der Interpolation und müssen vermieden werden!



Lemma (Fehlerkontrolliertes Triangulieren (1))

Wird eine bilineare Interpolationsfläche

$$q(u,v) = \sum_{i=0}^{1} \sum_{j=0}^{1} b_{ij} \cdot B_i^1(u) \cdot B_j^1(v)$$

über $[0,1]^2$ *durch stückweise lineare Funktionen über den Dreiecken*

$$\Delta((0,0),(1,0),(1,1)) \qquad \textit{und} \qquad \Delta((0,0),(1,1),(0,1))$$

bzw.

$$\Delta((0,0),(1,0),(0,1)) \qquad \textit{und} \qquad \Delta((1,0),(1,1),(0,1))$$

approximiert, so ergibt sich unabhängig von der Wahl der Triangulierung ein Fehler

$$\epsilon = \frac{1}{4} \left\| b_{00} - b_{01} + b_{11} - b_{10} \right\|_2$$

Lemma (Fehlerkontrolliertes Triangulieren (2))

Der parametrische Abstand zwischen einer Bézier-Tensorproduktfläche

$$q(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{ij} \cdot B_i^m(u) \cdot B_j^n(v)$$

und einer bilinearen Interpolationsfläche g der Eckpunkte $b_{00}, b_{0n}, b_{m0}, b_{mn}$ kann wie folgt abgeschätzt werden:

$$\sup_{(u,v)\in[0,1]^2} \|q(u,v) - g(u,v)\|_2 \leq \max \|c_{ij}\|_2$$

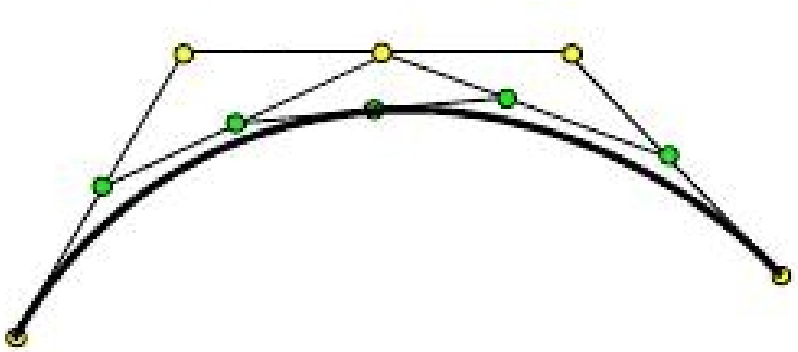
mit

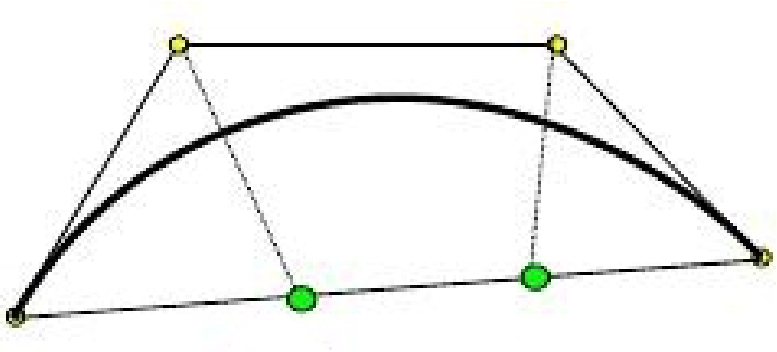
$$c_{ij} = b_{ij} - \left(\frac{(m-i)(n-j)}{mn} b_{00} + \frac{(m-i)j}{mn} b_{0n} + \frac{i(n-j)}{mn} b_{m0} + \frac{ij}{mn} b_{mn} \right)$$

Fehlerkontrollierte Approximation der Trimmingkurven im 3D:

Adaptive Unterteilung einer Bézierkurve:

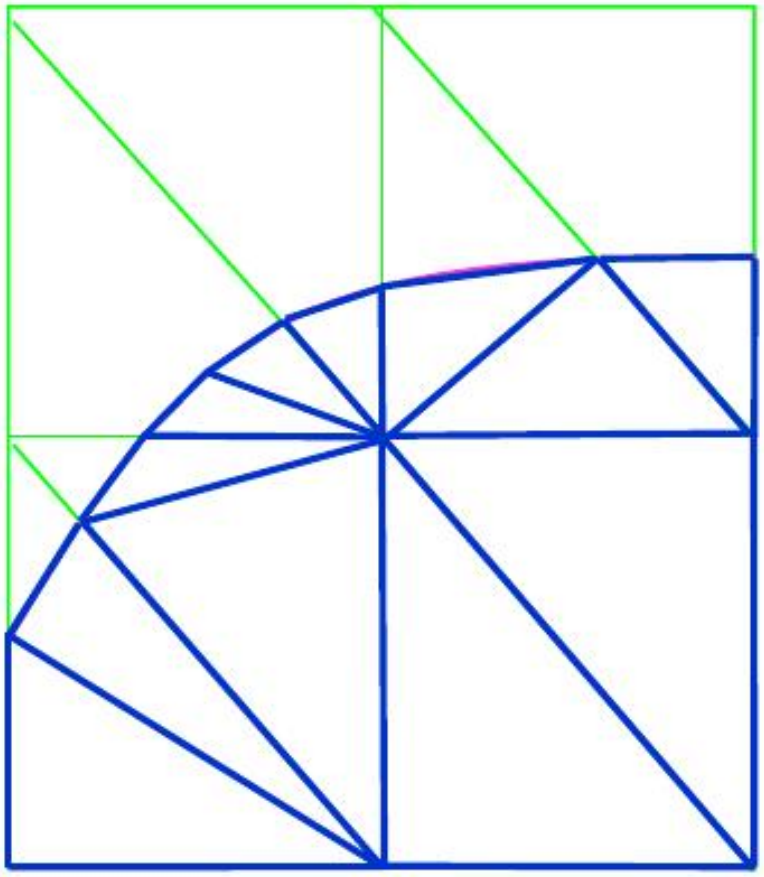
- Mittelpunktsunterteilung mit Algorithmus von de Casteljau



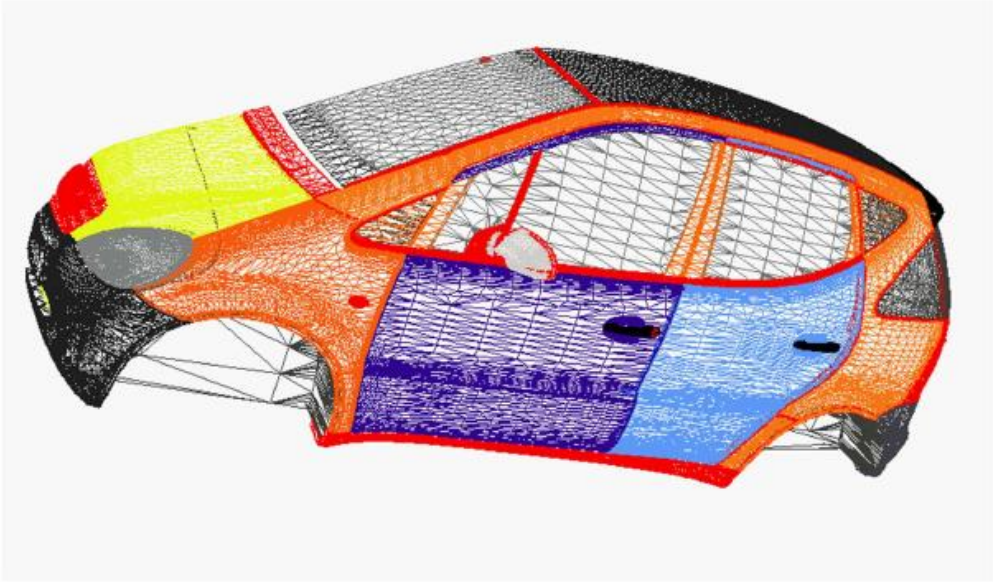


Einfügen von Trimmingkurven:

Nach der Triangulierung des adaptiven Quadtrees werden alle Dreiecke, die von der Trimmingkurve geschnitten werden, entsprechend trianguliert.



Beispiel:



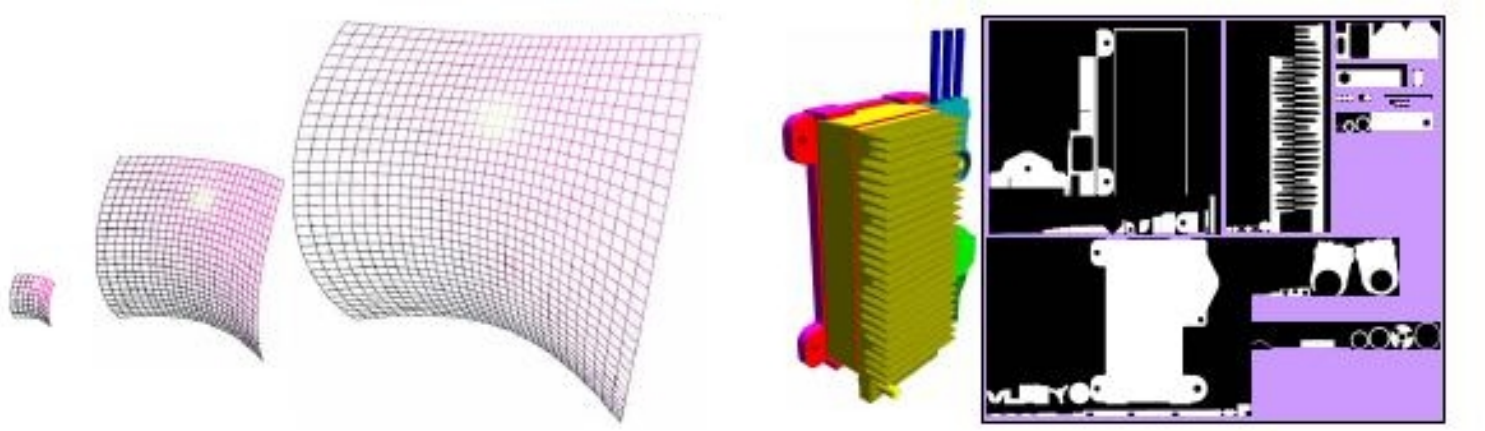
~96.000 Dreiecke

Fazit:

+ Ergebnis ist ein Dreiecksnetz, das beliebig weiterverarbeitet werden kann
- Bei zu wenigen Dreiecken werden die Flächen kantig, bei zu vielen steigt die Rechenzeit
- Triangulierung ist relativ zeitaufwendig, kann also nicht während des Renderings gemacht werden

Um obiges Verfahren hardwarebeschleunigt zu implementieren, werden folgende Grundideen verwendet:

- Verwende uniformes Gitter statt Quadtree und berechne Flächenpunkte in der Vertexeinheit
- Rasterisiere die Trimmingkurven in eine Textur und verwende diese für das Trimmen in der Fragmenteinheit



Der Fehler einer Approximation einer Fläche q durch ein trianguliertes uniformes Gitter g mit Schrittweiten Δu und Δv ist beschränkt durch:

$$\begin{aligned}\sup_{(a,b)\in[0,1]^2} \|q(a,b) - g(a,b)\|_2 &\leq \frac{1}{8}(\Delta u^2 D_{u^2} + \Delta u \Delta v D_{uv} + \Delta v^2 D_{v^2}) \\ &\leq \underbrace{\frac{1}{8}\Delta u^2 (D_{u^2} + D_{uv})}_{\delta_u} + \underbrace{\frac{1}{8}\Delta v^2 (D_{v^2} + D_{uv})}_{\delta_v}\end{aligned}$$

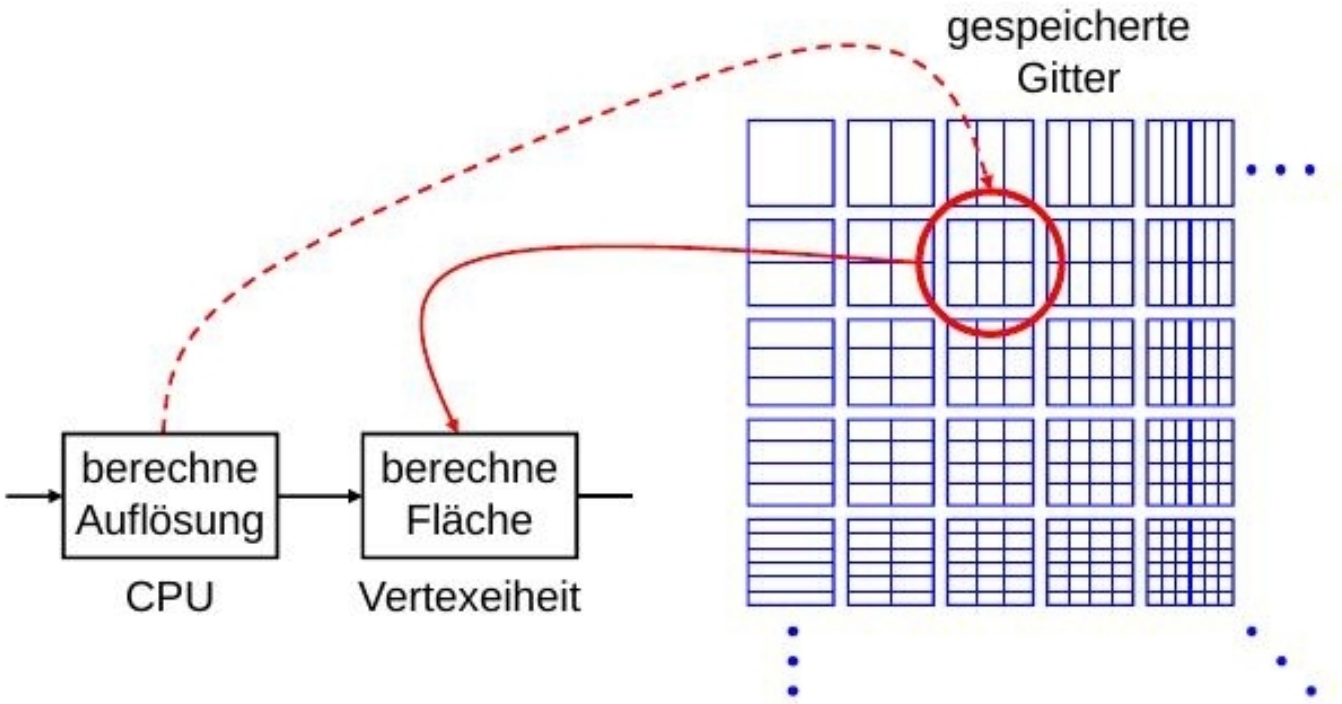
mit

$$D_{u^2} = \sup_{(a,b)\in[0,1]^2} \left\| \frac{\partial^2 q}{\partial u^2} \right\|_2 \quad D_{v^2} = \sup_{(a,b)\in[0,1]^2} \left\| \frac{\partial^2 q}{\partial v^2} \right\|_2 \quad D_{uv} = \sup_{(a,b)\in[0,1]^2} \left\| \frac{\partial^2 q}{\partial u \partial v} \right\|_2$$

Wähle nun die Schrittweiten Δu und Δv so dass

$$\delta_u, \delta_v \leq \frac{1}{2}\epsilon$$

Die Schrittweiten werden aber nicht beliebig gewählt, sondern sie werden anhand von vordefinierten regulären Gittern ausgewählt:



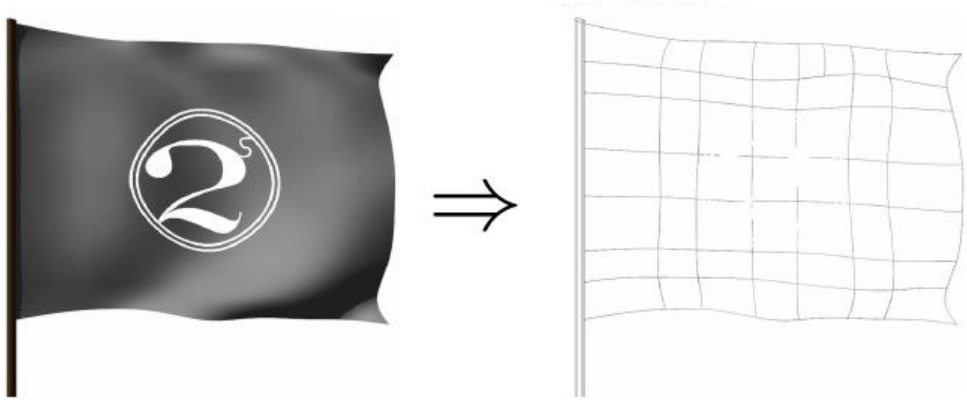
Die Schrittweitenberechnung und die Auswahl werden dabei auf der CPU berechnet.

Ebenso wie die Schrittweitenberechnung wird auch die Umwandlung von NURBS- in Bézierflächen auf der CPU berechnet:

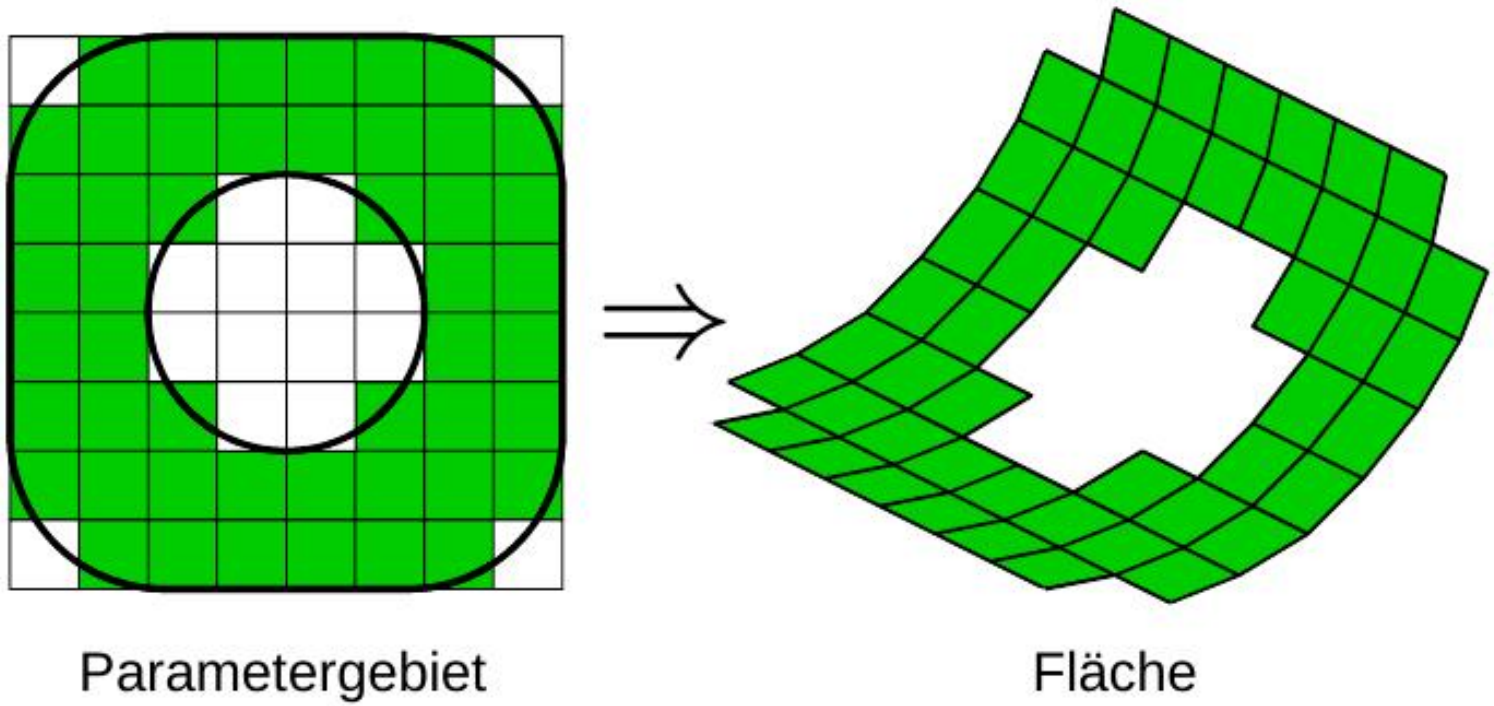
- Gitter enthalten $(u, v) \in [0, 1]^2$ als Parameter
- Algorithmus speziell für bikubische Bézierflächen entwickelt
- daher müssen Flächen höheren Grades durch bikubische Flächen approximiert werden
- Auswertung von Flächenpunkt und ersten Ableitungen mit Algorithmus von de Casteljau

Bikubische Approximation:

- Möglichst effizienter Algorithmus $\rightarrow$ keine Fallunterscheidungen
- Möglichst viele Grafikkarten unterstützen $\rightarrow$ maximal bikubische rationale Bézierflächen



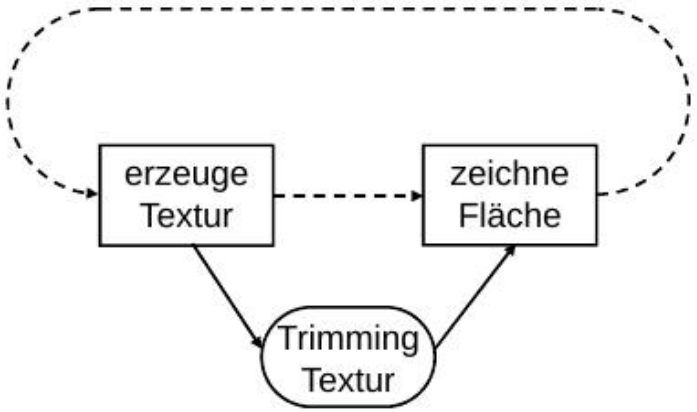
Idee: Erzeuge Textur, die Trimming im Parametergebiet enthält:



Die Trimming Textur kann

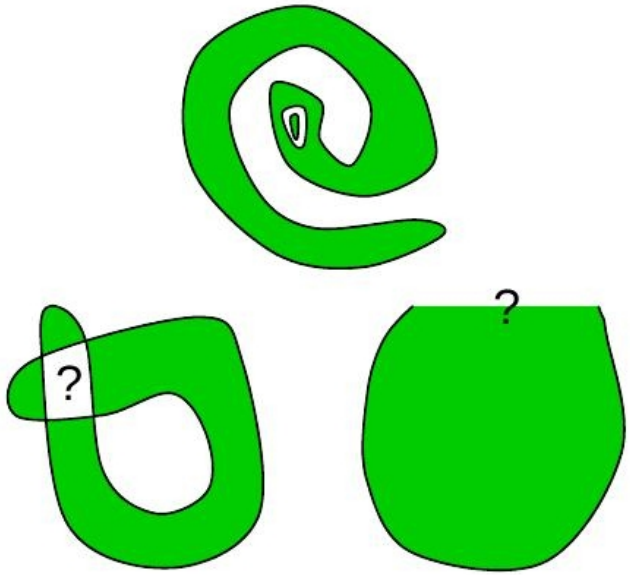
- angepasst an den Approximationsfehler auf dem Bildschirm (x Pixel)
- und während des Renderns

erzeugt werden.

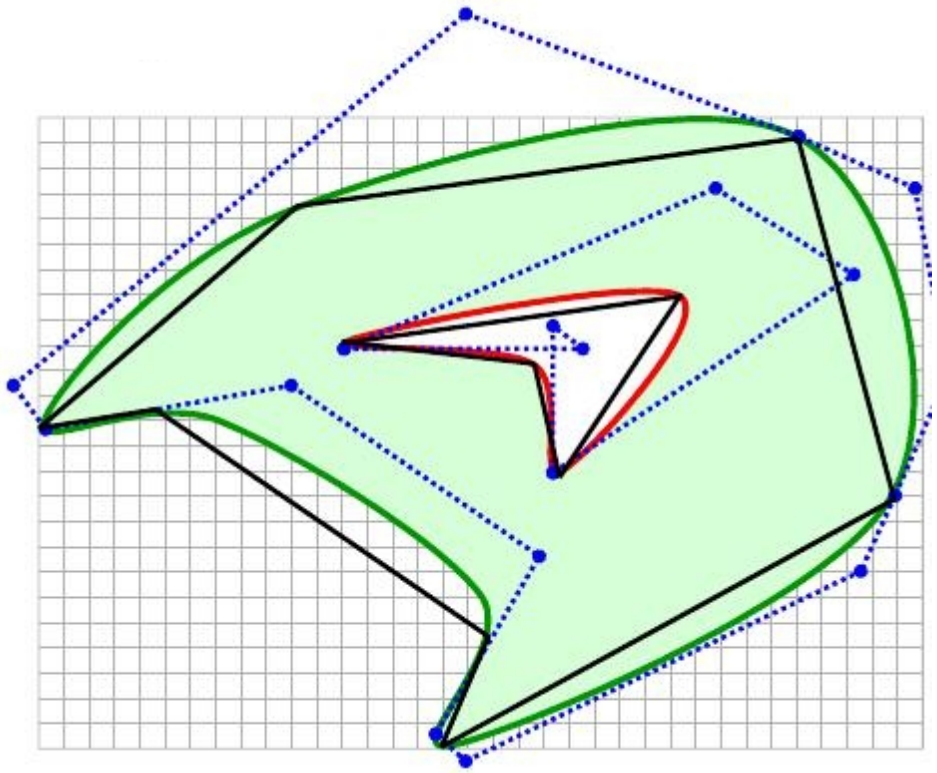


Probleme:

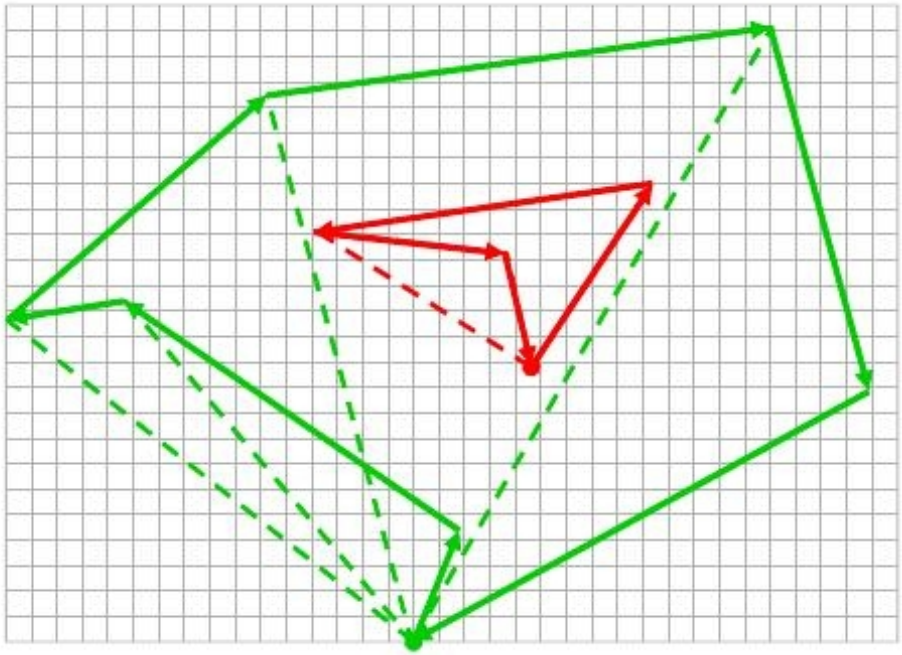
- Verschachtlung
- Überschneidungen
- Offene Schleifen



1. Approximiere Trimmingkurven



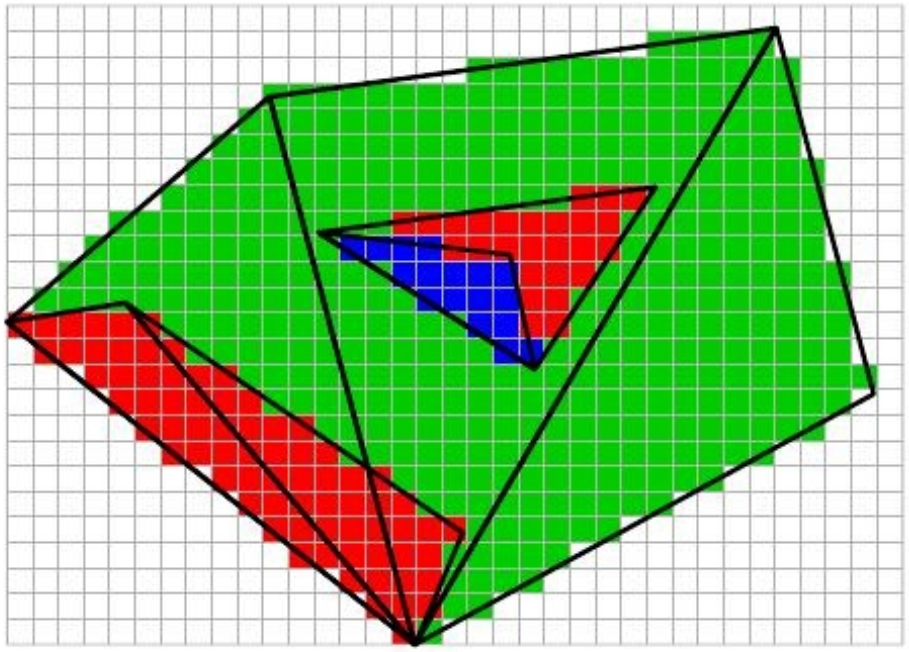
1. Approximiere Trimmingkurven
2. Erzeuge Dreiecksnetz für jede Schleife



Hardwarebeschleunigung: Texturbasiertes Trimming

1. Approximiere Trimmingkurven
2. Erzeuge Dreiecksnetz für jede Schleife
3. Anzahl der Überdeckungen bestimmt ob Dreieck innen / außen ist

1 2 3



Hardwarebeschleunigung: Texturbasiertes Trimming

1. Approximiere Trimmingkurven
2. Erzeuge Dreiecksnetz für jede Schleife
3. Anzahl der Überdeckungen bestimmt ob Dreieck innen / außen ist
   $\rightarrow$ Umschalten statt Zählen