# Computergraphik II

**Prof. Dr. Andreas Kolb**

**Fachgruppe Computergraphik und Multimediasysteme**

**Universität Siegen – Fachbereich 12**

Version: 9. November 2004

## 4.1 Grundlagen Polygon-Netze

**Motivation:** 1. Echtzeit-Graphik: Umwandlung Geometrien in Polygone/Dreiecke

2. Viele Geometrien liegen nur in Polygonform vor

**Begriffe:** *Polygon-Netze* bestehen aus

- Menge von *Punkten/Vertices*: $\mathcal{V} = \{\mathbf{V}_i\}, i = 1, \ldots, N_V$
- Menge von *Kanten*: $\mathcal{E} = \{\mathbf{E}_{ij}\}, \mathbf{E}_{ij} = \overline{\mathbf{V}_i\mathbf{V}_j}, i, j \in \{1, \ldots, N_V\}, N_E = |\mathcal{E}|$
- Menge von *ebenen Polygonen/Flächen*: $\mathcal{F} = \{\mathbf{F}_i\}, i = 1, \ldots, N_F$

Folgende Beziehungen gibt es zwischen diesen Entitäten:

**Adjazenz (angrenzend):** $\overline{\mathbf{V}_i\mathbf{V}_j}$ *grenzt an* Punkte $\mathbf{V}_i, \mathbf{V}_j$, Polygon $\mathbf{F}$ grenzt an entspr. Kanten und Punkte

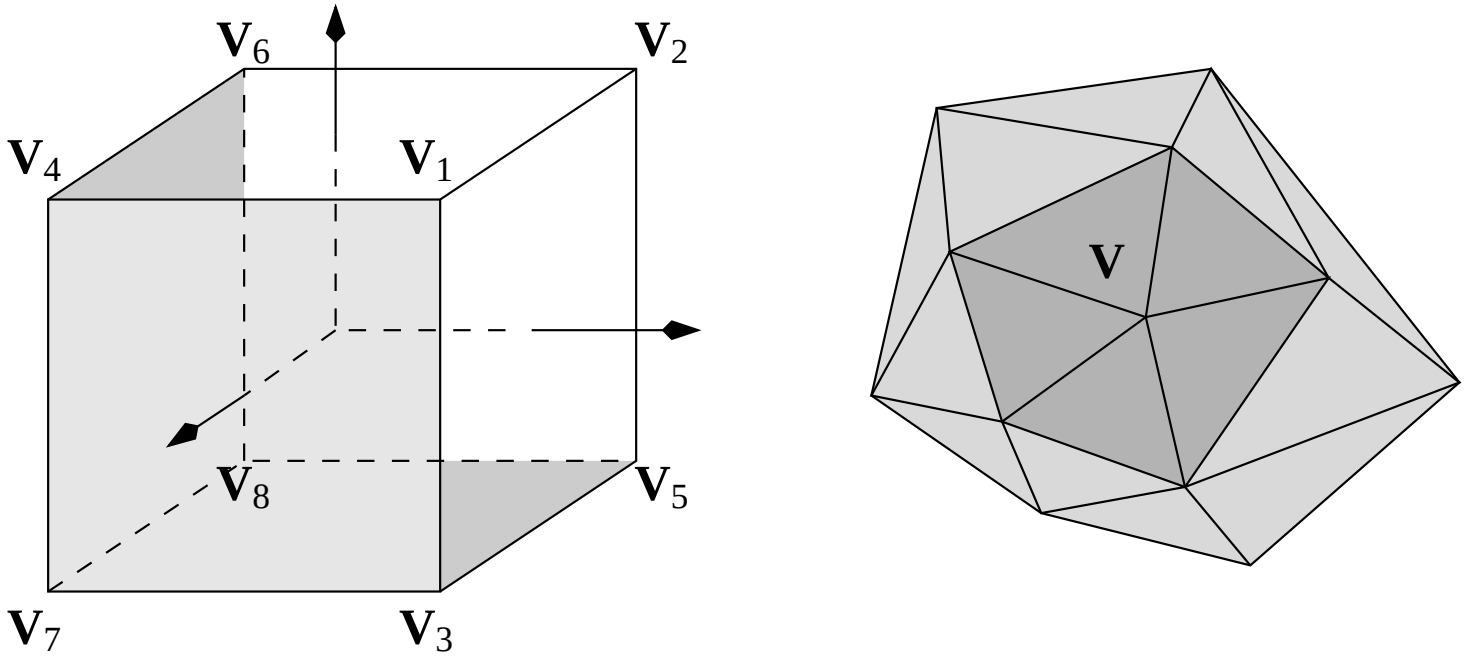
**Valenz:** Anzahl der Kanten, die an einem Punkt angrenzen

**$1$-Nachbarschaft von $\mathbf{V}$:** Alle (direkt) angrenzenden Polygone, Kanten und Punkte

**$n$-Nachbarschaft von $\mathbf{V}$:** Die $1$-Nachbarschaft der $(n-1)$-Nachbarschaft

**Beispiel: Würfel**

- Punkte: $\{(1,1,1),(1,1,-1),(1,-1,1),(-1,1,1),(1,-1,-1),(-1,1,-1),(-1,-1,1),(-1,-1,-1)\}$
- Kanten: $\mathcal{E} = \{\mathbf{E}_{1,2},\mathbf{E}_{1,3},\mathbf{E}_{1,4},\mathbf{E}_{2,5},\mathbf{E}_{2,6},\mathbf{E}_{3,5},\mathbf{E}_{3,7},\mathbf{E}_{4,6},\mathbf{E}_{4,7},\mathbf{E}_{5,8},\mathbf{E}_{6,8},\mathbf{E}_{7,8}\}$
- Flächen: $\mathcal{F} = \{\{1,3,5,2\},\{1,4,7,3\},\{1,2,6,4\},\{8,5,3,7\},\{8,6,2,5\},\{8,7,4,6\}\}$



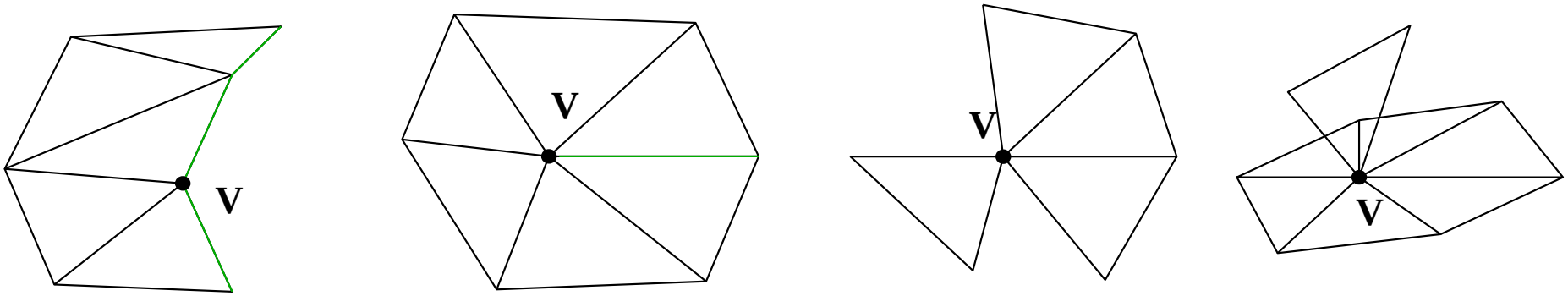

Links: Würfel mit 1-Nachbarschaft von $\mathbf{V}_7$; Rechts: Beispiel einer 1- und 2-Nachbarschaft

## 2-mannigfaltige Polygon-Netze

Ein *2-mannigfaltiges Polygon-Netz* hat folgende Eigenschaften:

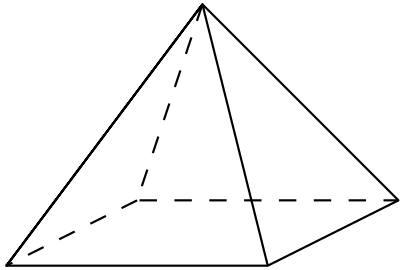
- keine Durchdringung, d.h. Schnitt zweier Polygone ist Punkt, Kante oder leer
- an jeder Kante liegen ein (*Randkante*) oder zwei Polygone (*innere Kante*) an
- Polygone um einen Punkt: *Offener (Randpunkt)* oder *geschlossener Fächer (innerer Punkt)*
- die Fläche ist orientierbar und besteht aus einer *Zusammenhangskomponente*
- und die *Euler-Formel* gilt: $\boxed{N_V - N_E + N_F = 2(1 - N_G) - N_B}$

  wobei $N_G$ *Anzahl Durchgangslöcher (Genus)*, $N_B$ *Anzahl der Rand-Linienzüge*

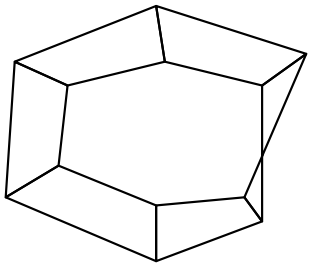



Links: Rand- und innerer Punkt bzw. Kante; Rechts: nicht-mannigfaltige Polygon-Netze

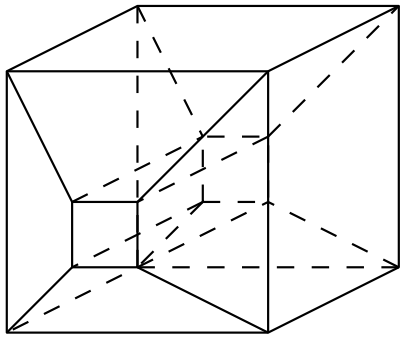
**Beispiel: Euler-Formel**



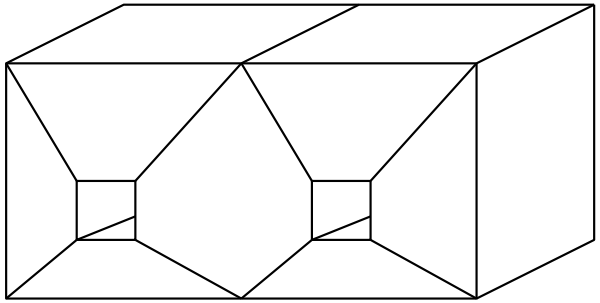
Pyramide
(ohne Boden)



Möbuis-Band



Würfel
(1 Durchgangsloch)



Quader
(2 Durchgangslöcher)

**Eulerformel:**

Pyramide ohne Boden:

$$N_P - N_E + N_F = 2 - 2 \cdot N_G - N_B$$
$$5 - 8 + 4 = 2 - 0 - 1$$

Würfel mit Durchgangsloch:

$$N_P - N_E + N_F = 2 - 2 \cdot N_G - N_B$$
$$16 - 32 + 16 = 2 - 2 - 0$$

Möbuisband:

$$N_P - N_E + N_F \stackrel{?}{=} 2 - 2 \cdot N_G - N_B$$
$$12 - 18 + 6 \neq 2 - 0 - 1$$

Quader mit 2 Durchgangslöcher:

$$N_P - N_E + N_F = 2 - 2 \cdot N_G - N_B$$
$$28 - 58 + 28 = 2 - 4 - 0$$

**Ziel:** Anwendungsoptimierte Repräsentationsstruktur, z.B.

- Effizientes Rendering: Ausschließlich Dreiecke, möglichst geringe Daten(-übertragung)
- Mesh-Editing: Effizienter Zugriff auf Polygondaten, z.B. Nachbarn

**Separate Speicherung** von *Geometrie* (Punkt-Koord.) und *Topologie* (Verbindungsdaten)

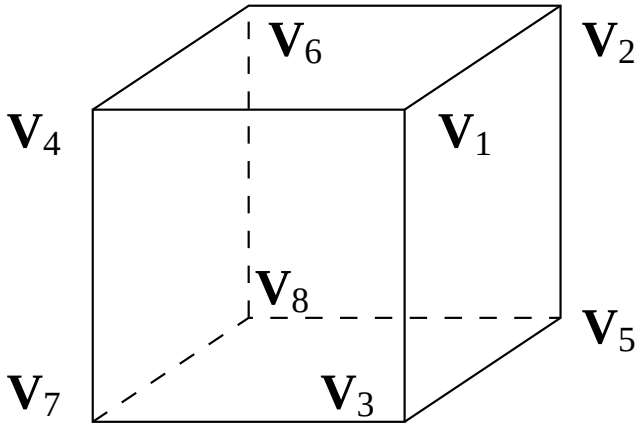
**Shared Vertex Format (auch: Indexed Face-Set)**

**Anwendung:** Rendering

**Ansatz:** Explizite Speicherung der Geometrie, Referenzierung für Topologie

**Optimierung in OpenGL:**

`GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`, `GL_QUAD_STRIP` oder Vertex-Arrays (CG I)




- Punkte: $\mathbf{V}_i = (x_i, y_i, z_i), i = 1, \ldots, 8$
- Flächen: $\mathcal{F} = \{\{1,3,5,2\}, \{1,4,7,3\}, \{1,2,6,4\}, \{8,5,3,7\}, \{8,6,2,5\}, \{8,7,4,6\}\}$

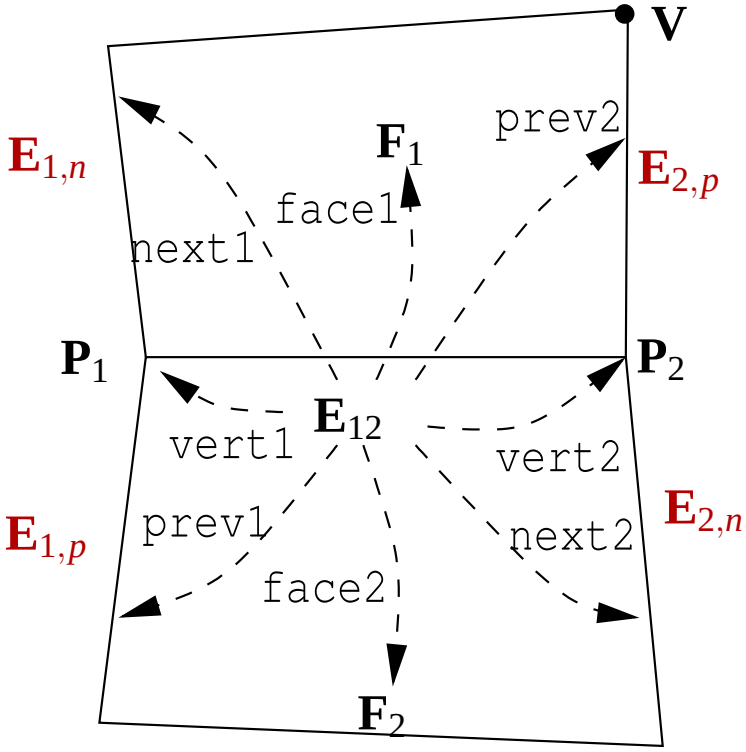
## Winged-Edge Repräsentation

Speicherung von Daten pro

**Punkt $\mathbf{V}_i$:** Koordinaten und Verweis auf eine Kante (`edge`)

**Polygon $\mathbf{F}$:** Verweist auf eine anliegende Kante (`edge`)

**Kante $\mathbf{E}_{ij}$:** Verwaltet Verweise auf die anliegenden Punkte (`vert1, vert2`) und Polygone und je zwei nächsten Kanten um Endpunkte (math. pos. orientiert) (`prev1, next1, prev2, next2`)




**Bemerkung:**

**Vorteil:** Schneller Zugriff auf Kanten und Ecken eines Polygons oder Kanten um eine Ecke

**Problem:** Orientierte Kante erzwingt if-Statement: Zugriff auf $\mathbf{V}$:

```
if( e_12->prev2->vert1 == e_12->vert2 ) v = e_12->prev2->vert2;
else                                    v = e_12->prev2->vert1;
```

## Half-Edge Datenstruktur

**Ansatz:** Wie Winged-Edge Datenstruktur, allerdings wird Kanteninformation geteilt

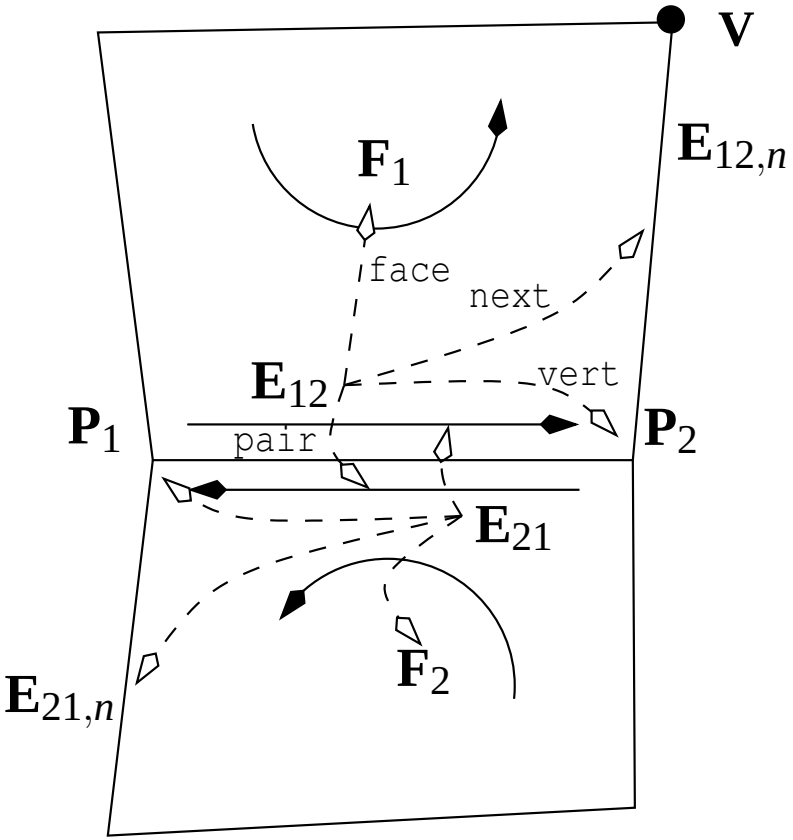
**Daten pro Halbkante:**

1. Verweis auf Endpunkt (`vert`)
2. Verweis auf andere Halbkante (`pair`)
3. Verweis auf zugehöriges Polygon (`face`)
4. Verweis auf nächste Halbkante (`next`)

**Zugriff auf $\mathbf{V}$** ohne if-Statement:

```
v = e_12->next->vert
```

**Beachte:** Zwei mehr Verweise pro Kante (zwei Halbkanten) als bei Winged-Edge durch `pair`



## Repräsentation konvexer Polygon-Netze

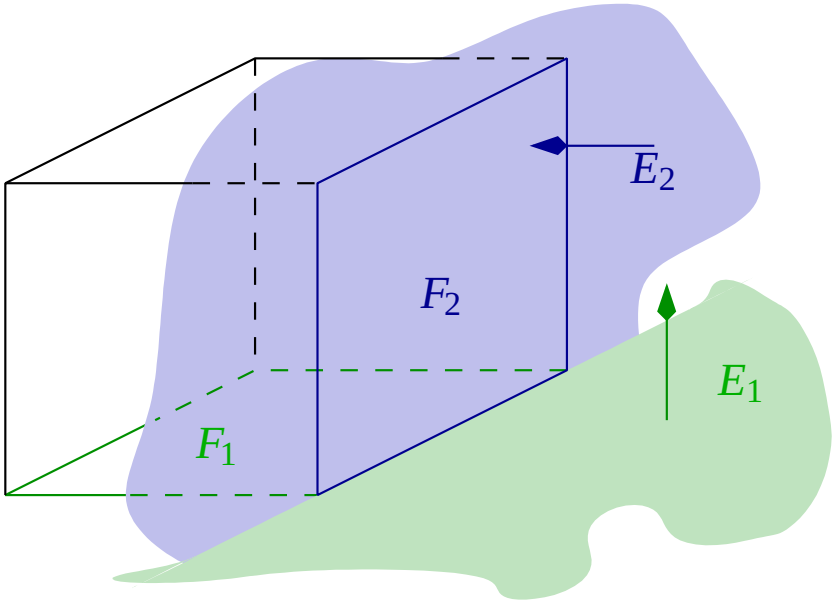
**Konvexe Polygon-Netze:** Geschlossene Polygon-Netze, die konvexe Menge $\mathcal{M}$ umschließen

Anwendungen: Hauptsächlich als Bounding-Volumen

**Repräsentation:** Für ein Polygon $\mathbf{F}_i \in \mathcal{F}$:

Polygon-Ebene $E_i : a_i p_x + b_i p_y + c_i p_z + d_i = 0,\ \mathbf{P} \in \mathbb{R}^3$

so dass *Halbraum* $\mathcal{H}_i = \{\mathbf{P} \ :\ f_i(\mathbf{P}) \geq 0\} \subseteq \mathcal{M}$; dann gilt: $\mathcal{M} = \bigcap_{i=1}^{N_F} \mathcal{H}_i$



**Anwendung: Beobachterabhängige Terrain-Modellierung**

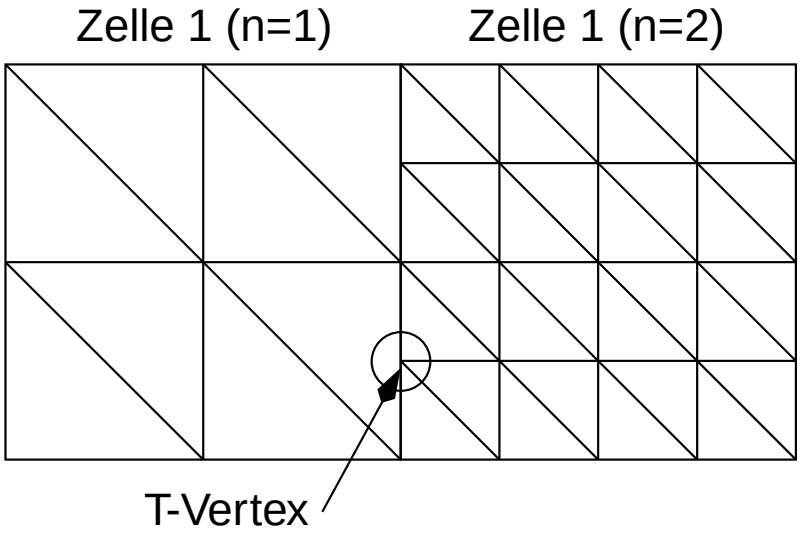
**Gegeben:** Höheninformation $h(u,v)$ über $(u,v)-$Ebene in beliebiger Genauigkeit

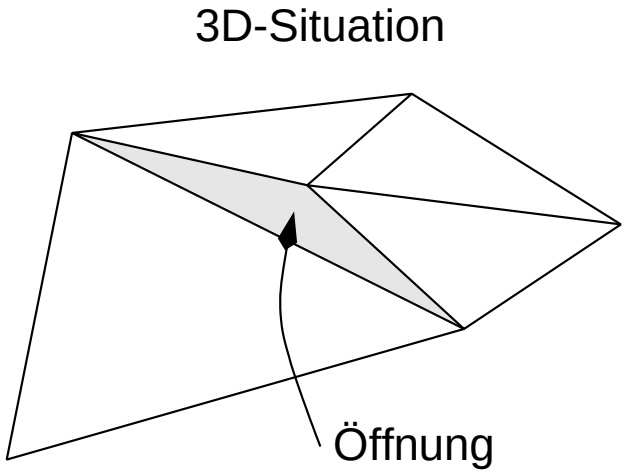
**Ziel:** Erzeugung eines Polygon-Netzes, abhängig von Beobachter-Position

**Ansatz:** 1. Festlegung von grober Auflösung $k \times k$ (*Zelle*)

2. $2^n \times 2^n$-Aufteilung jeder Zelle abhängig von Position und Orientierung des Beobachters

**Beachte:** Verschiedene Aufteilungen erzeugen u.U. *T-Vertices* $\Rightarrow$ Render-Problem



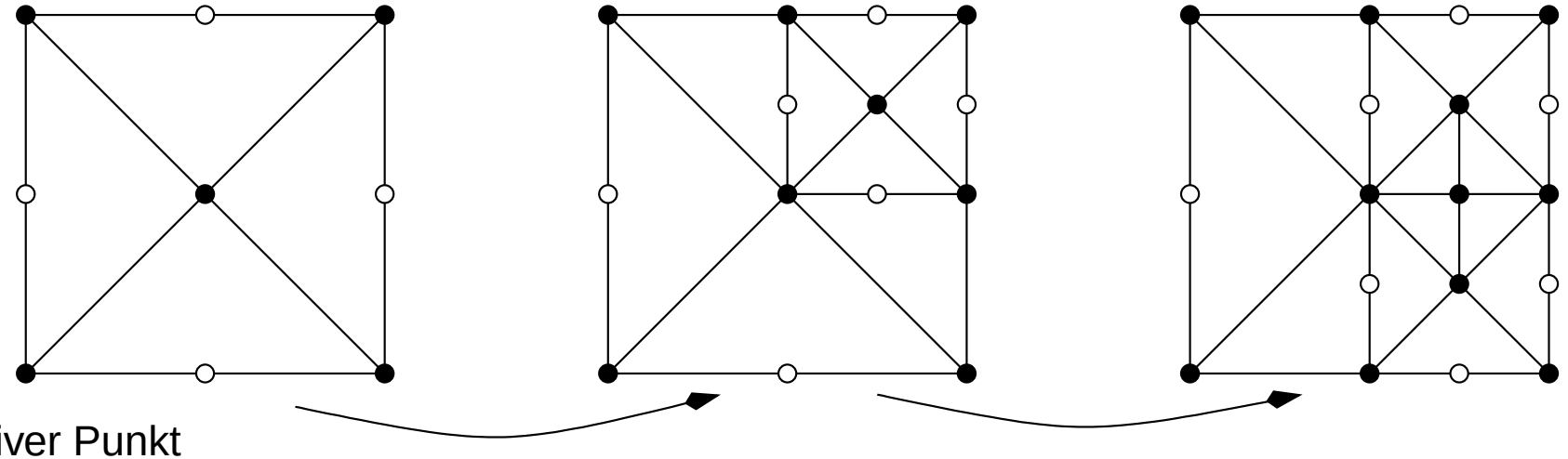



**Ansatz: Th. Ulrich, Gamasutra, 2000**

**Ausgangslage:** Eine Zelle mit *aktivierten (enabled)* Eckpunkten und Zentrum

**Unterteilungsregeln:**
- Aktiviertes Zell-Zentrum erzwingt aktive Zelleckpunkte
- Aktivierter Kanten-Mittelpunkt erzwingt aktives Zellen-Zentrum der Nachbarzelle

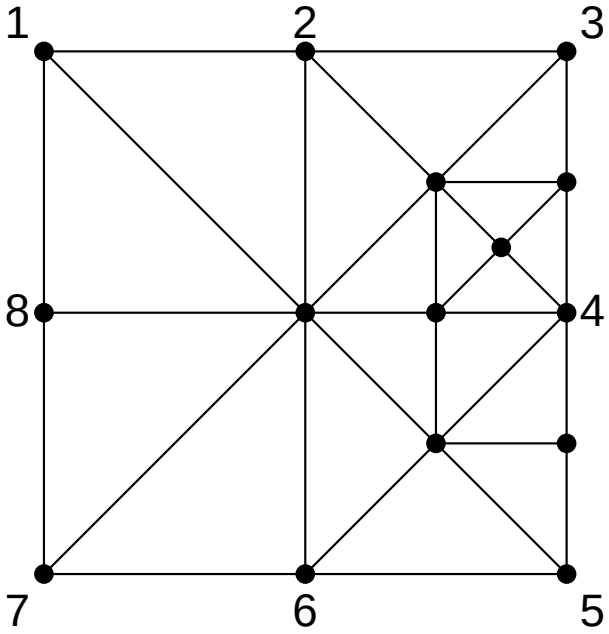
**Mögliche Entscheidungskriterien:**
- Abstand vom Beobachter
- Approximationsgüte des Höhenfelds $h$, z.B. Kante $\mathbf{E}_{ij}$ : $\left| h\left(\frac{\mathbf{V}_i+\mathbf{V}_j}{2}\right) - \frac{h(\mathbf{V}_i)+h(\mathbf{V}_j)}{2} \right| \geq \varepsilon$

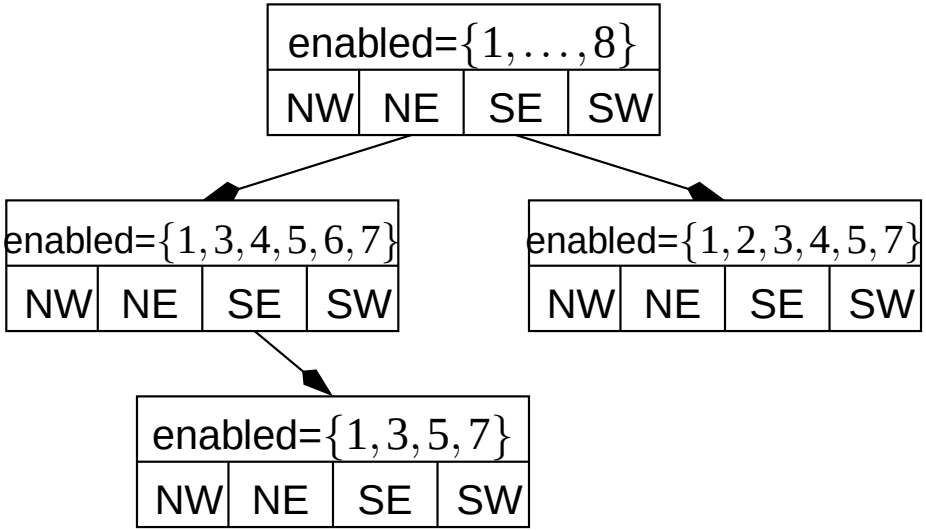



○ inaktiver Punkt
● aktiver Punkt

**Ansatz: Th. Ulrich, Gamasutra, 2000 (Forts.)**

**Speicherung** der Zell-Unterteilung in *Quad-Tree*



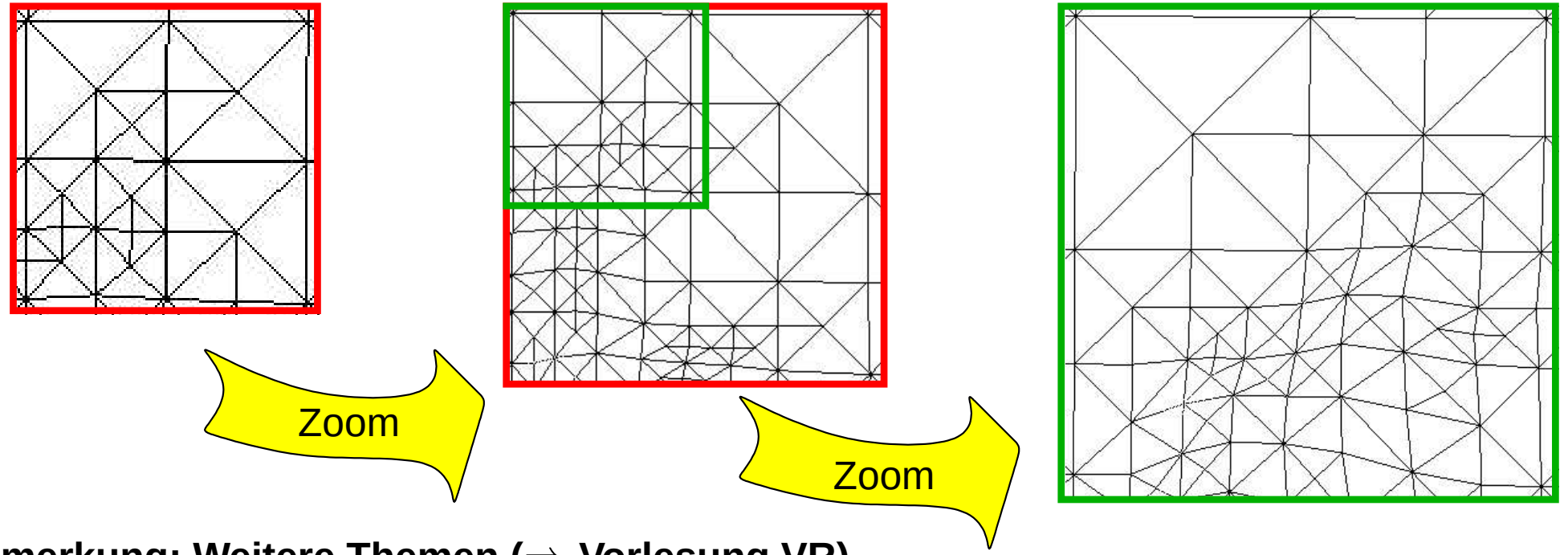

**Rendering** der Quad-Tree-Blätter mit `GL_TRIANGLE_FAN`, für Bereiche ohne Sub-Quad

**Quelle:** Thatcher Ulrich: *Continuous LOD Terrain Meshing Using Adaptive Quadtrees*

`www.gamasutra.com/features/20000228/ulrich_01.htm`

**Beispiel: Th. Ulrich, Gamasutra 2000**

Zoom-Folge von oben an ein Terrain:




**Bemerkung: Weitere Themen ($\Rightarrow$ Vorlesung VR)**

- Mesh-Optimierung, z.B. Glätten
- Mesh-Reduktion
- Mesh-Übertragung (Sequentialisierung)