Polygonale Netze

Einführung in die Computergrafik

Wiederholung

- Organisation
- Der 3D Raum
- Dreiecksnetze
- OpenGL
- Framework für das Praktikum
- Szenengraph

Ausblick

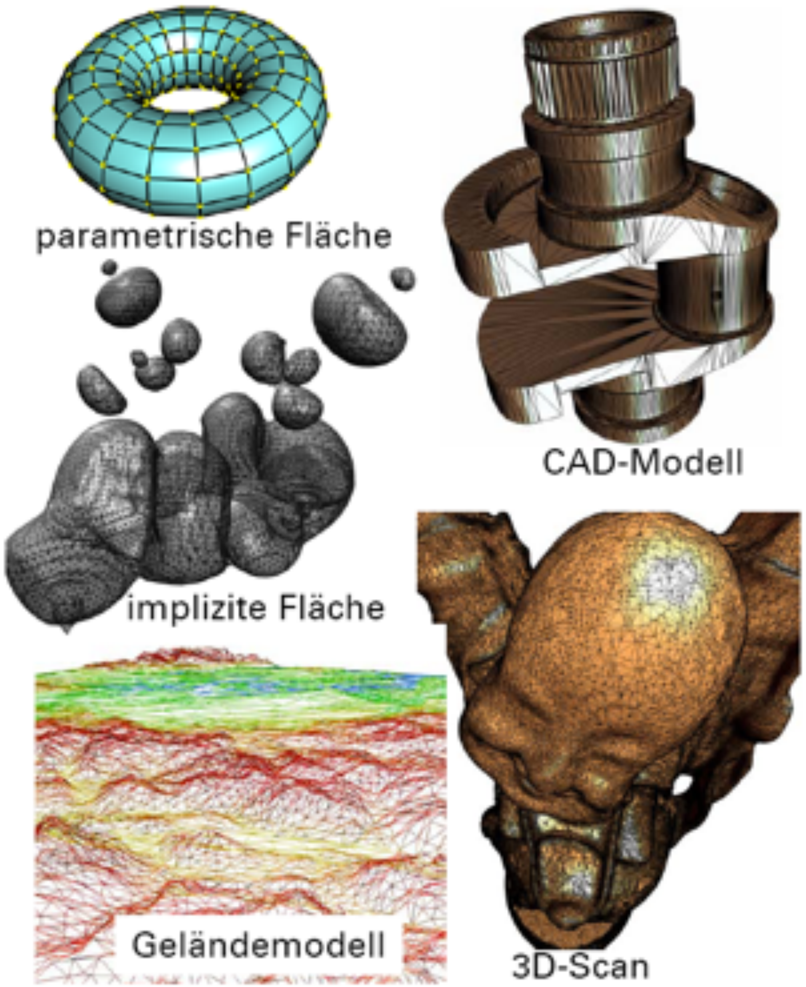
Worum gehts?

Agenda

- Einführung
- Normalen
- Volumen
- Glättung
- Nachbarschafts-Datenstrukturen
- Triangulierung von Polygonen
- Subdivision
- Vereinfachung
- Zusammenfassung

Einführung

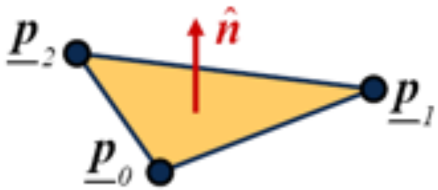
- Tesselierung parametrischer Oberflächen
- Tesselierung von CAD-Modellen aus Modelliertools wie AutoCAD, Maya, 3D Studio Max, Cinema 4D, …
- Modellierung mit wenigen Polygonen (low-poly)
- Tesselierung impliziter Flächen
- Triangulierung von Punktwolken in der Ebene, wie z.B. bei Geländemodellen
- Triangulierung von Punktwolken aus 3D Scannern



Normalen

Berechnung geometrischer Größen

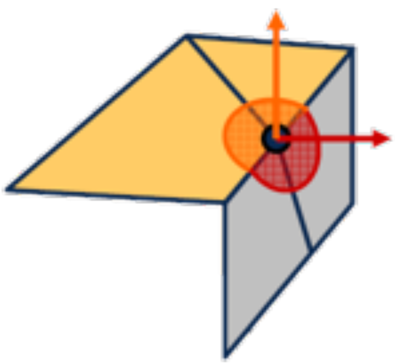
- Facettennormale
  - im Falle von Dreiecken aus Kreuzprodukt



$$\hat{n} \propto (\underline{p}_1 - \underline{p}_0) \times (\underline{p}_2 - \underline{p}_0)$$
$$= \underline{p}_0 \times \underline{p}_1 + \underline{p}_1 \times \underline{p}_2 + \underline{p}_2 \times \underline{p}_0$$

  - im Falle von Polygonen kann zweite Gleichung verallgemeinert werden:

$$\hat{n} \propto \underline{p}_0 \times \underline{p}_1 + \cdots + \underline{p}_{d-1} \times \underline{p}_0$$

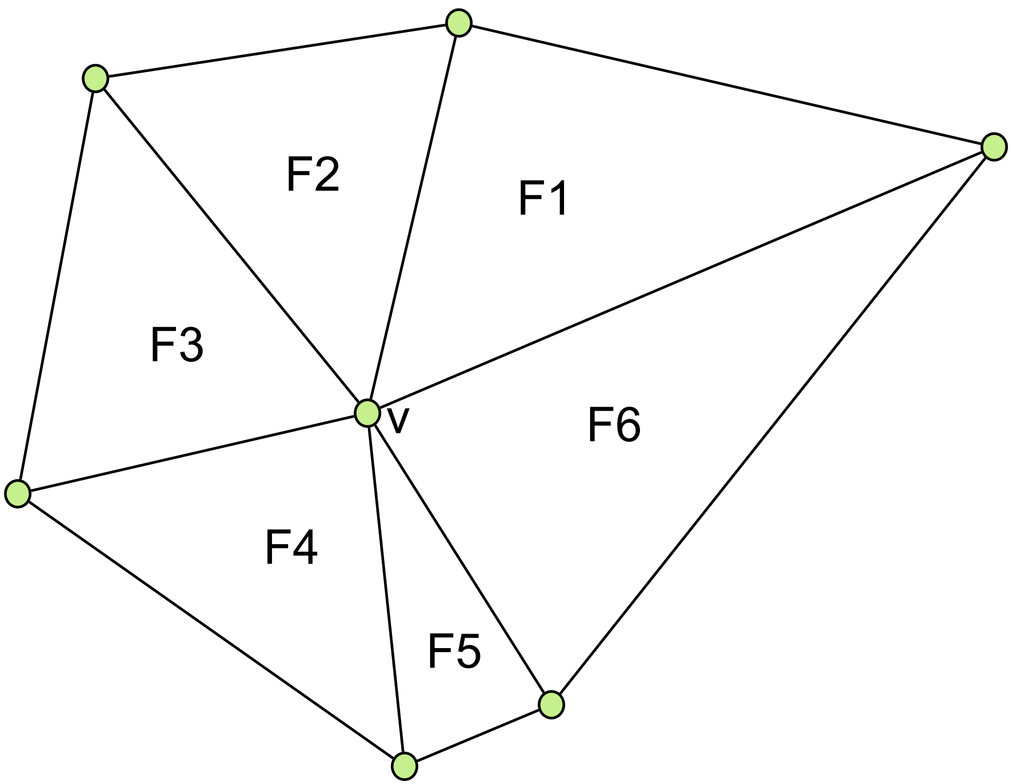


Berechnung geometrischer Größen

- Knotennormale
- gemittelte Normale aus inzidenten Facetten
- mehrere Möglichkeiten
    - Gewichte alle gleich
    - Gewichte durch Winkel
    - Gewichte durch Dreiecksfläche (ergibt Volumengradienten!)

Berechnung geometrischer Größen

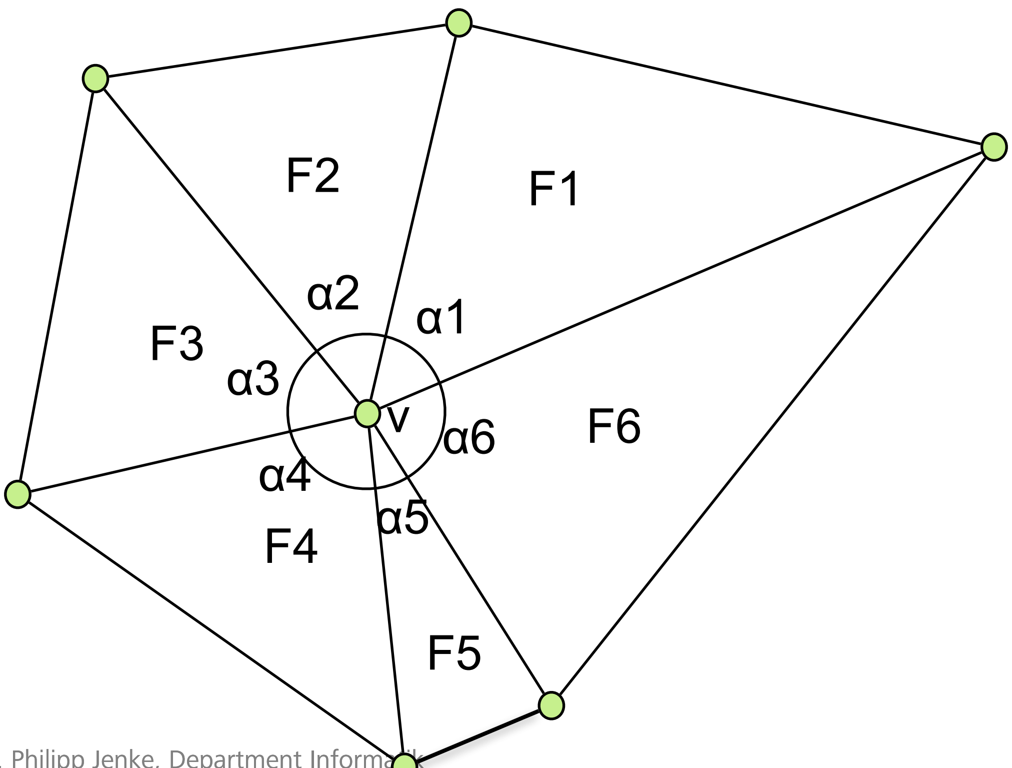
- Knotennormale: konstantes Gewicht
  - Normale der inzidenten Facette i: $\overrightarrow{n_i}$
  - deg(v) = N



$$\overrightarrow{n} \propto \sum_{i=1}^{N} \overrightarrow{n_i}$$

Berechnung geometrischer Größen

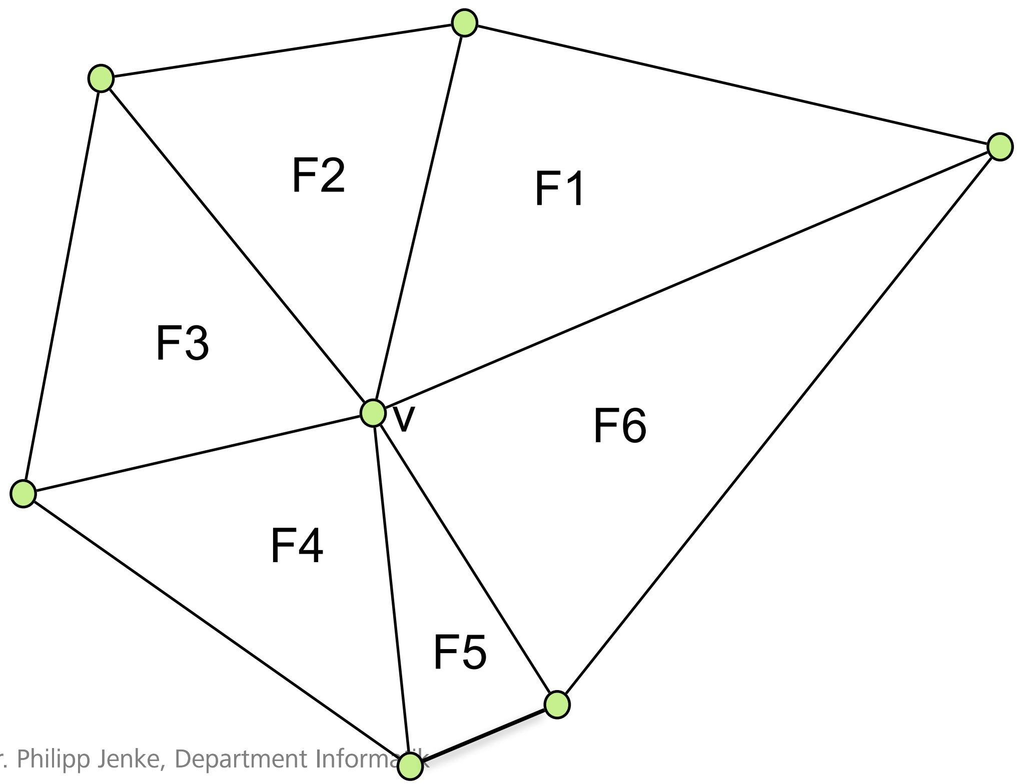
- Knotennormale: Winkel als Gewicht
  - Normale der inzidenten Facette i: $\overrightarrow{n_i}$
  - Winkel an der Facette i: αi
  - deg(v) = N



$$\overrightarrow{n} \propto \sum_{i=1}^{N} \alpha_i \overrightarrow{n_i}$$

Berechnung geometrischer Größen

- Knotennormale: Fläche als Gewicht
  - Normale der inzidenten Facette i: $\overrightarrow{n_i}$
  - Fläche der Facette i: Ai
  - deg(v) = N



$$\overrightarrow{n} \propto \sum_{i=1}^{N} A_i \overrightarrow{n_i}$$

Übung: Vertexnormale

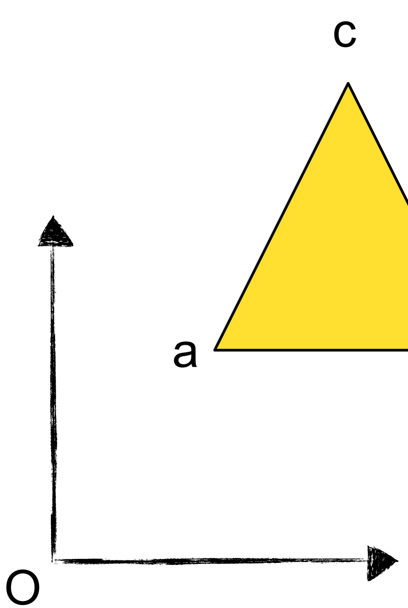
- Berechnen Sie eine Vertexnormale n.
- gegeben sind die Normalen der 5 inzidenten Facetten.
- verwenden Sie konstante Gewichtung.

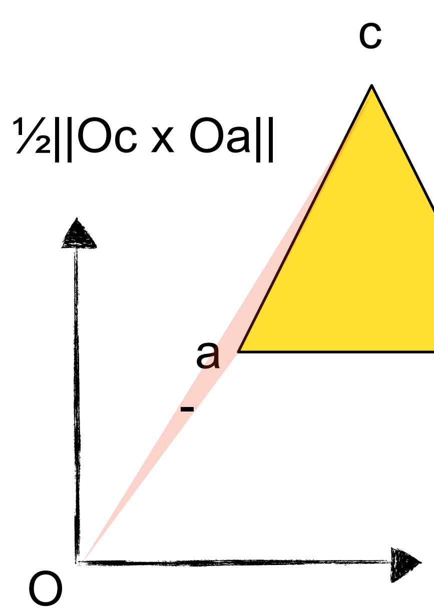
$$\begin{pmatrix} 0.5 \\ 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 1.5 \\ -0.5 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0.5 \\ 2.5 \end{pmatrix}, \begin{pmatrix} 1 \\ -2.5 \\ 1.5 \end{pmatrix}, \begin{pmatrix} 2 \\ 1.5 \\ 1 \end{pmatrix}$$

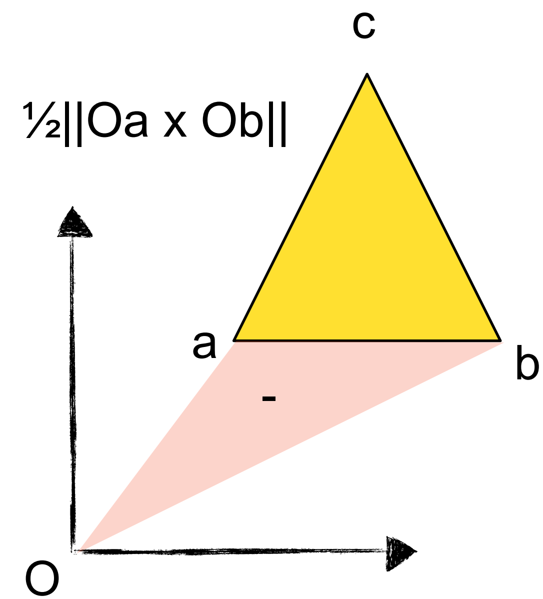
Volumen

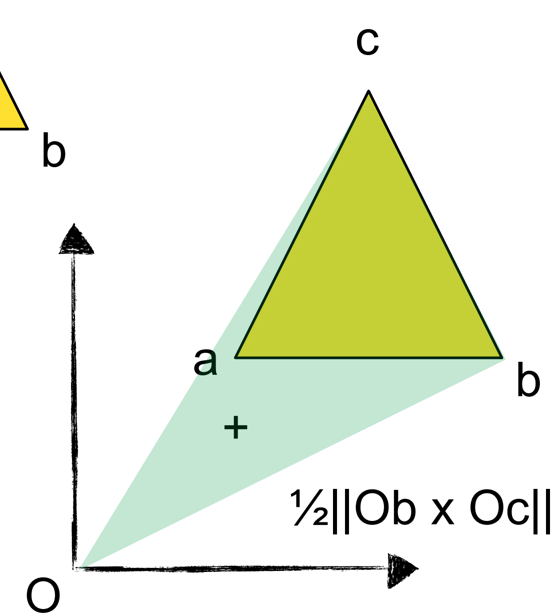
zunächst im 2D

- Idee: Berechnung über Kreuzprodukt









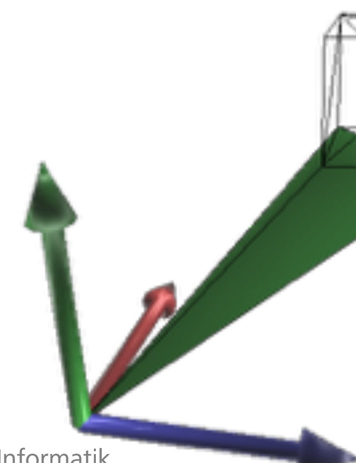
Volumen des umschlossenen Körpers

- geschlossene Flächen
- Volumen = 1/6 der Summe der Spatprodukte über alle Dreiecksknoten

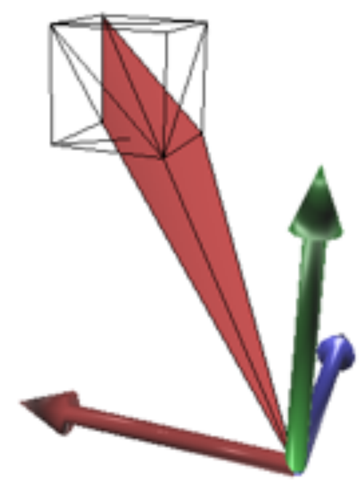
Dreieck (i,j,k)

$$V = \tfrac{1}{6}\left|\sum_{(i,j,k)\in T} \underline{p}_i \cdot (\underline{p}_j \times \underline{p}_k)\right| = \tfrac{1}{6}\left|\sum_{(i,j,k)\in T} \det(\underline{p}_i, \underline{p}_j, \underline{p}_k)\right|$$

Summe über alle Dreiecke



negativer Spat

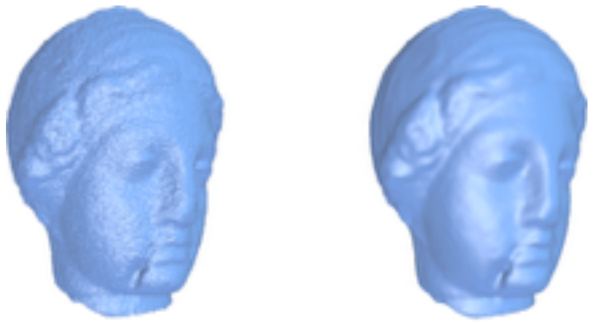


positiver Spat

Glättung

Glättung/Entrauschen

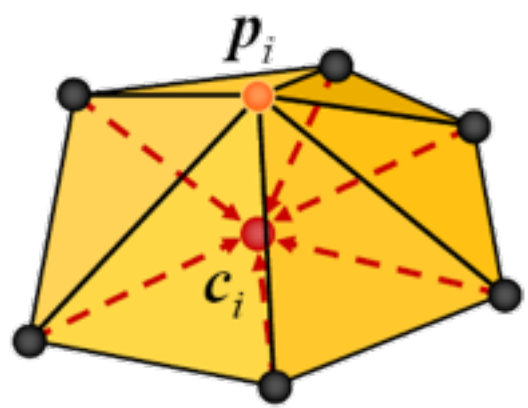
- viele Datensätze sind verrauscht
  - z.B. durch Aufnahme mit 3D Scanner
- Entrauschen/Glättung notwendig
- Idee
  - Projektion auf ein glattes Oberflächenstück
  - z.B. Ebene oder Kugel



Datensatz mit Rauschen (links), Glättung (rechts) [2]

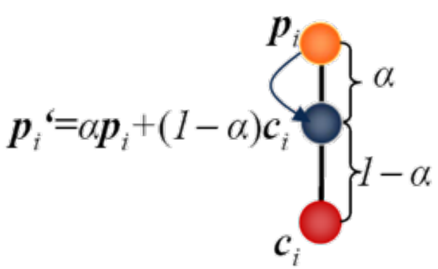
Laplace-Glättung

- Bewege jeden Vertex auf den Schwerpunkt seiner Nachbarvertices
- Algorithmus
  - betrachte jeden Vertex pi
  - berechne Schwerpunkt ci der Nachbarn



$$c_i = \frac{1}{|N(p_i)|} \sum_{p_j \in N(p_i)} p_j$$

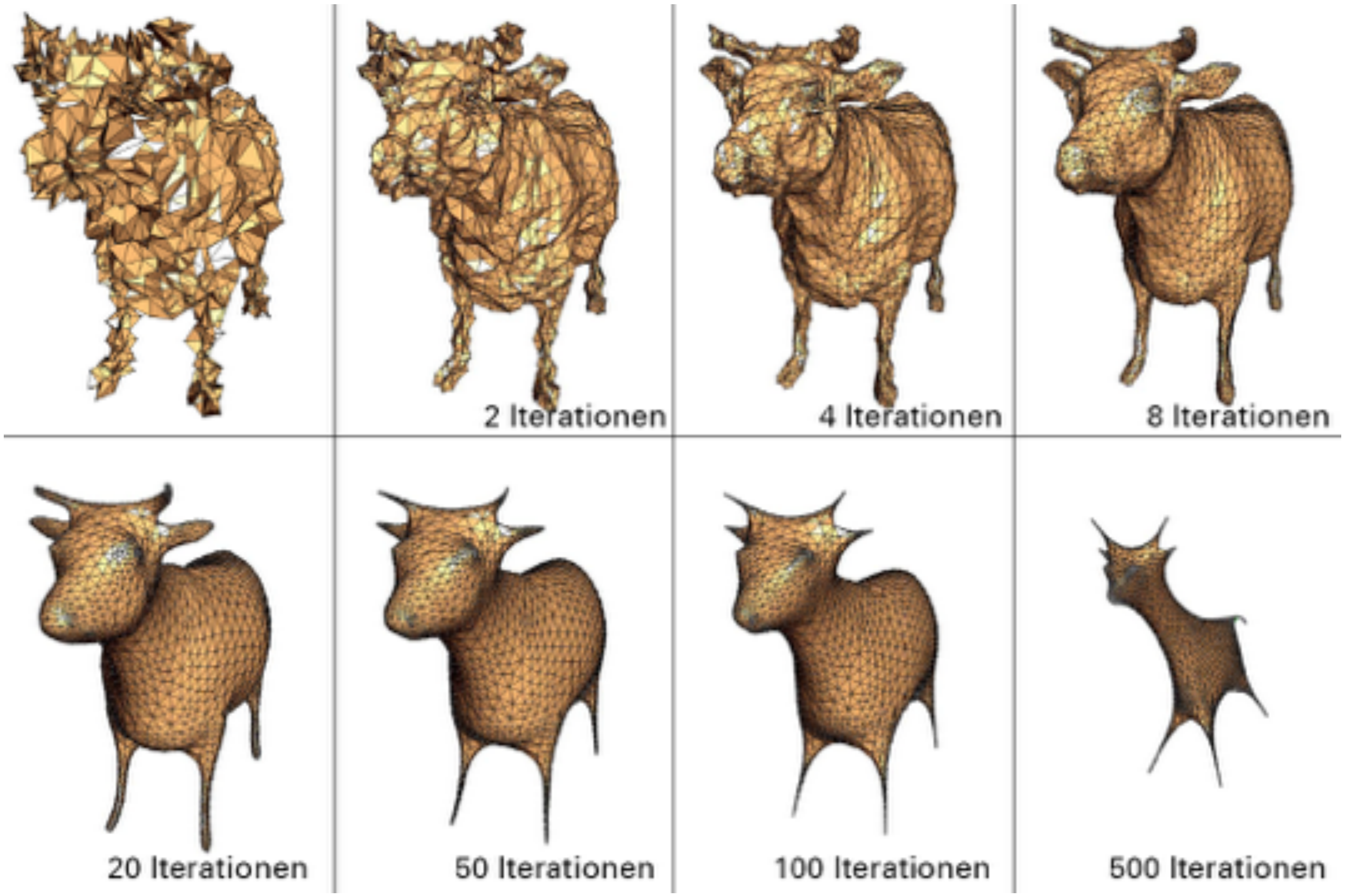
  - bewege pi in Richtung ci (z.B. 50%, dann ist α = 0.5)



- [Hinweis: erst alle ci berechnen (Zwischenspeichern), dann alle pi bewegen.]

Algorithmen: Glättung

- $\alpha = 0.3$



Nachbarschafts-Datenstrukturen

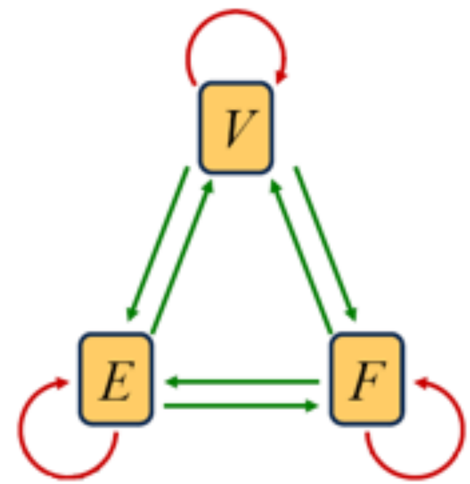
Nachbarschafts-Datenstrukturen

- stellen schnelle Navigation zwischen benachbarten Netzelementen zur Verfügung
- mögliche Relationen:
    - Inzidenz

$$\begin{array}{cc} v_i \rightarrow \{f_{i,j}\}_j & v_i \rightarrow \{e_{i,j}\}_j \\ e_i \rightarrow (v_{i,1}, v_{i,2}) & e_i \rightarrow (f_{i,1}, f_{i,2}) \\ f_i \rightarrow \{e_{i,j}\}_j & f_i \rightarrow \{v_{i,j}\}_j \end{array}$$

    - Adjazenz

$$\begin{array}{cc} v_i \rightarrow \{v_{i,j}\}_j & f_i \rightarrow \{f_{i,j}\}_j \\ e_i \xrightarrow{v} \{e_{i,j}\}_j & e_i \xrightarrow{f} \{e_{i,j}\}_j \end{array}$$



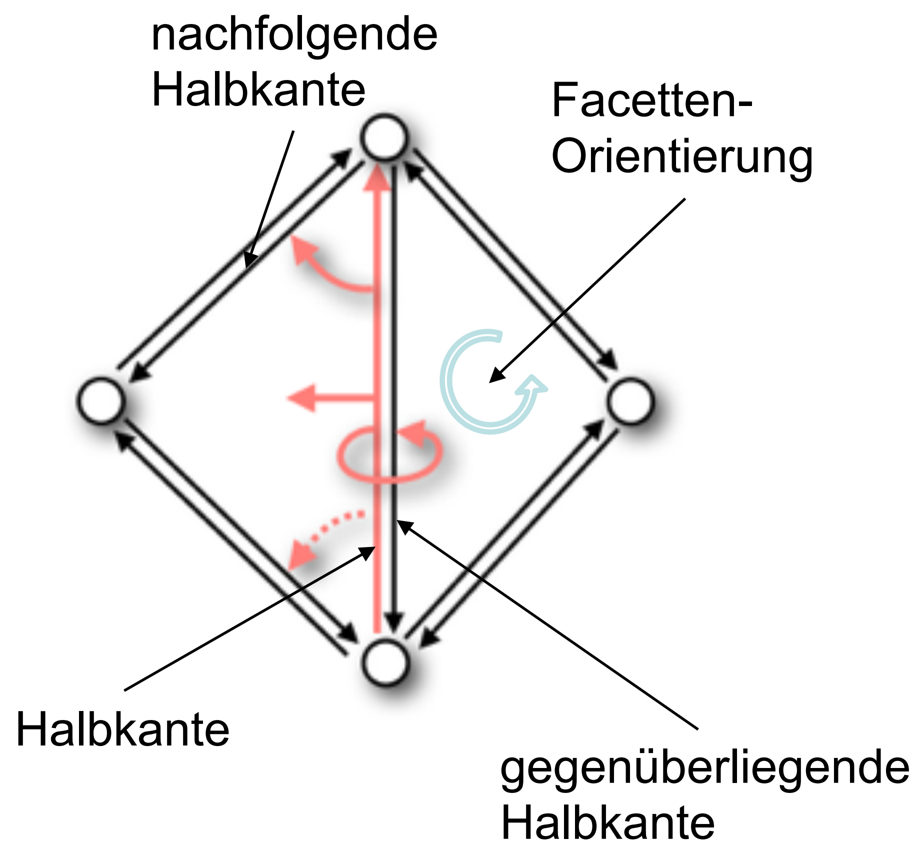
- meist Beschränkung auf wohlgeformte Netze
- Speichern aller Relationen ist teuer, vor allem auch in dynamischen Updates

Sparmaßnahmen

- Berechnung von Relationen durch wenige gespeicherte Relationen
- teilweise Speicherung mit berechenbarer Vervollständigung
- Sortierung der Netzelemente so, dass manche Relationen berechenbar werden

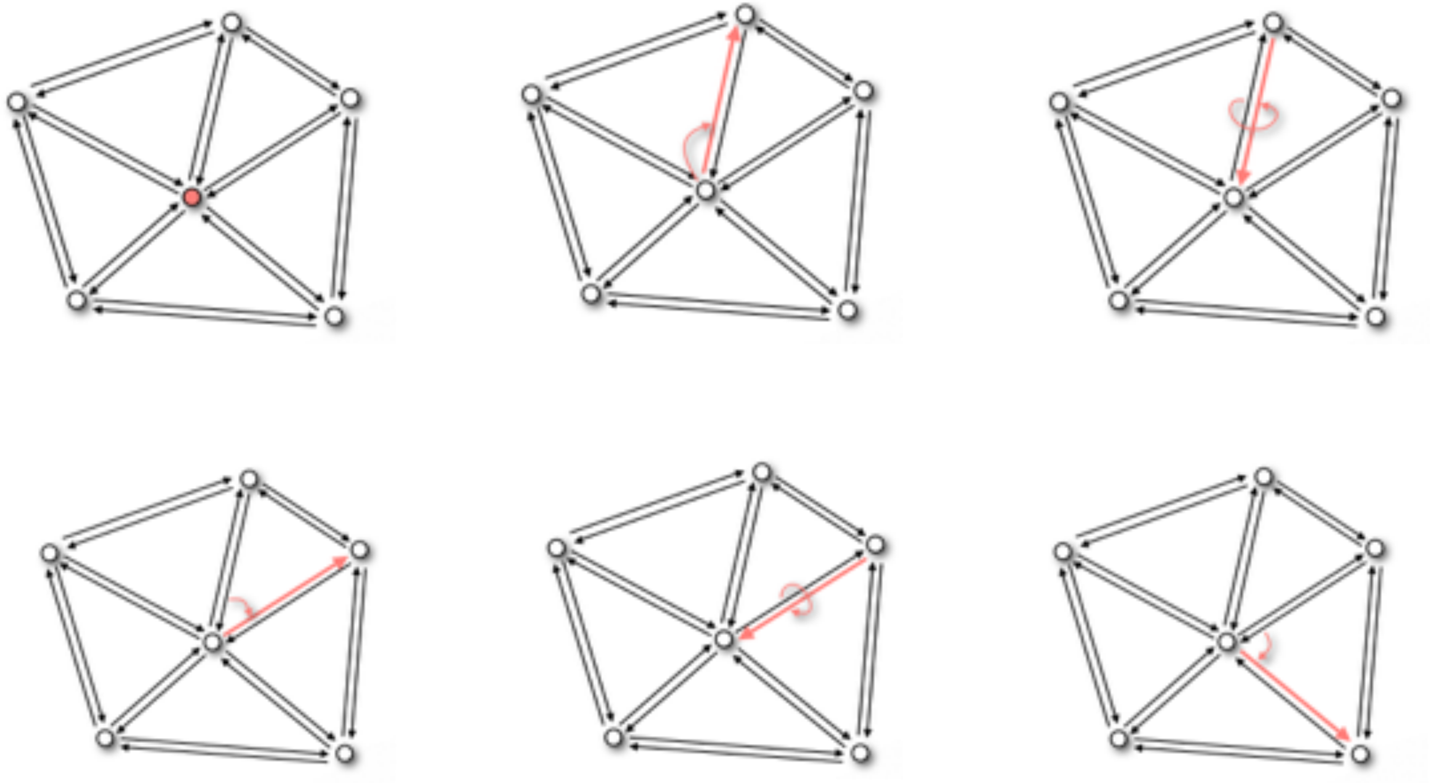
Halbkanten-Datenstruktur

- Halbkanten Datenstruktur
- pro Vertex
  - Position
  - eine Halbkante
- pro Halbkante
  - ein Vertex
  - gegenüberliegende Halbkante
  - nachfolgende Halbkante
  - zugehörige Facette
- pro Facette
  - eine Halbkante



Anwendung: Anliegende Halbkanten

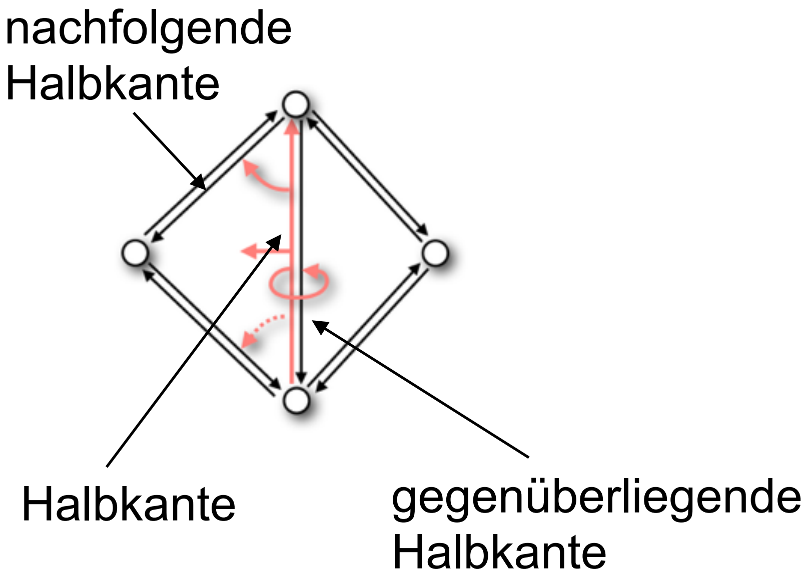
- Finden aller anliegenden Halbkanten



Übung: Adjazente Vertices

- Geben Sie Pseudocode an, um bei einem polygonalen Netz in Halbkantendarstellung für einen Vertex v alle adjazenten Vertices zu finden.

  - Halbkanten Datenstruktur
  - pro Vertex
    - Position
    - eine Halbkante
  - pro Halbkante
    - ein Vertex
    - gegenüberliegende Halbkante
    - nachfolgende Halbkante
    - zugehörige Facette
  - pro Facette
    - eine Halbkante

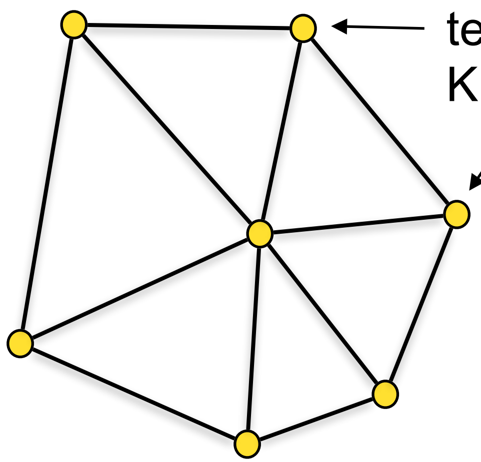


Geschlossene Netze

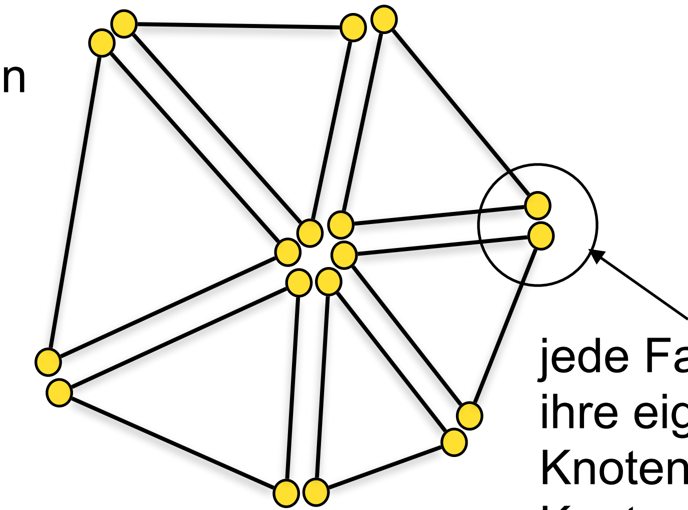
- polygonale Netze können unterschiedlich eng verbunden sein
- man unterscheidet:

Geschlossenes Dreiecksnetz

Dreieckssuppe



mehrere Facetten teilen sich einen Knoten

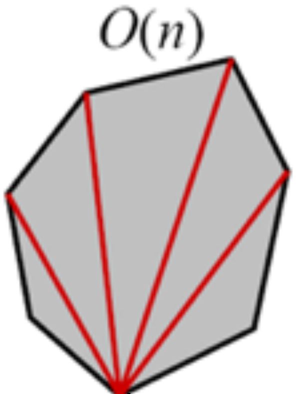


jede Facette hat ihre eigenen Knoten, mehrere Knoten haben die gleiche Position

bei der Darstellung (z.B. mit OpenGL) ist der Unterschied nicht zu erkennen

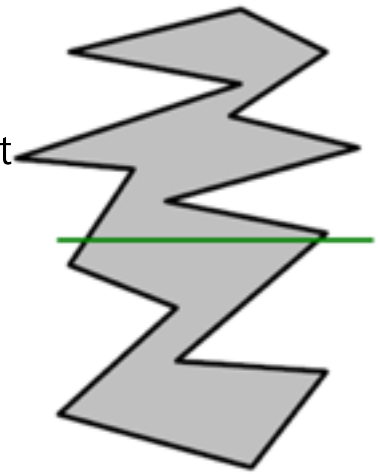
Triangulierung

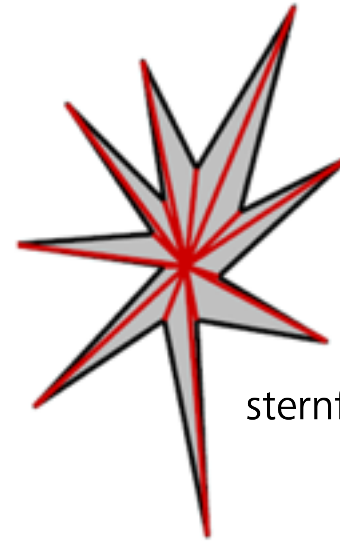
Klassifikation von Polygonen



konvex

y-monoton: der Schnitt mit jeder zur y-Achse orthogonalen Gerade ist zusammenhängend





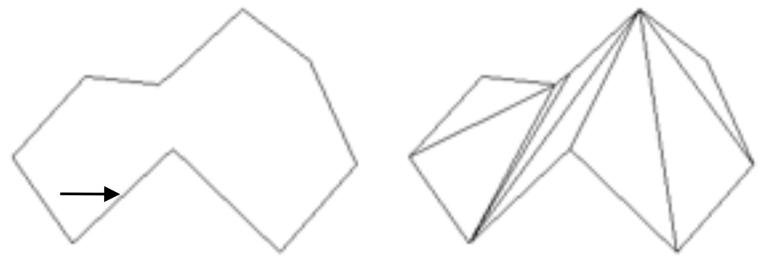
sternförmig

einfach: eine Schleife, wobei sich in jedem Knoten genau zwei Kanten schneiden und alle Kantenschnittpunkte sind Knoten

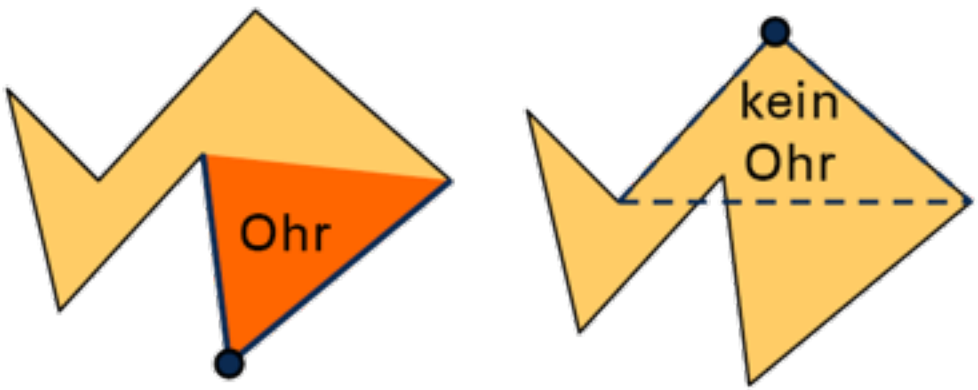
einfach mit Löchern

allgemein

Ear-Cutting Algorithmus

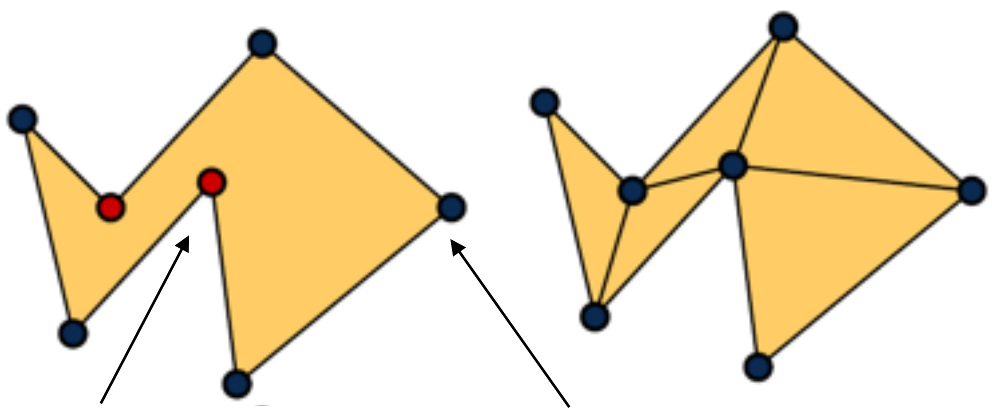


- Algorithmus zur Triangulierung von einfachen Polygonen
- ein Ohr ist eine konvexe Polygonecke, in der keine weiteren Knoten liegen
- bis auf Dreiecke haben alle einfachen Polygone mindestens zwei Ohren
- ist eine Ecke kein Ohr, so liegt wenigstens ein konkaver Knoten in der Ecke

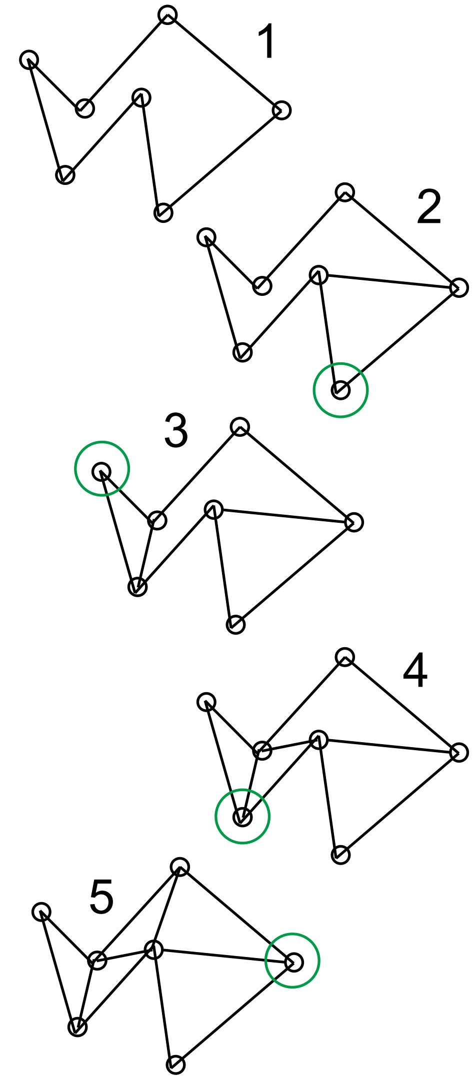


Ear-Cutting Algorithmus

```
klassifiziere alle Knoten in konvex
      oder konkav;
wiederhole n-3 mal {
   iteriere über alle konvexen Knoten {
      prüfe, ob Ecke des Knotens ein Ohr
      ist (nur gegen konkave Knoten
      testen); wenn ja, schneide Ohr ab,
      klassifiziere benachbarte Knoten
      neu und breche innere Iteration ab;
   }
}
```

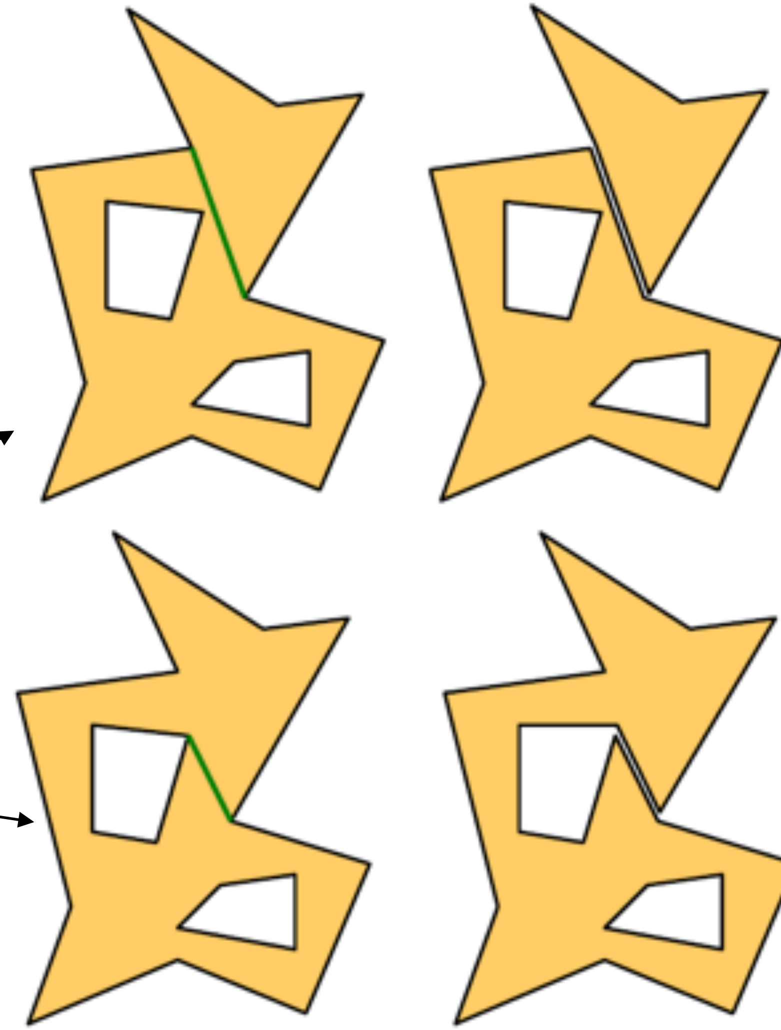


konkav (Innenwinkel > 180°)   konvex (Innenwinkel < 180°)



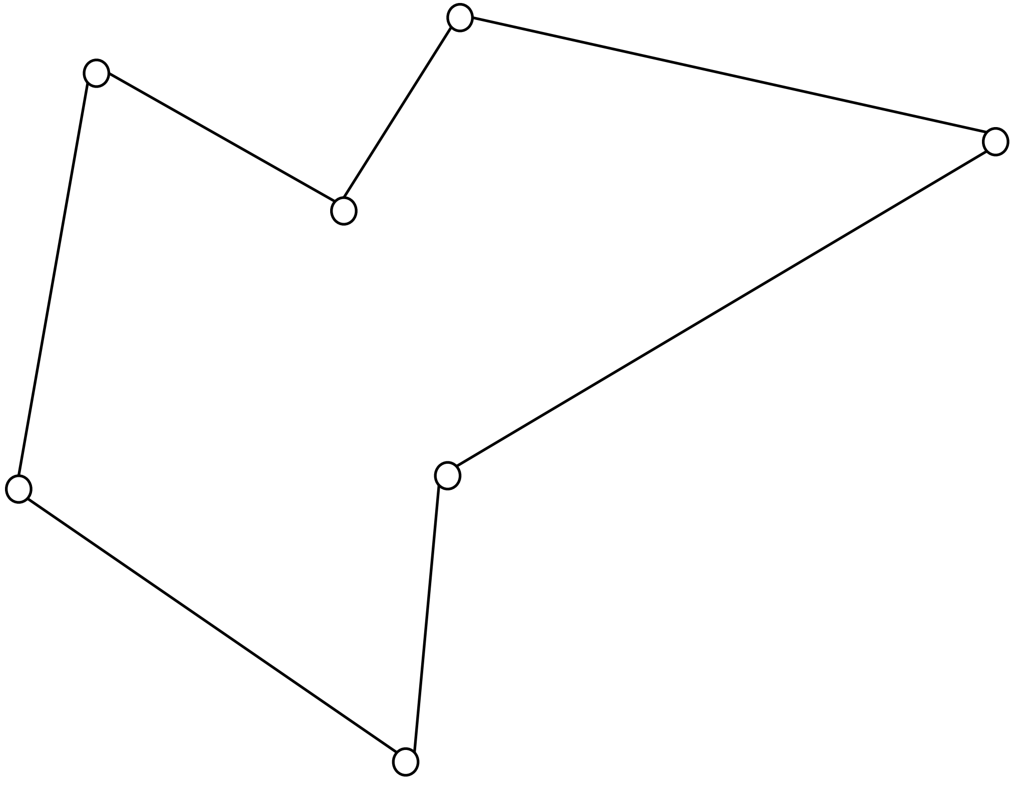
Triangulierung von Polygonen

- Satz
  - jedes einfache Polygon mit Löchern kann trianguliert werden
- Beweisidee
  - in jedem einfachen Polygon (mit Löchern) gibt es eine innere Diagonale, die
    - das Polygon in zwei einfache Polygone (mit Löchern) zerteilt oder
    - ein Loch eliminiert
  - zerlege Polygon rekursiv bis nur noch Dreiecke übrig sind



Übung: Triangulierung

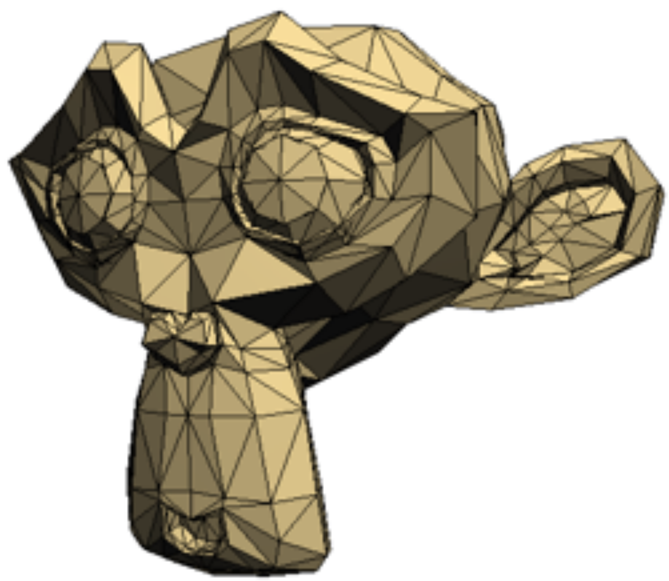
- Triangulieren Sie das folgende Polygon mit dem Ear-Cutting-Algorithmus



Subdivision

Ziel

- Verfeinern von (groben) Oberflächen
  - zur Darstellung
  - zur Manipulation

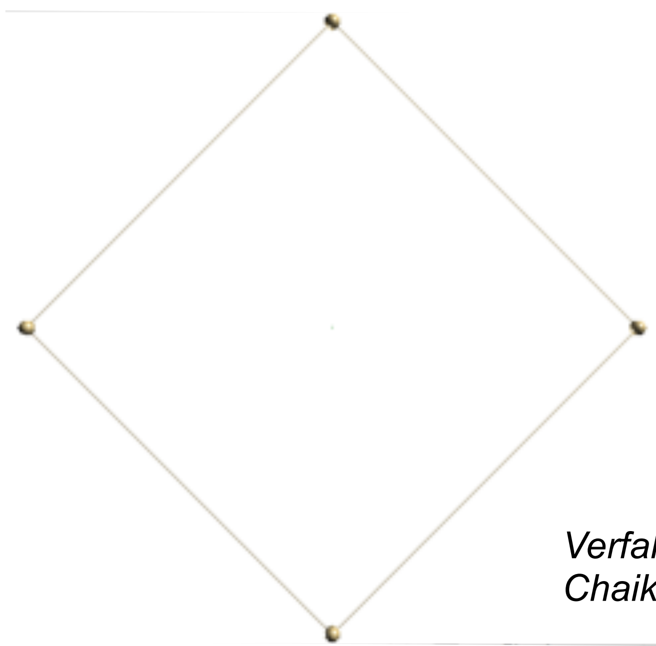


Suzanne, a primitive in the 3D modelling program Blender.

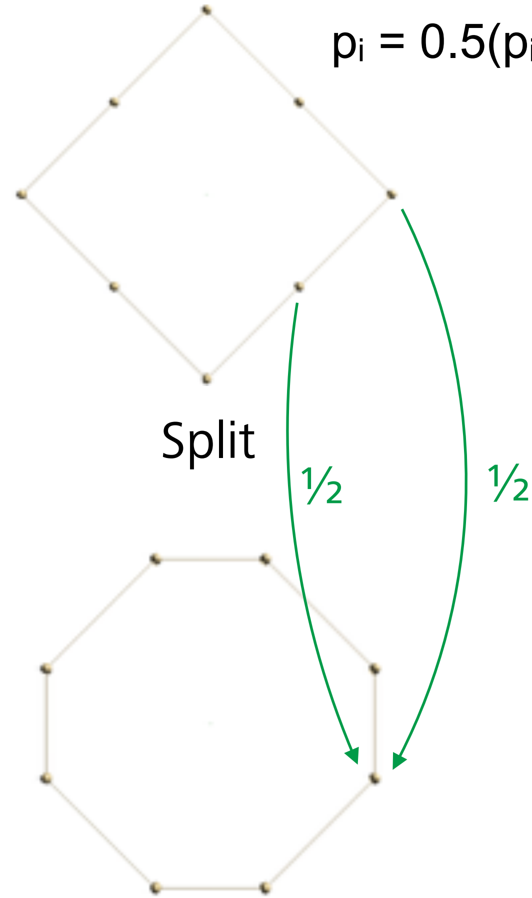
2D Subdivision: Split + Durchschnitt

- Idee (Zweischrittverfahren):
    - Unterteilung (Verfeinerung)
    - Durchschnittsberechnung (Glättung)

$p_i = 0.5(p_i+p_{i+1})$



Ausgangspolygon



Durchschnitt

Verfahren: Corner-Cutting, Chaikin, 1974

Siehe auch: https://www.youtube.com/watch?v=_IZMVMf4NQ0

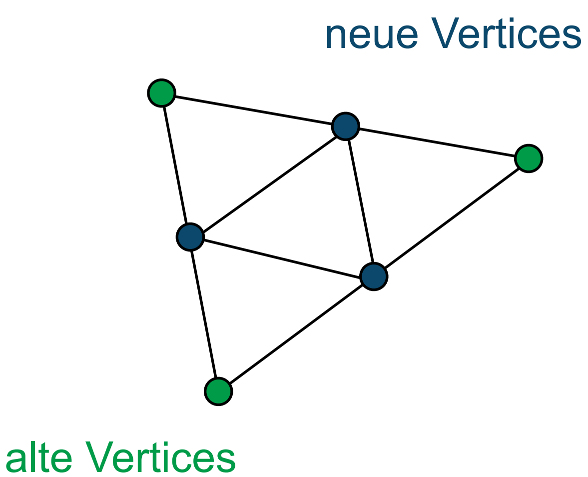
2D Subdivision

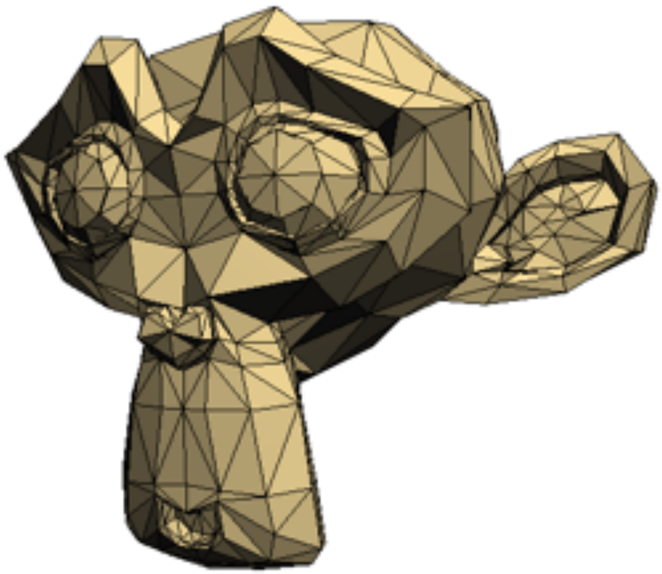
- Schrittweise Wiederholung der beiden Schritte (Split + Durchschnitt = Subdivision)

- Subdivision-Surface konvergiert gegen glatte Zielfläche
    - anders als Laplace-Glättung
    - Zielfläche kann berechnet werden
- Modellierung
    - Bewegung der Kontrollpunkte im Original-Mesh

3D Subdivision

- wieder Zweischritt-Verfahren
- Schritt 1: Dreiecke unterteilen 1 → 4
    - neuer Vertex auf jeder Kantenmitte

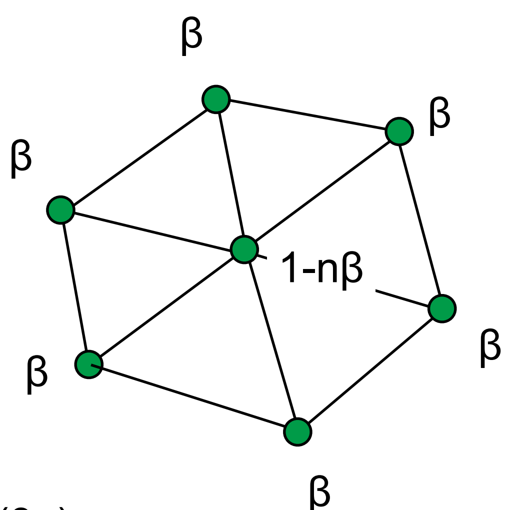




3D Subdivision

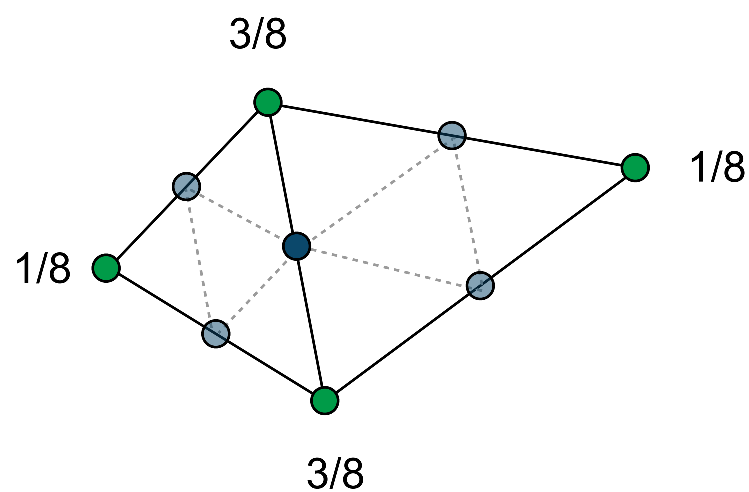
- Schritt 2: Vertices verschieben
  - verschiedene Verfahren
  - hier: Loop-Schema

- alte Vertices
  - betrachte alte Nachbarschaft

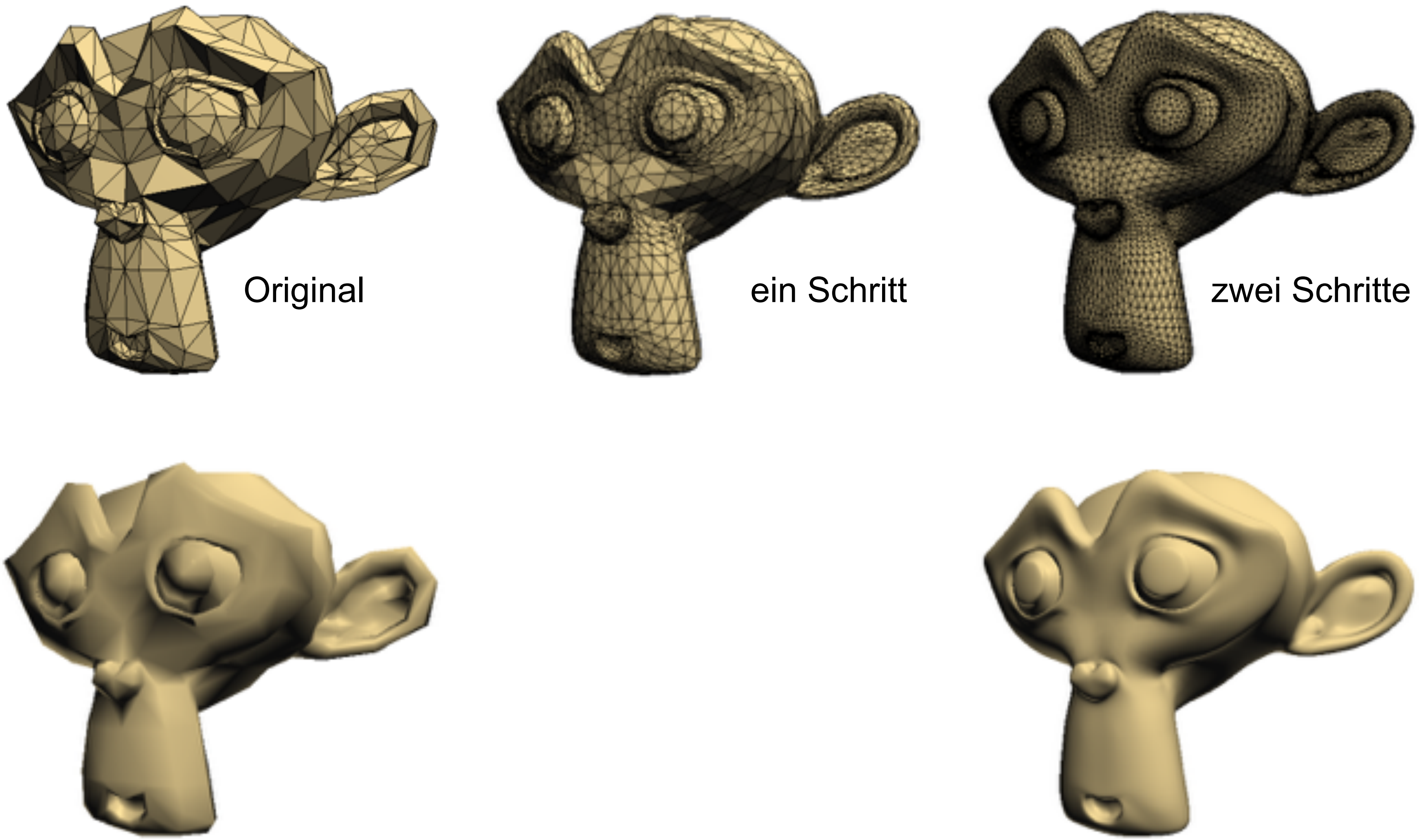


β=3/(8n)

- neue Vertices
  - alte Vertices des anliegenden alten Dreiecke

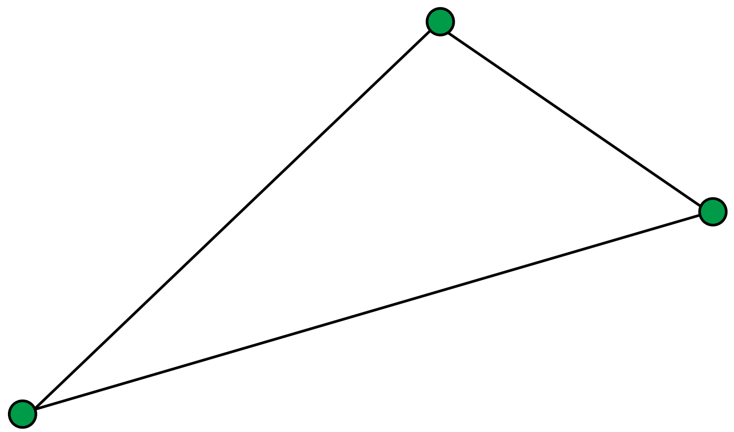


3D Subdivision



Übung: Subdivision

- Führen Sie zwei Corner-Cutting-Schritte für das folgende Polygon durch



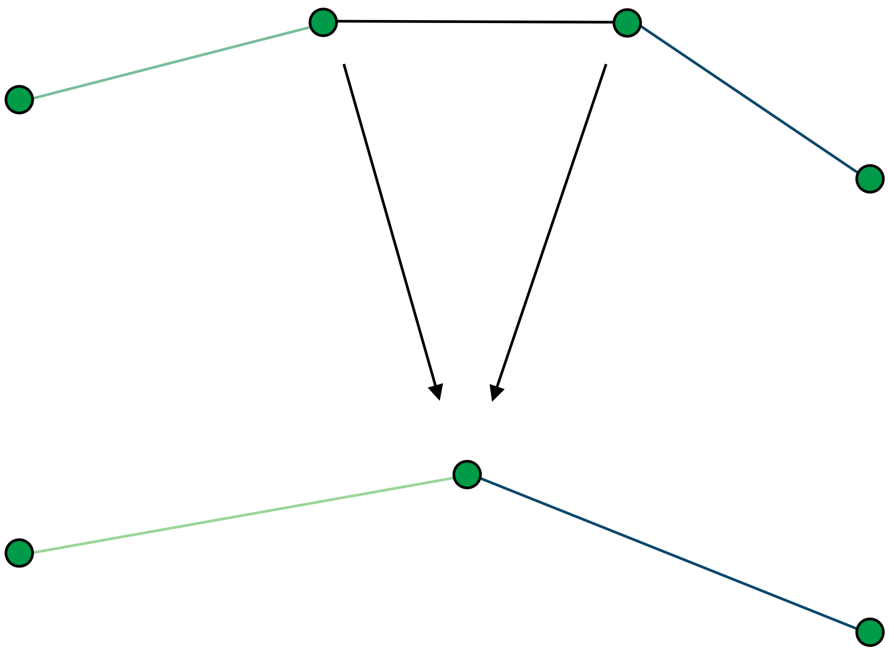
Vereinfachung

Vereinfachung

- einfache Netze häufig benötigt
  - Datenübertragung
  - Level-of-Detail
  - …
- Idee:
  - Entferne Detail (Vertices, Dreiecke) aus einem Mesh, um es zu komprimieren
  - Oberfläche des Meshes soll sich dabei möglichst wenig ändern (visuell, Volumen, Innen vs. Außen, …)
- Es gibt viele Ansätze, wir betrachten:
  - Fehlerquadriken (Garland, Heckbert, 1997)

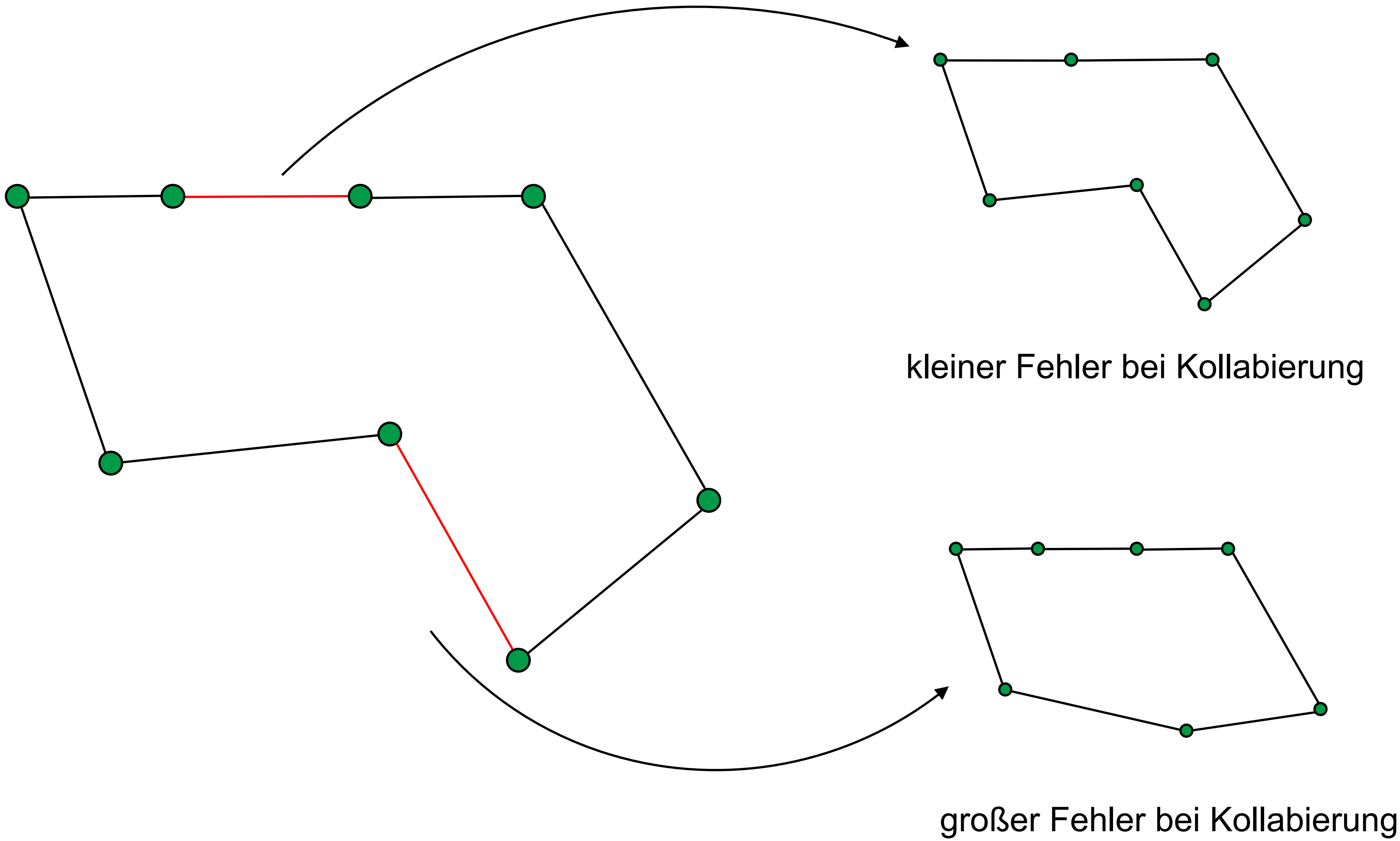
Kantenkollabierung

- Kante wird kollabiert (entfernt)
- Vertices der Kante werden verschmolzen



Fehlerquadriken

- In welcher Reihenfolge sollten die Kanten kollabiert werden?



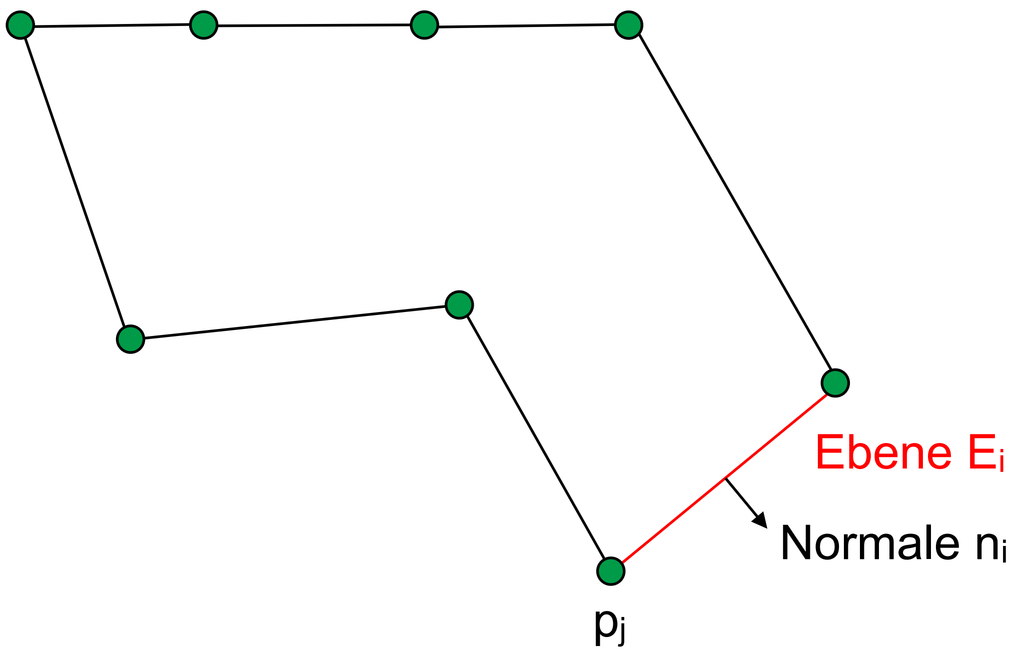
Fehlerabschätzung

- Konstruiere Ebene E_i für jedes Dreieck/Segment i

Quadratischer Abstand Punkt x zu Ebene E_i:

$$dist^2(x, E_i) = (n_i x - n_i p_j)^2$$

Hesse-Normal-Form (HNF) der Ebene



Fehlerabschätzung Dreiecke/Segmente

- Quadratischer Abstand Punkt x zu Ebene E_i:

$$dist^2(x, E_i) = (n_i^T x - n_i^T p_j)^2$$

- Trick: Verwenden homogener Koordinaten ((x,y,z) → (x,y,z,w)):

$$\bar{x} = (x, 1) \qquad \bar{n} = (n, -n_i^T p_j)$$

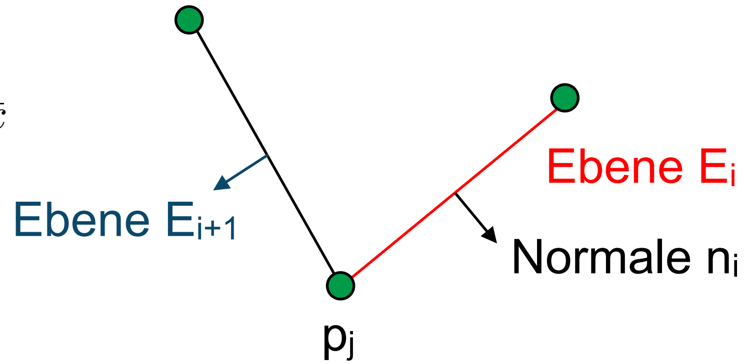
- damit vereinfacht sich die Entfernungsberechnung zu

$$dist^2(x, E_i) = (\bar{n_i}^T \bar{x})^2 = \bar{x}^T \bar{n_i} \bar{n_i}^T \bar{x} = \bar{x}^T Q_i \bar{x}$$

inneres Produkt!
Vektor * Vektor^T = Matrix

- die Matrix $Q_i = \bar{n_i}\bar{n_i}^T$ wird für alle Dreiecke/Kanten bestimmt
- damit ergibt sich eine Fehlermatrix für jeden Vertex durch Aufsummieren der Q_i der inzidenten Dreiecke/Kanten:

$$E_j(x) = \sum_{i \in N(j)} \bar{x}^T Q_i \bar{x} = \bar{x}^T (\sum_{i \in N(j)} Q_i) \bar{x} = \bar{x}^T Q \bar{x}$$



Fehlerabschätzung Kanten

- Fehlerberechnung pro Vertex

$$E_j(x) = \sum_{i \in N(j)} \bar{x}^T Q_i \bar{x} = \bar{x}^T (\sum_{i \in N(j)} Q_i) \bar{x} = \bar{x}^T Q \bar{x}$$

- Fehlerquadrik pro Kante $e_{(j,k)}$ (von p_j nach p_k):

$$Q_{e_{j,k}} = Q_j + Q_k$$

- Aber was bedeutet das? Wo ist der Fehler am kleinsten?

Optimale Vertex-Position

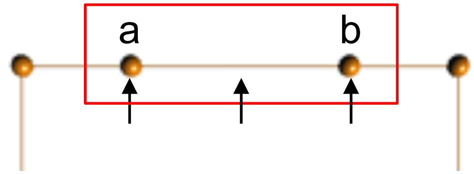
- Die optimale Vertexposition minimiert den Fehler, man muss also die Ableitung der Fehlerfunktion xQx = 0 setzen.
- durch Umstellung ergibt sich, dass man dazu folgendes Gleichungssystem lösen muss:

$$\bar{Q}_{j,k} \cdot (v^{neu}, 1) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \text{ also } (v^{neu}, 1) = \bar{Q}_{j,k}^{-1} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

- $\bar{Q}_{j,k}$ ergibt sich aus $Q_{j,k}$ durch Ersetzen der letzten Zeile durch (0,0,0,1)

Optimale Vertex-Position - Ausnahme

- Was, wenn $\bar{Q}_{j,k}$ nicht invertierbar ist (Determinante = 0)?
    - tritt auf, wenn alle Dreieck-Segment-Ebenen die gleiche Normale haben



- Suche optimale Position auf Kante
    - Punkt v muss auf der Strecke a-b liegen, also
    - finde Minimum des Fehlers (Ableitung nach λ = 0 setzen):

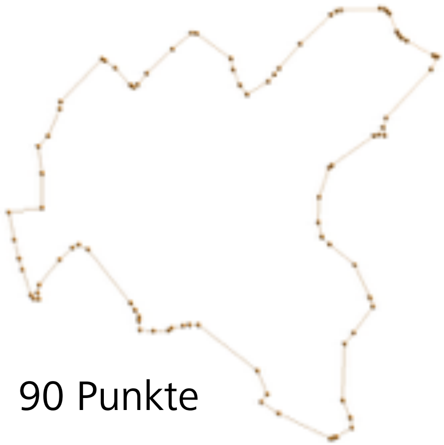
$$v^T Q_{e_{j,k}} v \rightarrow min \qquad v = (1-\lambda)a + \lambda b$$

- falls das auch nicht klappt:
    - Verwende optimale Position aus Menge $\{p_j, p_k, (p_j+p_k)/2\}$

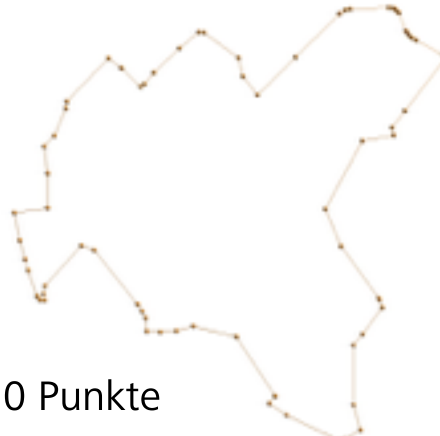
Iterative Vereinfachen

- Welche Kante wählt man zunächst aus?
    - diejenige, für die $v_{neu,j,k}^T Q_{e_{j,k}} v_{neu,j,k}$ am kleinsten ist
- Was muss man dann aktualisieren?
    - der neue Vertex $v_{neu,j,k}$ bekommt die Fehlerquadrik $Q_{e_{j,k}}$ der Kante aus der er entstanden ist
    - damit aktualisieren sich die Fehlerquadriken der anliegenden Kanten

Ergebnisse (Land Hamburg)



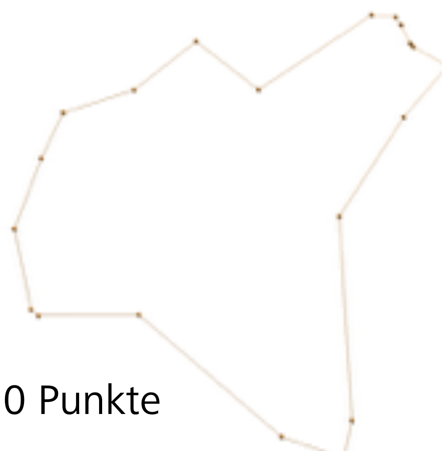
90 Punkte



70 Punkte

50 Punkte

30 Punkte



20 Punkte

10 Punkte

Zusammenfassung

```
berechne Fehlerquadriken für alle Dreiecke/Segmente
berechne daraus die Fehlerquadriken für alle Vertices
berechne daraus die Fehlerquadriken für alle Kanten
solange Vereinfachung nicht grob genug
    wähle Kante mit kleinstem Fehler
```
$v_{neu,j,k}^T Q_{e_{j,k}} v_{neu,j,k}$
```
    beim Kollabieren
    kollabiere Kante
    weise neuem Vertex die Fehlerquadrik der Kante zu
    aktualisiere Fehlerquadriken für anliegende Kanten
```

Zusammenfassung

- Einführung
- Normalen
- Volumen
- Glättung
- Nachbarschaftsdatenstrukturen
- Triangulierung von Polygonen
- Subdivision
- Vereinfachung
- Zusammenfassung

Quellen

- Die Folien basieren teilweise auf den Vorlesungsfolien von Prof. Dr. Stefan Gumhold (Technische Universität Dresden), Prof. Wolfang Straßer (emeritiert, Universität Tübingen), Prof. Hao Li (University of Southern California USC)

- [1] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, Bruno Levi: Polygon Mesh Processing, A. K. Peters, 2010
- [2] B. Mederos, L. Velho, L. H. de Figueiredo: Smooth surface reconstruction from noisy clouds, in: J. Braz. Comp. Soc. vol.9 no. 3 Campinas Apr. 2004
- Corner Cutting: Chaikin, G. An algorithm for high speed curve generation. Computer Graphics and Image Processing 3 (1974), 346–349.
- Loop-Schema: Charles Loop: Smooth Subdivision Surfaces Based on Triangles, M.S. Mathematics thesis, University of Utah, 1987
- Fehlerquadriken: M. Garland and P. Heckbert. Surface Simplification Using Quadric Error Metrics. In Proceedings of SIGGRAPH' 97