Virtual Reality

Sommer 2012

6 Mesh-Optimierung und -Übertragung

Versionsdatum: 18. Juni 2012

6 Mesh-Optimierung und -Übertragung ...

Motivation: Mesh-Reduktion

Polygonmodelle sind je nach Herkunft sehr hoch aufgelöst (> 1 Mio. Tri.)

Reduktion: Verringerung der Modellkomplexität

- möglichst geringe (visuelle) Fehler im Modell
- steuerbar je nach Zielsetzung, z.B. max. Dreieckzahl oder max. Fehler
- ggf. Entfernung unsichtbarer Objektteile (vgl. Culling)

Fragen:

- Wie wird Modell vergröbert?
- Wie misst man den entstandenen Fehler?

Motivation: Mesh-Übertragung

VR-Anwendungen sind häufig verteilte Systeme bei evtl. schmalbandigen Netzen

Fragen:

- Wie kann ein Modell komprimiert übertragen?
- Wie kann ein Modell streamingfähig gemacht werden?

6.1 Edgebreaker

Zielsetzung: Komprimierte Übertragung von Dreieckesnetzen

Naiver Ansatz: Speicherung als indiziertes Dreiecksformat

- ❍ Vertex-Koordinaten: 3×32 bit (float)
- ❍ Konnektivität: 3×32 bit pro Dreieck (int)

Pro Vertex zwei neue Dreiecke $\Rightarrow$ Konnektivitättabelle doppelt so groß

Grundgedanke: ❍ Trenne *Geometrie* (Vertexdaten) und *Topologie* (Konnektivität)

- ❍ Effiziente Speicherung der Konnektivität

Randbedingungen: ❍ Mannigfaltiges Dreiecksnetz (siehe CG2)

- ❍ Das Mesh hat Genus 0, d.h.
  - ❑ es ist geschlossen und „kugelähnlich“, **oder**
  - ❑ einen durchgängigen Rand

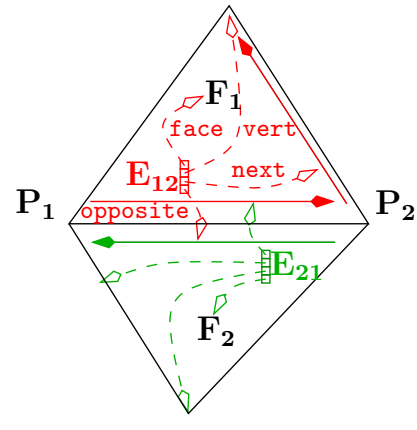
6.1 Edgebreaker ...

Half-Edge Datenstruktur

Ansatz: Zwei Halbkanten pro Kante

Daten für Halbkante h:

- $h.v$ Verweis auf Eckpunkt (**v**ert)
- $h.o$ Verweis auf andere Halbkante (**o**pposite)
- $h.f$ Verweis auf zugehöriges Polygon (**f**ace)
- $h.n$ Verweis auf nächste Halbkante (**n**ext)



Spezifisch für Edgebreaker: $h.v.m$ Vertex $h.v$ markiert (boolsch)

Weitere Operatoren auf Halbkante h

Startpunkt: $h.s = h.n.v$ Endpunkt: $h.e = h.n.n.v$ Vorige Halbk.: $h.p = h.n.n$

Vorige Randkante: $h.P$ Nächste Randkante: $h.N$ (nur bei Rand-Vertex)

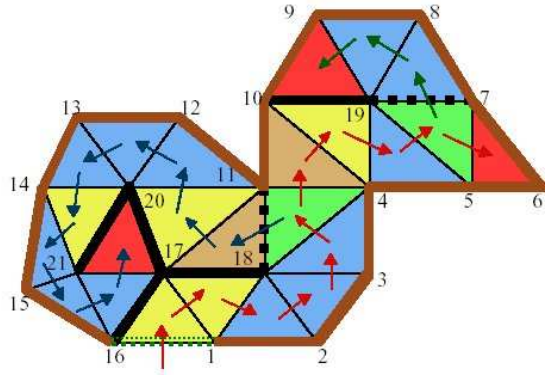
Algorithmus im Überblick

Grundsätzlich: Traversierung des Meshes

- Abarbeitung (Entfernung) der Dreiecke in bestimmter Reihenfolge
- *Active Gate* g*:* Übergangskante in aktuelles Dreieck
- *Active Boundary* bezeichnet die aktuellen Randkanten

Datenstruktur: ❍ Vertexliste: Initial Randvertices; neue Vertices anhängen

- Konnektivität: Speichern des „Weges“ durch das Mesh



Encoding: Fallunterscheidungen

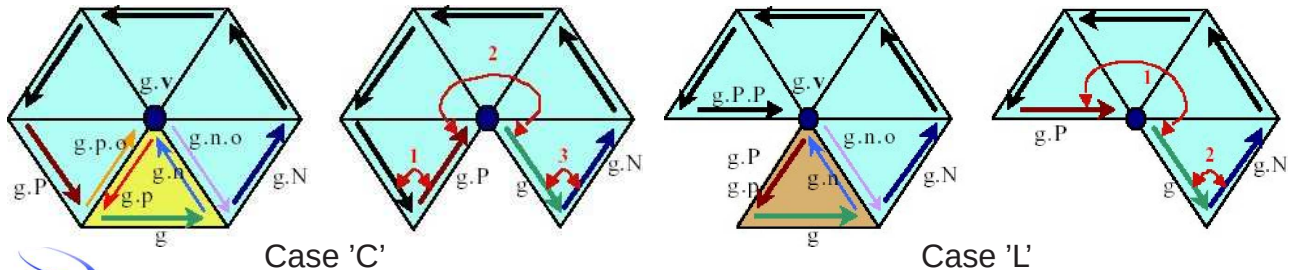
Ansatz: „Weg“ durch das Mesh anhand lokaler Situation

Case ’C’: Vertex $g.v$ noch nicht besucht, d.h. $g.v.m = \texttt{false}$

- Speichere $g.v$ in Vertexliste; speichere ’C’ in Historie; $g.v.m \rightarrow \texttt{true}$
- Active Gate $g \leftarrow g.n.o$

Case ’L’: Vertex $g.v$ vorlaufend zu Gate g auf Rand: $g.p = g.P$

- Speichere ’L’ in Historie
- Active Gate $g \leftarrow g.n.o$



Case ’C’ Case ’L’

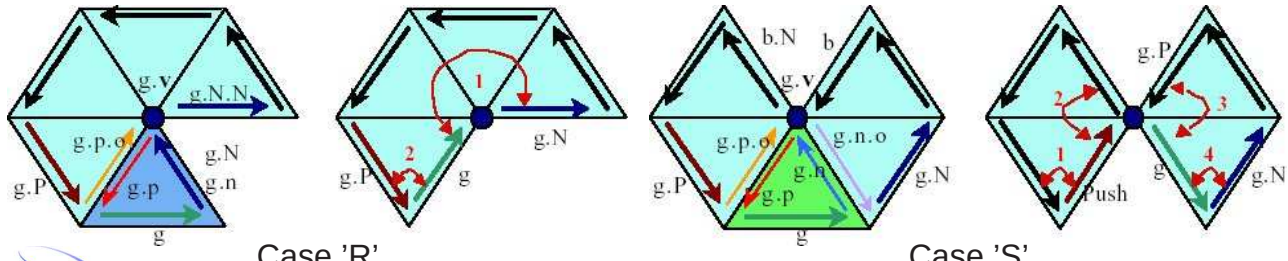
Encoding: Fallunterscheidungen (Forts.)

Case 'R': Vertex $g.v$ nachlaufend zu Gate g auf Rand: $g.n = g.N$

- Speichere 'R' in Historie
- Active Gate $g \leftarrow g.p.o$

Case 'S': Vertex $g.v$ weder vor- noch nachlaufend zu Gate g auf Rand

- Speichere 'S' in Historie
- Active Gate $g \leftarrow g.n.o$
- Push $g.p.o$ auf Kantenstack

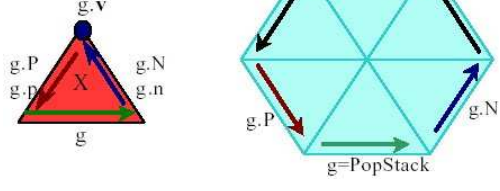


Case 'R' Case 'S'

Encoding: Fallunterscheidungen (Forts.)

Case 'E': Vertex $g.v$ vor- und nachlaufend zu Gate g auf Rand: $g.v = g.P.s = g.N.e$

- Speichere 'E' in Historie
- Pop Kantenstack
- Active Gate: Oberstes Element auf Kantenstack (sonst Ende des Algorithmus)

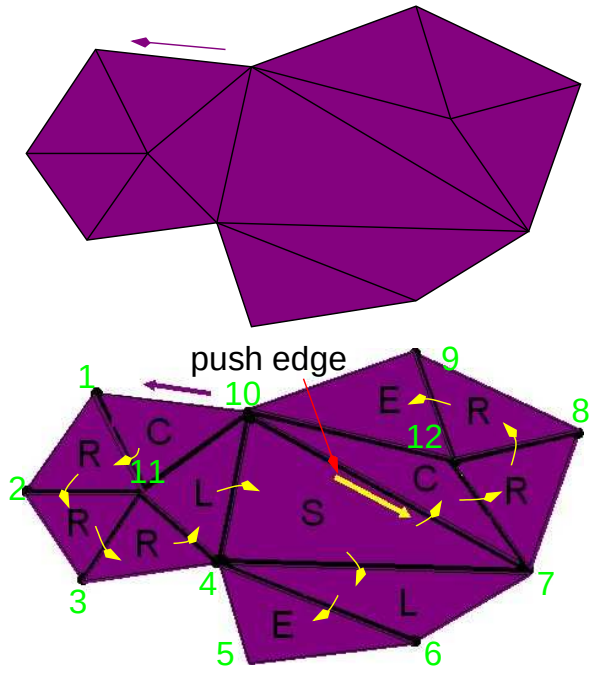


Case 'E'

Algorithmus: Encoding

```
g = pickInitialEdge();        // on boundary if any
if ( hasBoundary ) vertexList.appendBoundaryVertices(g.p.v)
edgeStack.push(g);
while ( (g = edgeStack.pop()) != NULL )   // still an element
  while ( true ) {            // loop "forever"
    if ( g.v.m == false ) handleCase('C');
    else
      if ( g.p == g.P )       // left is boundary
        if ( g.n == g.N ) {
          handleCase('E');
          break;
        }
        else handleCase('L');
      else
        if ( g.n == g.N ) handleCase('R');
        else handleCase('S');
  }
```

Beispiel:



1. Initiales Gate auf Boundary
2. Boundary Vertices an Vertex-Liste anhängen
3. Codierung (Fall 'C': $g.v$ an Vertex-Liste anhängen)
4. einmaliges `edgeStack.push` im Fall 'S'
5. Ergebnis: CLERS-String: CRRRLSLECR-RE

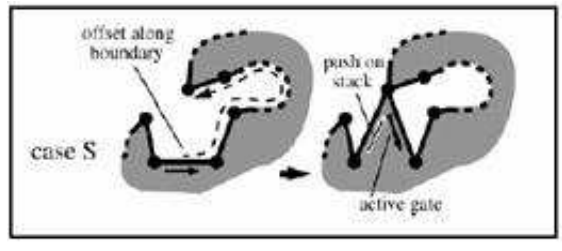
Kodierung & Dekodierung

Kodierung der Konnektivität: Jedes Dreieck durch C,L,E,R oder S kodiert

- Naiv: 5 Zeichen $\rightarrow$ 3 bit pro Dreieck
- Besser: Neuer Vertex $\Leftrightarrow$ 'C' $\Rightarrow$ 50% Wahrscheinlichkeit für 'C'
  'C' 1 bit, 'L','E','R','S' jeweils 3 bits $\Rightarrow$ im Mittel 2 bit/Dreieck

Dekodierung: Umwandlung der Vertexliste & CLERS-String in indiziertes Dreiecksnetz

Problem: Bei der 'S' Operation ist der dritte Vertex in der Vertexliste versetzt und muß erst aufwendig ermittelt werden



Case 'S': Offset-Problem

Wrap & Zip

Eingabedaten: Vertexliste und CLERS-String

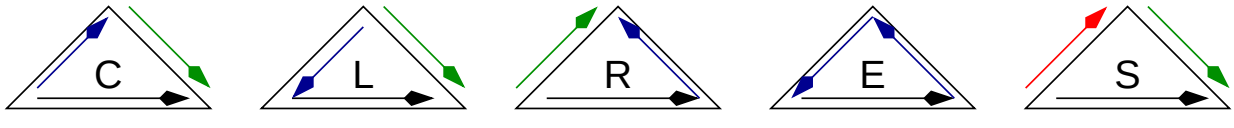
Wrapping: ❍ Jedes CLERS-Symbol erzeugt ein Dreieck

- Fälle 'C','L','E' und 'R' kennen alle beteiligten Punkte
- für Fall 'S' muß dritter Eckpunkt ausfindig gemacht werden

Idee: ❍ Erzeuge im Fall 'S' zunächst nur die Topologie („freie Punkte")

- Rekonstruiere 'S'-Vertex wenn Topologie (lokal) vollständig ist
- ***Freie Kanten*** zeigen an, wie doppelte Kanten zusammengefasst werden

Gerichtete, *freie Kanten*: Kanten, die bei Dekodierung nicht active gate sind



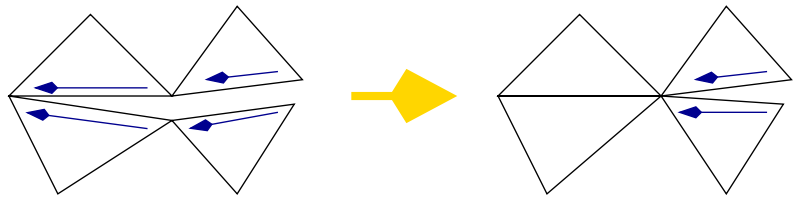
Active Gate (schwarz), nächstes Active Gate (grün) und freie Kante (blau)

Bei 'S' wird die rote Kante auf einen Stack gelegt

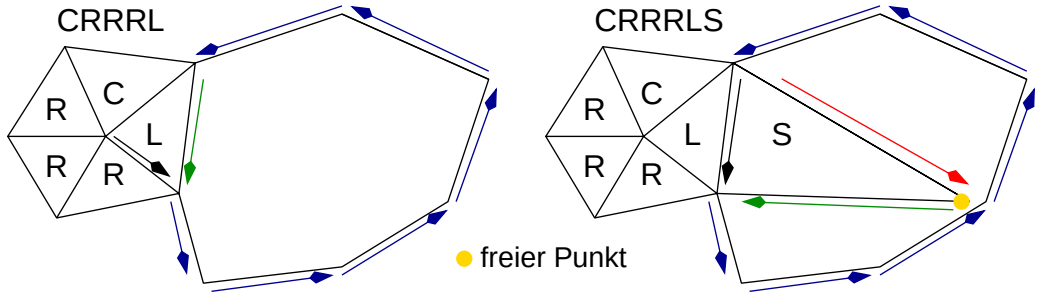
Wrap & Zip (Forts.)

Randkanten erhalten frei Kanten im Gegenuhrzeigersinn

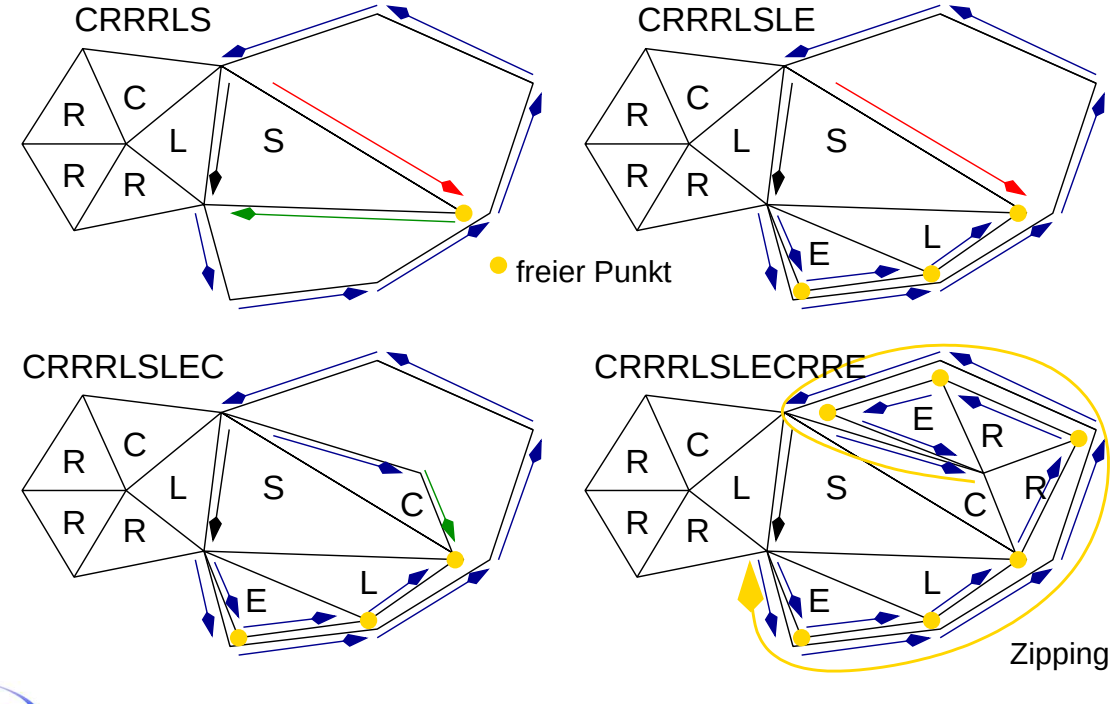
Zipping: Zusammenführung zweier Kanten in einem Punkt mit Zipping-Richtung auf diesen Punkt (nur nach 'L' und 'E')



Beispiel: Wrap & Zip



Beispiel: Wrap & Zip (Forts.)

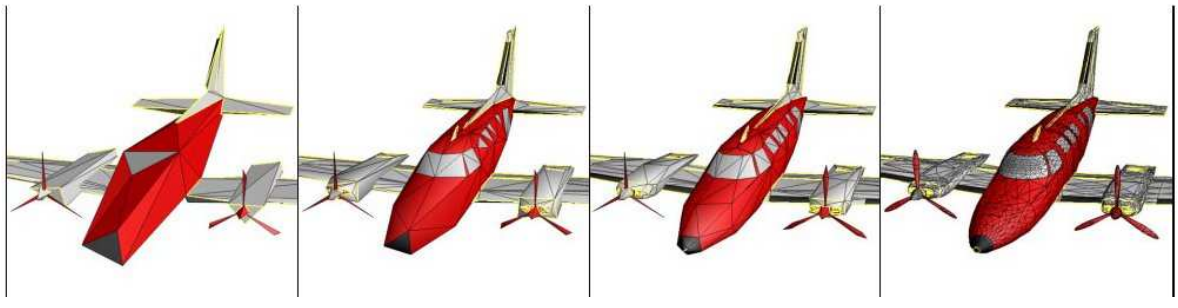


Motivation: Progressive Meshes

Edgebreaker Algorithmus:

+ Kompression für Dreiecksnetze
+ Streamingfähiges Format
– kein progressives Format, d.h. Übertragung der Daten bzgl. der feinsten Auflösung und damit kein LOD

Progressive Meshes: Progressiv, streamingfähig, keine Kompression



Grobe Auflösung $\mathcal{M}^{75}$ (links) und weitere Verfeinerungen $\mathcal{M}^{175}, \mathcal{M}^{425}, \mathcal{M}^{6500}$

6.2 Progressive Meshes (PM) ...

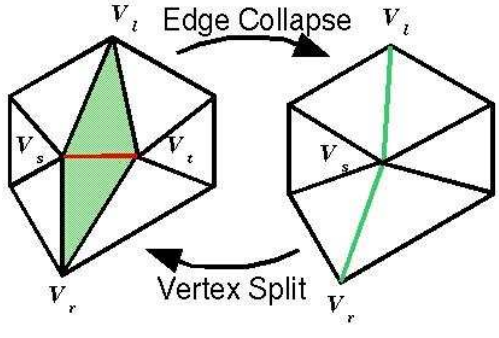
Idee: PM-Ansatzes

Ausgangslage: Hochaufgelöstes Polygonmodell $\mathcal{M}^n$ mit n Vertices

Reduktion: Sukzessives Entfernen von Vertices durch *Edge-Collapse*

Es entstehen Modelle $\mathcal{M}^k,\ k = n-1, n-2, \ldots, m$

Verfeinerung: Umkehrung des Edge-Collapse (*vertex-split*), d.h. $\mathcal{M}^{k+1}$ aus $\mathcal{M}^k$ restaurierbar.



PM-Datenstruktur

Struktur zur Speicherung eines PM besteht aus:

1. einfache Struktur zur Speicherung von $\mathcal{M}^m$
2. Differenzdaten D_k für den k-ten Split $k = m, \ldots, n-1$:

i^k	Index des zu splittenden Punktes
i_l^k, i_r^k	Indizes der angrenzenden Punkte
$\mathbf{V}_s^k, \mathbf{V}_t^k$	Koordinaten des gesplitteten Punkts

Damit entsteht eine Sequenz:

$$\mathcal{M}^m \xrightarrow{D_m} \mathcal{M}^{m+1} \xrightarrow{D_{m+1}} \mathcal{M}^{m+2} \ldots \mathcal{M}^{n-1} \xrightarrow{D_{n-1}} \mathcal{M}^n$$

Erzeugung erfolgt durch $n - m$ Edge-Collapse Operationen aus $\mathcal{M}^n$

Algorithmus zur Mesh-Reduktion

Kostenfunktion bewertet die Verformung aufgrund von Edge-Collapse

Input: Fein aufgelöstes Dreiecksmodell

Algorithmus: Erzeugung der PM-Datenstruktur

1. Ermittlung der Kostenfunktion $E(\mathcal{M}_e^n)$, $\forall e \in \mathcal{M}^n$
2. Sortierung der $E(\mathcal{M}_e^n)$ in einer Priority-Queue Q^n
3. Mesh-Reduktion:
   - 3.1. Edge-Collapse für die erste Kante e^* in Q^k
   - 3.2. Neue Kosten: In Nachbarschaft von e^* neu berechnen, sonst

$$E(\mathcal{M}_e^{k-1}) = E(\mathcal{M}_e^k)$$

   - 3.3. Q^{k-1}= Neusortierung von Q^k

6.2.1 Quadric Error Metrics (Garland & Heckbert '97)

Ziel: Kostenfunktion anhand des Abstandsfehler von $\mathbf{V}$

Initial: Vertex $\mathbf{V}$ ist Schnittpunkt der umliegenden Dreiecksebenen, wenn

$$E_i : a_i \cdot x + b_i \cdot y + c_i \cdot z + d_i = 0$$

$$\Leftrightarrow [a_i, b_i, c_i, d_i] \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

Alle umliegenden Ebenen mit $\vec{\mathbf{p}}_i = [a_i, b_i, c_i, d_i]$

$$E(\mathbf{V}) = \sum_i (\vec{\mathbf{p}}_i \cdot \mathbf{V})^2 = 0$$

Visualisierung von Vertexpositionen mit konstanten Kosten

Kostenfunktion Q ist eine quadratische Funktion, d.h.

$$E(\mathbf{V}) = \mathbf{V}^T Q \mathbf{V}$$

6.2.1 Quadric Error Metrics (Garland & Heckbert '97) ...

Herleitung der Kostenfunktionsmatrix Q

Umformulierung des quadratischen Ebenenabstands

$$E(\mathbf{V}) = \sum_i (\vec{\mathbf{p}}_i \cdot \mathbf{V}) \cdot (\vec{\mathbf{p}}_i \cdot \mathbf{V})$$

$$(\mathbf{V} \cdot \vec{\mathbf{p}}_i) \cdot (\vec{\mathbf{p}}_i \cdot \mathbf{V}) = (\mathbf{V}^T \cdot \vec{\mathbf{p}}_i) \cdot (\vec{\mathbf{p}}_i^T \cdot \mathbf{V}) = \mathbf{V}^T \underbrace{(\vec{\mathbf{p}}_i \cdot \vec{\mathbf{p}}_i^T)}_{=Q_i} \mathbf{V}$$

mit

$$Q_i = \vec{\mathbf{p}}_i \cdot \vec{\mathbf{p}}_i^T = \begin{bmatrix} a_i^2 & a_i b_i & a_i c_i & a_i d_i \\ b_i a_i & b_i^2 & b_i c_i & b_i d_i \\ c_i a_i & c_i b_i & c_i^2 & c_i d_i \\ d_i a_i & d_i b_i & d_i c_i & d_i^2 \end{bmatrix} \quad \text{und} \quad E(\mathbf{V}) = \vec{\mathbf{V}}^T \left(\sum_i Q_i \right) \mathbf{V}$$

Beachte: Q_i ist symmetrisch bzgl. Diagonale, damit auch $\sum_i Q_i$.

6.2.1 Quadric Error Metrics (Garland & Heckbert '97) ...

Kosten eines Edge-Collapse

Gegeben: Kante e zwischen $\mathbf{V}_s, \mathbf{V}_t$ mit Kostenmatrizen Q_s, Q_t

Gesucht: Position des gemergten Punktes $\mathbf{V}_s^*$ und zugehörige Kosten $E(\mathcal{M}_e^k)$

Vertex-Position: Kostenfunktion für gemergten Vertex

$$Q_s^* = Q_s + Q_t \Rightarrow E(\mathbf{V}_s^*) = (\mathbf{V}_s^*)^T (Q_s + Q_t)\mathbf{V}_s^*$$

- positioniere $\mathbf{V}_s^*$, so dass $E(\mathbf{V}_s^*)$ minimal: Mit $Q_s^* = (q_{ij}^*)_{i,j=1,\ldots,4}$, $q_{ij}^* = q_{ji}^*$

$$\frac{\partial E(\mathbf{V}_s^*)}{\partial \mathbf{V}_s^*} = 0 \quad \Leftrightarrow \quad \begin{bmatrix} q_{11}^* & q_{12}^* & q_{13}^* & q_{14}^* \\ q_{12}^* & q_{22}^* & q_{23}^* & q_{24}^* \\ q_{13}^* & q_{23}^* & q_{33}^* & q_{34}^* \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{V}_s^* = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

- Kostenfunktion: $E(\mathcal{M}_e^k) = E(\mathbf{V}_s^*)$ für optimale Position

Beachte: Addition der Kostenmatrizen bewertet angrenzende Dreiecke doppelt

Bewertung: Das Verfahren ist sehr schnell, aber tendenziell ungenau

6.2.2 Kostenfunktion nach Campgna etal. '98

Kostenfunktion

Ansatz: Durch Edge-Collapse soll möglichst wenig verändert werden $\Longrightarrow$ Bewertung durch eine Kostenfunktion

Quantitative Größen: Entferne eine Kante e „auf Probe" $\Longrightarrow \mathcal{M}_e^k$

Abstand $E_{dist}(\mathcal{M}_e^k)$: Räumlicher Abstand zwischen $\mathcal{M}^k$ und $\mathcal{M}_e^k$

Farbe $E_{col}(\mathcal{M}_e^k)$: Farbabstand korresponierender Punkte

Normalen $E_{norm}(\mathcal{M}_e^k)$: Normalenabweichung korresponierender Punkte

Diskontinuitäten $E_{disc}(\mathcal{M}_e^k)$: Veränderungen entlang scharfer Kanten

Gesamtfunktion:

$$E(\mathcal{M}_e^k) = E_{dist}(\mathcal{M}_e^k) + \alpha_{col} E_{col}(\mathcal{M}_e^k) + \alpha_{normal} E_{normal}(\mathcal{M}_e^k) + \alpha_{disc} E_{disc}(\mathcal{M}_e^k)$$

Beachte: Bestimme Koordinaten des gemergten Punktes, so dass $E(\mathcal{M}_e^k)$ minimal wird.

6.2.2 Kostenfunktion nach Campgna etal. '98 ...

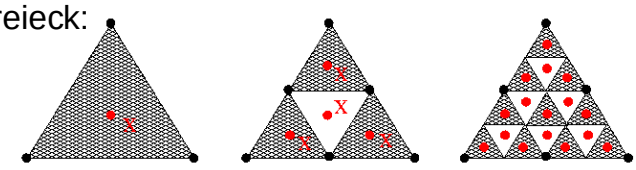
Kostenermittlung: Distanz

Sample-Punkte $\mathbf{X}_i$: Schwerpunkte einer regelmässigen Dreiecksunterteilung

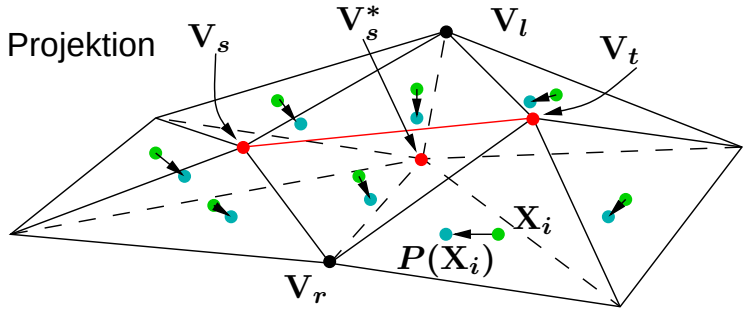
Abstandsbestimmung: Lokal Projektionen $P(\mathbf{X}_i)$ auf $\mathcal{M}_e^k$

Abstandskosten für $\mathbf{X}_i$ und $P(\mathbf{X}_i)$: $E_{dist}(\mathcal{M}_e^k) = \sum_i \|\mathbf{X}_i - P(\mathbf{X}_i)\|^2$

Sample-Points pro Dreieck:



Ermittlung der Projektion und von $\mathbf{V}_s^*$:



6.2.2 Kostenfunktion nach Campgna etal. '98 ...

Kostenermittlung: Distanz (Forts.)

Exakte Berechnung: Gleichzeitige Berechnung von $P(\mathbf{X}_i)$ und des gemergten Punktes $\mathbf{V}_s^*$ ist nicht-linear und schwierig!

Linearisierung der Berechnung von $\mathbf{V}_s^*$: Sequentielle Arbeitsweise:

1. ermittle Projektionen für fixes $\mathbf{V}_s^*$
2. variiere $\mathbf{V}_s^*$ für fixe Projektionen $P(\mathbf{X}_i) \in \Delta(\mathbf{V}_s^*, \mathbf{V}_i^1, \mathbf{V}_i^2)$:

$$P(\mathbf{X}_i) = a_i^s \mathbf{V}_s^* + a_i^1 \mathbf{V}_i^1 + a_i^2 \mathbf{V}_i^2,\ a_i^s + a_i^1 + a_i^2 = 1$$

Variation von $\mathbf{V}_s^*$: E_{dist} ist quadratisch in $\mathbf{V}_s^*$, denn:

$$\sum_i \Big\| \underbrace{\mathbf{X}_i - (a_i^1 \mathbf{V}_i^1 + a_i^2 \mathbf{V}_i^2)}_{=\mathbf{Y}_i} - a_i^s \mathbf{V}_s^* \Big\|^2 = \sum_i \|\mathbf{Y}_i\|^2 - 2a_i^s(\mathbf{Y}_i \cdot \mathbf{V}_s^*) + a_i^s \|\mathbf{V}_s^*\|^2$$

$\Rightarrow$ Optimumssuche ergibt lineares Gleichungssystem

6.2.2 Kostenfunktion nach Campgna etal. ’98 ...

Kostenermittlung: Andere Anteile

Nach Ermittlung von $\mathbf{V}_s^*, P(\mathbf{X}_i)$ werden andere Kostenanteile berechnet:

Farbe $E_{col}(\mathcal{M}_e^k)$:

1. $\mathbf{X}_i, P(\mathbf{X}_i)$ erhalten Farben via linearer Interpolation aus Farben der Ecken
2. Farbe von $\mathbf{V}_s^*$: Projektion auf $\mathcal{M}^k$ & lineare Interpolation
3. Kosten für Farben $\mathbf{C_1}, \mathbf{C_2}$ im HSV-Modell:

$$E_{col}(\mathbf{C_1}, \mathbf{C_2}) = |h_1 - h_2| + |s_1 - s_2| + 2 \cdot |v_1 - v_2|$$

Normalen $E_{norm}(\mathcal{M}_e^k)$:

1. Normalen bei $\mathbf{X}_i, P(\mathbf{X}_i)$ durch lineare Interpolation
2. Normale von $\mathbf{V}_s^*$: Projektion auf $\mathcal{M}^k$ & lineare Interpolation
3. Kosten für Normalen $\hat{\mathbf{n}}_1, \hat{\mathbf{n}}_2$:

$$E_{normal}(\hat{\mathbf{n}}_1, \hat{\mathbf{n}}_2) = 1 - (\hat{\mathbf{n}}_1 \cdot \hat{\mathbf{n}}_2)$$

6.2.2 Kostenfunktion nach Campgna etal. ’98 ...

Kostenermittlung: Andere Anteile (Forts.)

Diskontinuitäten $E_{disc}(\mathcal{M}_e^k)$:

1. Flag für Ecken auf Kanten
2. ist $\mathbf{V}_s$ oder $\mathbf{V}_t$ eine Kanten-Ecke, so wird $\mathbf{V}_s^*$ ebenso Kanten-Ecke und Abstand zwischen Kantenecken wird gemessen