# „Direct Construction of Polynomial Surfaces from Dense Range Images through Region Growing“

Tino Schwarze

4. Februar 2000


## Zusammenfassung

Bisherige Techniken zur Approximation diskreter Daten durch glatte Flächen basieren entweder auf Kurvenanpassung oder „edge detection“. Sie benötigen meist die Erfahrung des Anwenders, um brauchbare Ergebnisse zu erzielen.

„Region Growing“ approximiert eine Menge von 3D-Punkten durch eine Fläche $z = z(x, y)$ und versucht, die Datenmenge pro Teilfläche bei minimaler Interaktion mit dem Anwender zu maximieren.


TU-Chemnitz: Hauptseminar „Rekonstruktive Computergrafik“

# Einführung / Vorbereitung 1

- Ausgangspunkt ist ein „Dense Range Image“, DRI[1]
- die Daten liegen in Form von $(x, y)$-Koordinaten und zugehöriger „Höhe“ $z$ vor (siehe Abb. 1)

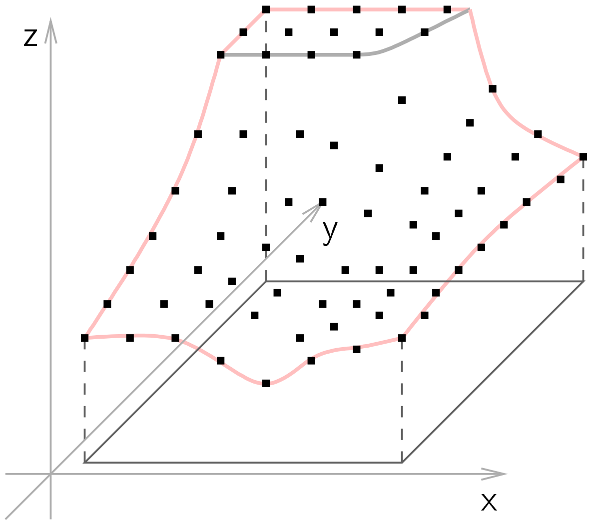


Abbildung 1: Eingabedaten als DRI

- $I$ sei das DRI mit:

$$I = \{p_{i,j} \equiv (x(i),\, y(j),\, z(i,j)) \\ : 0 \leq i \leq K, 0 \leq j \leq L\}$$

[1]auch „digital surface“ oder Punktwolke genannt

# Einführung / Vorbereitung 2

- $(i, j)$ sind also „vereinfachte“ Koordinaten, die sich linear aus $(x, y)$ berechnen lassen

- zu jedem Punkt $p_{i,j}$ läßt sich ein Normalenvektor $n_{i,j}$ von $I$ berechnen, da die Eingabedaten sehr dicht und sehr genau sind

- $\forall p_{i,j} \in I : z(i, j) > 0$
  $\Rightarrow\ z(i, j) \leq 0 \iff p_{i,j} \notin I$

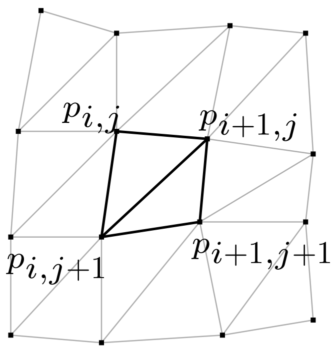


Abbildung 2: Triangulierung im Punkt $p_{i,j}$

- $I$ läßt sich triangulieren: (Abb. 2)

$$\begin{aligned}\triangle_{i,j} \equiv \{ & \triangle\left(p_{i,j}, p_{i+1,j}, p_{i,j+1}\right), \\ & \triangle\left(p_{i+1,j}, p_{i+1,j+1}, p_{i,j+1}\right)\}\end{aligned}$$

# Einführung / Vorbereitung 3

- analog lassen sich Vierecke $Q_{i,j}$ definieren und die Menge aller Vierecke:

$$Q \equiv \bigcup_{\substack{0 \le i \le K-1 \\ 0 \le j \le L-1}} Q_{i,j}$$

- Begriff der „Region" spielt wesentliche Rolle: Region ist sowohl eine Teilmenge von $Q$ als auch die dazugehörige Teilmenge von $P$.

- Menge aller Triangulierungen in $R$ nach obigem Schema wird mit $\hat{R}$ bezeichnet

- Vierecke können sichtbar (alle Punkte gehören zum DRI) oder unsichtbar (mind. 1 Punkt gehört nicht zum DRI) sein. Dies wird in sog. „visibility labels" gespeichert:

$$V_{i,j} = \begin{cases} 0 & \text{falls } Q_{i,j} \text{ sichtbar} \\ -1 & \text{falls } Q_{i,j} \text{ nicht sichtbar} \end{cases}$$

# Einführung / Vorbereitung 4

- die Domain $d(R)$ einer Region ist das kleinste achsenparallele Rechteck, das alle Projektionen der Punkte aus $R$ auf $z = 0$ enthält (siehe Abb. 3)

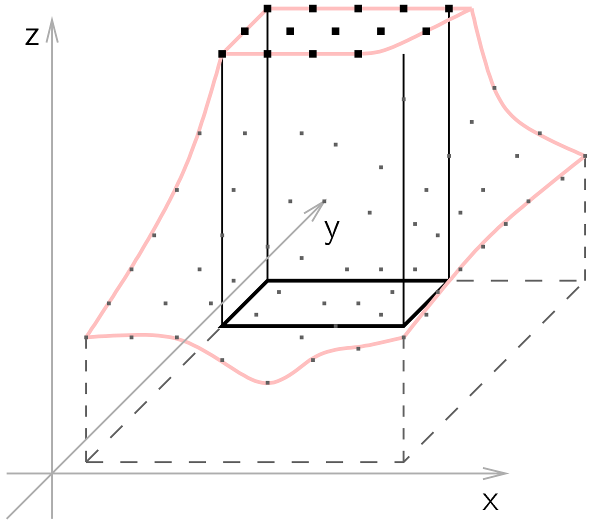


Abbildung 3: Domain einer Region

- das DRI $I$ wird jetzt betrachtet als

$$I \equiv \{p_{i,j} \cup Q_{i,j} \cup V_{i,j}\}$$

# Diskrete Fehlerquadratmethode

- beschreibt nach Gauß Kriterium für Approximation von Kurven und Flächen durch Funktionen des $\mathbb{R}^3$.
- geg.: Punkte $P_i(x_i, y_i, z_i); \quad i = 0, \ldots, N$
- gesucht sei Fkt. $z = f(x, y, b_1, b_2, \ldots, b_k, \ldots, b_n)$ mit $b_k$ als unbekannten Formparametern, die $z$-Abstände $d_i$ zwischen den Punkten $P_i$ und $f$ minimiert:

$$\sum_{i=0}^{N} (d_i)^2 = \sum_{i=0}^{N} (z_i - f(x_i, y_i, b_1, \ldots, b_n))^2 \Rightarrow \text{ min.}$$

- setzt man $f$ jetzt aus $n$ linear unabhängigen Fkt. $\varphi_k(x, y)$ und lin. Formfaktoren $b_i$ zusammen, läßt sich das Problem in Matrixschreibweise bringen:

$$d_i = z_i - \sum_{k=0}^{n} \varphi_k(x_i, y_i) b_k \Rightarrow \; D = Z - MB$$

- $B$ ergibt sich dann[2] aus $M^T M B = M^T Z$
- Minimierung der $z$-Abstände hier willkürlich gewählt

[2] siehe auch [HL92], Seite 66 ff.

# Problemstellung

Ein gegebenes „Dense Range Image“ $I$ soll unter Beachtung einer geg. Fehlertoleranz $T$ in $N$ quasi-diskunkte Regionen[3] $R_q$ mit zugeh. polynomiellen Funktionen $z_q(x, y)$; $q = 1, \ldots, N$ zerlegt werden, so daß:

(A) für jeden Punkt $p \in R_q$ der Abstand (Fehler) von $p$ zur Fläche $z_q$ kleiner als $T$ ist und

(B) jede Region $R_q$ so viele Vierecke, wie möglich aus $I$ enthält.

Zur Sicherung der $C^1$ oder $C^2$ Stetigkeit zwischen benachbarten Teilflächen wird weiter gefordert, daß

(C) jede Region $R_q$ einer Triangulierung $\hat{R}_q$ entspricht, die so wenig wie möglich Löcher hat[4]

(D) die Domain $d(R_q)$ jeder Region $R_q$ einem Quadrat möglichst nahe kommt

[3]Zwei Regionen sind quasi-disjunkt, genau dann, wenn Ihre Schnittmenge leer ist oder nur aus Punkten besteht, die am „Rand“ beider Regionen liegen. Kurz: Die Regionen dürfen sich nicht überlappen.

[4]idealerweise nur mit dem physikalischen Modell korrespondierende

# Algorithmus, grob

Der Algorithmus ist eine modifizierte Variante von „Region Growing“ (RG), das für Probleme der Bildverarbeitung eingesetzt wird[5].

(1) kleine „seed region“ $R^{k=0}$ mit einigen Punkten aus $I$ wird ausgewählt. Ein Polynom $z^{k=0}$ von Grad $t \times t$ (z. B. $t = 3$ oder $t = 4$) wird an die Punkte aus $R^{k=0}$ angepaßt.

(2) Ein Schwellwert $H < T$ wird gewählt (interaktiv oder algorithmisch).

(3) Es werden weitere, zu $z^k$ „kompatible“ Vierecke aus $I$ zu $R$ hinzugenommen und $R = R^{k+1}$ bestimmt. Kompatibel sind Vierecke, deren Punkte

  (a) einen Abstand kleiner als $H$ zu $z^k$ haben
  (b) einen zugehörigen Normalenvektor fast parallel zum positiven Normalenvektor von $z^k$ haben

(4) Ein Polynom $z^{k+1}$ von Grad $t \times t$ wird an die Punkte in $R$ angepaßt und $k$ erhöht.

[5] „seed fill“ benutzt z. B. ähnliche Ansätze

(5) Schritte (3) und (4) werden so lange wiederholt, wie $R$ noch signifikant wächst. Sonst wird $H$ erhöht und (3) und (4) wiederholt, falls $H < T$ ist.

(6) Wenn noch eine signifikante Anzahl von Punkten in $I$ übrig sind, wird (1)-(5) auf das verbleibende Bild $I \setminus R$ angewendet.

Der hier genannte Algorithmus hat den Nachteil, daß er die Anforderungen (C) und (D) nicht beachtet, die allerdings essentiell für CAD- bzw. CAE-Anwendungen sind. Die Beachtung von (C) und (D) in RG wäre allerdings sehr kostspielig; deshalb wird eine zweistufige Strategie verfolgt - dazu später mehr.

# Algorithmus 1

## Bemerkungen:

- Grad $t$ der Polynome wird i. A. durch CAD- / CAE-Sytem bestimmt, das für Weiterverarbeitung verwendet wird

- Maximale Fehlertoleranz $T$ vom Anwender vorgegeben; hängt stark vom Einsatzgebiet ab[6]

- Repräsentation der Regionen mit Hilfe der $V_{i,j}$:

$$Q_{i,j} \in R_q \iff V_{i,j} = q$$

- Repräsentation der $z_q$ nach Belieben (z. B. B-spline oder Bezier); SAPIDIS und BESL wählten die Form „diskrete orthogonale Polynome“ [Bes88]

[6] im Automobildesign z. B. Werte von 0.05 mm bis 0.40 mm

# Algorithmus 2
# *RegionGrowing* - 1

## Variablen

*RemainingQuads:* Anzahl der sichtbaren Vierecke aus $I$, die noch zu keiner Region gehören.

*H:* Schwellenwert, gibt an, wie weit ein Punkt von der approximierenden Fkt. $z_q$ abweichen darf, um zur Region hinzugefügt zu werden; von *ThresholdSelection* ermittelt, von *LocalRegionGrowing* u. U. verändert

*V:* Toleranz des Normalenvektors - wie gut muß $n_{i,j}$ zur positiven Normale von $z_q$ passen (Details siehe *GrowRegionIN*)

$D_q$: $= \{d_q^l, d_q^r, d_q^u, d_q^d\}$, min. & max. Zeile und Spalte von Punkten aus $R_q$ (Domain von $R_q$)

# Algorithmus 3
# *RegionGrowing* - 2

- Bestimmung der $n$-ten Region beginnt mit Best. der Saatregion $R_n^{k=0}$ ($k$-te Iteration beim Ermitteln der $n$-ten Region), z. B. durch den Anwender

- erste Anpassung von $z_n^{k=0}$ an $R_n^{k=0}$ („fitting“)

- *ThresholdSelection* ermittelt $H$:

  $$H \approx W * MQF^7(R_n^{k=0}, z_n^{k=0})$$

  wobei $W$ eine Konstante zwischen $2$ und $3$ ist[8].

- Berechnung / Verwendung von $H$ anstatt von $T$ aufgrund des Rauschens in $I$

[7] Mittlerer quadratischer Fehler

[8] $W$ wurde experimentell ermittelt; hat Einfluß auf „Wachstum“

# Algorithmus 4
## *LocalRegionGrowing*

- Erweitern von $R_n$ um Vierecke, die zu $z_n$ kompatibel sind und „nah“ bei $R_n$ liegen
- Erweiterung in zwei Schritten

  *GrowRegionIN:* Wachsen innerhalb von $d(R_n)$
  *GrowRegionOUT:* Wachsen aus $d(R_n)$ heraus

  $\Rightarrow$ höhere Kontrolle über Wachstum von $R_n$

  $\Rightarrow$ keine globalen Operationen $\Rightarrow$ höhere Effizienz

- nur Vierecke, die nicht bereits in $R_1, \ldots, R_{n-1}$ enthalten sind, werden in Betracht gezogen („passende“ Vierecke) $\Rightarrow V_{i,j} = 0$

# Algorithmus 5
# *GrowRegionIN* - 1

- prüft alle Vierecke $Q$, die „passend" erscheinen, d.h. daß für alle Punkte $p_{i,j}$ von $Q$:

  (1) der Abstand von $p_{i,j}$ zu $z_n$ kleiner als $H$ ist und
  (2) das Skalarprodukt von $n_{i,j}$ mit der entspr. Normale von $z_n$ kleiner als $V$ ist

  (Abstand $p_{i,j}$ zu $z_n$ kann z. B. mit Newton-Raphson-Näherung ermittelt werden, von der ermittelten Stelle wird auch der Normalenvektor von $z_n$ gewonnen)

- versucht, so viele Vierecke aus $d(R_n)$ wie möglich in $R_n$ zu integrieren

- berechnet:

  *OldRegionQuads:* Anzahl der Vierecke in $R_n$ vor dem Wachsen
  *TotalQuads:* Anzahl in Betracht kommender Vierecke in $R_n$ vor dem Wachsen

# Algorithmus 6
# *GrowRegionIN* - 2

- Identifizierung von Vierecken, die zu $z_n$ „inkompatibel" sind, d. h. die wahrscheinlich nie in $R_n$ integriert werden

- „inkompatibel" zu $R_n$ ist ein Viereck $Q$, wenn für alle $p_{i,j}$ aus $Q$
  - der Abstand von $p_{i,j}$ zu $z_n$ größer $10H$ ist, oder
  - das Skalarprodukt von $n_{i,j}$ mit der pos. Normale von $z_n$ kleiner als $\frac{2}{3}V$ ist

  Es gibt Vierecke, die weder kompatibel noch inkompatibel zu $R_n$ sind!

- Berechnung von *OutQuads* - Anzahl der inkompatiblen Vierecke in $d(R_n)$

# Algorithmus 7
# *GrowRegionIN* - 3

- Vorbereitungen für eventuelle Ausdehnung von $R_n$ über $d(R_n)$ hinaus
- sind in einer Richtung am Rand nur sehr wenige Vierecke von $R_n$, so wird nicht in diese Richtung erweitert (wegen (B) und (D), Folie 6)
- Testen der Bedingung pro Richtung:
  - $N_T =$ Anzahl kompatibler Vierecke am Rand von $d(R_n)$ in entspr. Richtung
  - $N_R =$ Anzahl Vierecke in $R_n$
  - $N_O =$ Anzahl inkompatibler Vierecke
  - $\frac{N_R}{N_T - N_O} > 0.1 \Rightarrow$ in diese Richtung darf gewachsen werden

  Diese Information wird in *Lt*, *Rt*, *Up* und *Dn* gespeichert und an *GrowRegionOUT* übergeben.

# Algorithmus 8
# *GrowRegionIN* - 4

Ausgaben:

- $R_n^{k+1}$: erweiterte Region
- $RegionQuads$ : Anzahl Vierecke in $R_n^{k+1}$
- $OutQuads$ : Anzahl inkompatibler Vierecke in $R_n^{k+1}$
- Alle Vierecke der Region außer inkompatiblen wären in idealer Region $\bar{R_n}$; deren Anzahl sei $SegmentQuads$.

  Je mehr sich $\frac{RegionQuads}{SegmentQuads}$ $1.0$ nähert, desto besser wird $\bar{R_n}$ durch $R_n$ approximiert (siehe (C)).

# Algorithmus 9
# Regeln für *LocalRegionGrowing*

(1) wenn $H > C_{large} * T$, dann wird Herauswachsen unterbunden $\Rightarrow$ (B), (C) und (D) werden erfüllt

(2) wenn (1) nicht gilt, und $\frac{RegionQuads}{SegmentQuads} > C_{full}$ ist (mit $0 < C_{full} < 1$, z. B. $0.92 - 0.96$), dann wird Herauswachsen wieder erlaubt[9]

(3) wenn weder (1) noch (2) gilt, dann wird Herauswachsen verhindert, wenn $RegionQuads - OldRegionQuads > C_{in} * SegmentQuads$ (z. B. $C_{in} = 0.1$) gilt[10]

(4) wenn keine der o. g. Bedingungen erfüllt ist und $\frac{RegionQuads}{SegmentQuads} < C_{gap}$ ist (mit $0 < C_{gap} < 1$, z. B. $0.85$ oder $0.9$), dann wird Herauswachsen unterbunden und $H$ erhöht

(5) trifft keine der obigen Bedingungen zu, wird Herauswachsen erlaubt

---

[9] da die innere Fläche ausreichend gut approximiert wird

[10] d. h. ob *GrowRegionIN* für ausreichendes Wachstum von $R_n$ gesorgt hat

# Algorithmus 10
## *GrowRegionOUT*

- Erweiterung von $R_n$ über die Domain $d(R_n)$ hinaus
- für jede Richtung („links“, „rechts“, „oben“, „unten“) einzeln
- Erlaubnis für jede Richtung wird von *GrowRegionIN* errechnet (s. Folie 15)
- wenn erlaubt, wird versucht, $R_n$ um $N_{gr}$ Spalten[11] bzw. Zeilen in entspr. Richtung zu vergrößern
- für jede Spalte / Zeile werden die $N_T^C$ Vierecke untersucht, die innerhalb der Zeilen / Spalten von $R_n$ liegen und bei Erfüllung der Bed. zu $R_n$ hinzugefügt ($N_R^C$ = deren Anzahl), $N_O^C$ sei die Anzahl der inkompatiblen Vierecke
- ist $\frac{N_R^C}{N_T^C - N_O^C} > 0.1$, wird mit der nächsten Spalte / Zeile fortgefahren

[11] die Limitierung begünstigt quadratähnliche Regionen; ohne Limitierung könnten die Regionen entarten, $N_{gr}$ z. B. $4$

# Algorithmus 11
# Erhöhen von $H$

- wenn Region $R_n$ um Faktor $C_{grow}$ größer geworden ist (z. B. $C_{grow} = 1.05$), wird $z_n$ an neues $R_n$ angepaßt und *LocalRegionGrowing* erneut aufgerufen

- wuchs $R_n$ nicht ausreichend, so wird $H$ erhöht:

$$H := C_{incrH} * H, \quad C_{incH} > 1.0$$

(mit z. B. $C_{incH} = 1.1$)

- ist $RegionGrows \neq true$ und $H \geq T$, so wird $z_n$ nicht erneut an $R_n$ angepaßt, sondern Erweiterung von $R_n$ abgeschlossen

# Algorithmus 12
# Schlußbetrachtung

## Vorteile:

- erzeugt am phys. Modell orientierte Flächen
- wenig Interaktion mit Benutzer
- hält das Wachstum der Regionen unter Kontrolle
- kein Preprocessing notwendig
- hält sich an das „principle of simplest shape“
- effizient im Vergleich zu anderen Algorithmen

## Nachteile:

- enthält Konstanten, die auf Experimenten beruhen
- arbeitet nur auf „Height Fields“
- benötigt hochpräzise Eingabegeräte
- benötigt große Punktdichten

# Literatur


[Bes88] P. J. Besl. *Surfaces in Range Image Understanding*. Springer-Verlag, New York, 1988.

[HL92] Josef Hoschek and Dieter Lasser. *Grundlagen der geometrischen Datenverarbeitung*. B. G. Teubner, Stuttgart, 2. neubearbeitete und erweiterte edition, 1992.

[SB95] Nickolas S. Sapidis and Paul J. Besl. Direct construction of polynomial surfaces from dense range images through region growing. *ACM Transactions on Graphics*, 14(2):171–200, April 1995.