

# Computergraphik

Prof. Dr.-Ing. Kerstin Müller

## Kapitel 7

# Polygone und Polygonale Netze

## Kapitelübersicht

- 7.1 Überblick
- 7.2 Polygone
- 7.3 Polygonale Netze
- 7.4 Datenstrukturen für polygonale Netze
- 7.5 Verfeinerung von Netzen

# Überblick

- Polygone und insbesondere Dreiecke als einfachste Vertreter stellen die elementarste Geometrie dar, die wir projizieren und rastern können.
- Dreiecke sind deshalb so beliebt, weil sie per Definition planar und konvex sind.

# Überblick

- Insbesondere bei Animationen, Virtual Reality und Computerspielen spielen sogenannte „Low-Poly-Modelle“ eine wichtige Rolle
  - Ziel ist es, mit möglichst wenigen Polygonen, bzw. Dreiecken ein rund und detailliert erscheinendes Modell zu konstruieren.
- Netze im CAD Kontext:
  - Wichtig zur visuellen Darstellung von Freiform-Geometrien, die durch eine mathematische Beschreibung gegeben sind.

# Überblick

- Diese Kurven oder Flächen werden durch ein Netz approximiert, um so die Fläche auf dem Bildschirm darzustellen.
- Dabei ergeben sich folgende Fragestellungen:
  - Welche Polygonauflösung benötigt man für eine genaue Darstellung?
  - Welche Polygonauflösung wird benötigt, um die stückweise planare Approximation glatt erscheinen zu lassen?

# Polygone

### ■ Definitionen:

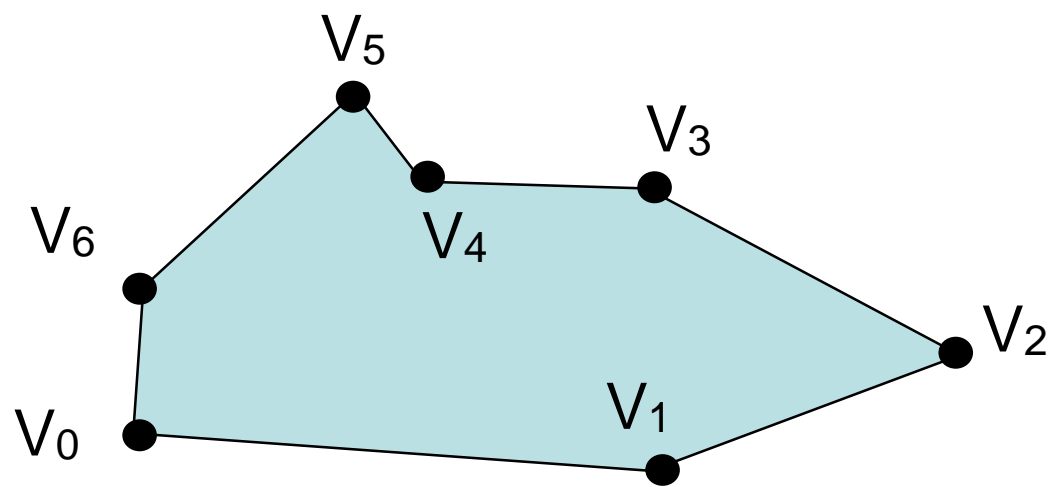
- Für eine Menge  $\{V_0, \dots, V_n\}$  von Punkten im zwei- oder dreidimensionalen Raum bezeichnen wir die Menge  $Q = \{(V_0, V_1), (V_1, V_2), \dots, (V_{n-1}, V_n)\}$  als Polygonzug oder Polyline.
- Die Paare von Ecken sind die Kanten des Polygonzugs.
- Ist  $V_n = V_0$ , (letzter Punkt stimmt mit dem ersten überein), sprechen wir von einem geschlossenen Polygonzug.
- Das von einem geschlossenen Polygonzug umrandete Gebiet nennen wir Polygon.

# Einfache und nicht-einfache Polygone

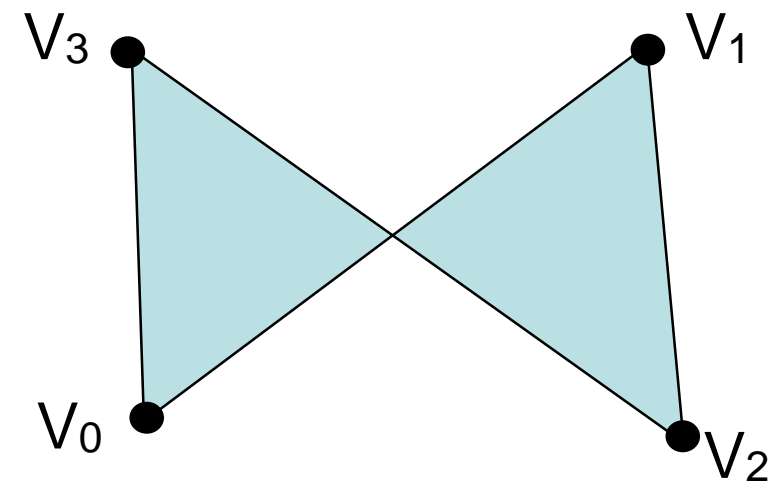
- Liegen alle Kanten eines Polygons in einer Ebene, wird das Polygon planar genannt.
- Ein einfaches Polygon liegt vor, wenn der Schnitt von jeweils zwei Kanten entweder die leere Menge ist, oder einer der Eckpunkte ist und jeder Endpunkt einer Kante höchstens zu zwei Kanten des Polygons gehört.
- Geschlossene einfache Polygone haben immer genauso viele Eckpunkte wie Kanten.

# Einfache und nicht-einfache Polygone

### ■ Beispiele:



geschlossenes, einfaches Polygon



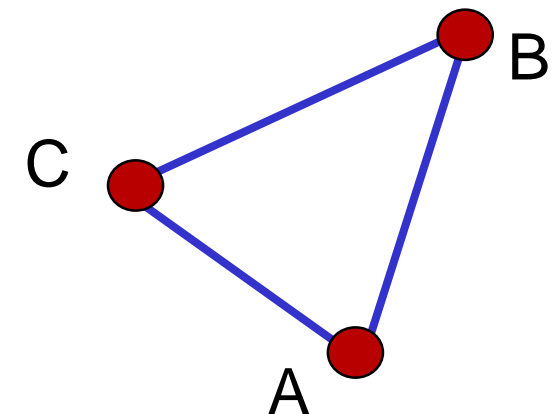
geschlossenes, nicht einfaches Polygon



# Unterscheidung Topologie / Geometrie

### ■ Topologie:

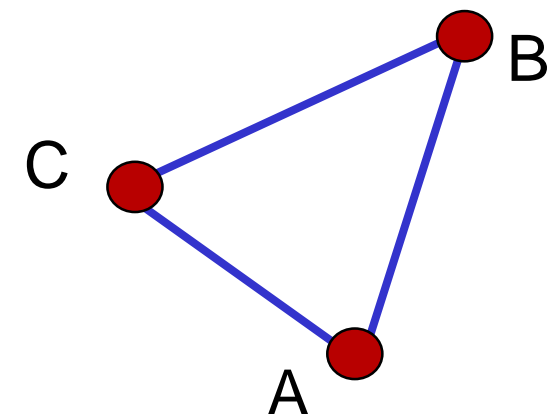
- Die Menge der Eigenschaften eines Objektes, die durch Starrkörpertransformationen (rigid motions, d.h. Rotation, Translation) nicht verändert werden - die Struktur des Modells.
- Im Beispiel: Das Polygon hat drei Ecken, die jeweils über Kanten in einer festgelegten Reihenfolge miteinander verbunden sind.
- Genauer: Die Reihenfolge ABC der Punkte ist gegen den Uhrzeigersinn angegeben (damit Vorderseite festgelegt) .



# Unterscheidung Topologie / Geometrie

## ■ Geometrie:

- Die „Instanzierung“ der Topologie durch Spezifikation der räumlichen Lage - die Form des Modells.
- Im Beispiel: Die Koordinaten der Eckpunkte.



# Eigenschaften von Polygonen

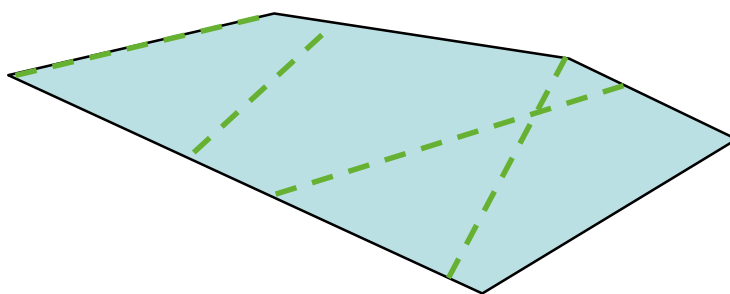
## ■ Topologische Eigenschaften von Polygonen

- Ein planares Polygon zerlegt die Ebene in mehrere Gebiete, mehrere Innere und ein äußeres, das unbegrenzt ist.
- Wichtig dabei ist die Durchlaufrichtung. Eine Kante wird in positiver Durchlaufrichtung durchlaufen, wenn dabei das Innere des Polygon links der Kante liegt.
- Der Rand wird also gegen den Uhrzeigersinn durchlaufen.

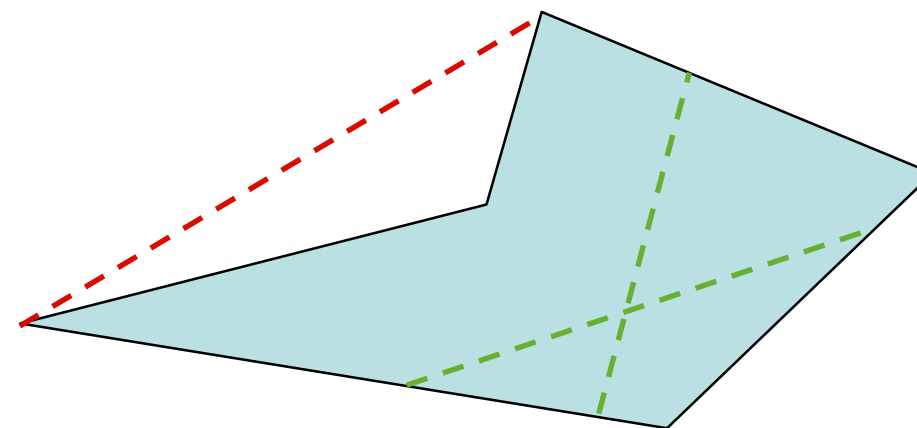
# Polygonale Netze

### ■ Viele Algorithmen setzen konvexe Polygone voraus:

- Definition: Ein Polygon ist genau dann konvex, wenn für zwei beliebige Punkte  $V$  und  $W$  auf dem Rand oder im Inneren des Polygons alle Punkte der Konvexkombination  $(1-\lambda)V + \lambda W$  mit  $\lambda \in [0,1]$  im Polygon liegen



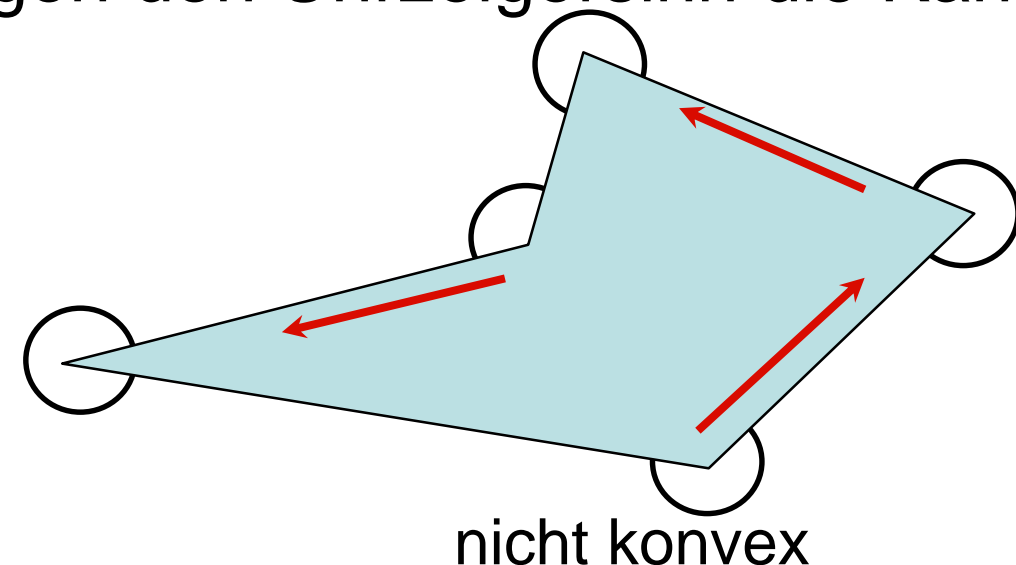
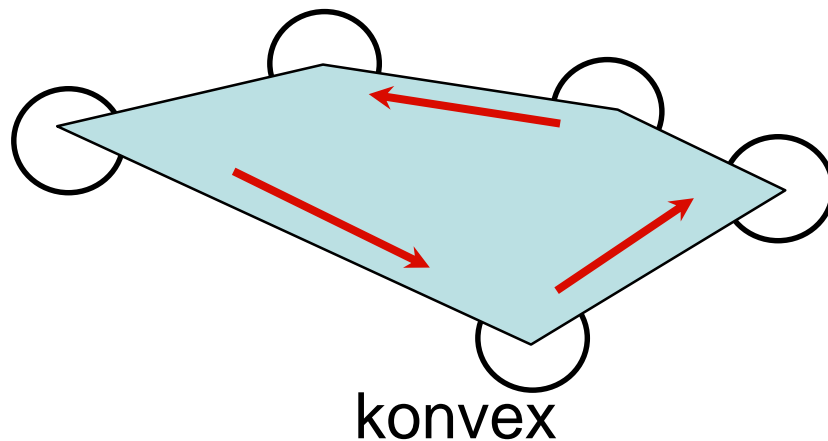
konvex



nicht konvex

# Anschauliche Deutung des Algorithmus

- "Wenn man mit dem Fahrrad die Aussenkanten des Polygons entlangfährt und dabei nur nach links oder nur nach rechts lenken muß, ist das Polygon konvex. Wenn man zwischendurch mal die Lenkrichtung wechseln muß, dann ist es konkav. Dabei ist es egal, ob man im Uhrzeigersinn oder gegen den Uhrzeigersinn die Kanten abfährt."



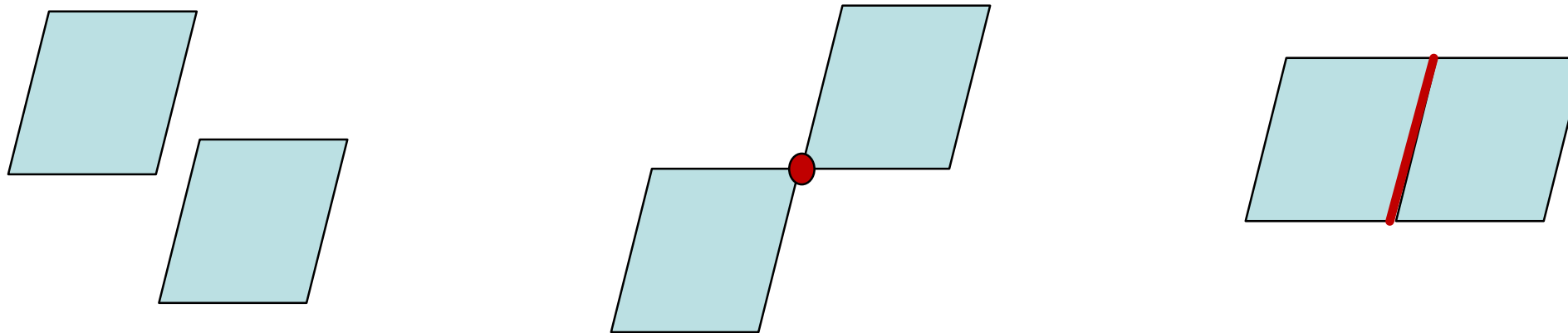
# Test auf Konvexität

### ■ Planares, einfaches Polygon auf Konvexität testen:

- (1) Nehme drei aufeinanderfolgende Punkte ( $V_i, V_{i+1}, V_{i+2}$ ) des Polyg.
- (2) Normalformen der Geradengleichungen der Kanten ( $V_i, V_{i+1}$ ) bilden, wobei die Normalen ins Innere des Polygons zeigen.
- (3) Berechne den orientierten Abstand des Punktes  $V_{i+2}$ .
- (4) Wiederhole Schritt (1) - (3) für jedes Tripel von aufeinanderfolgenden Punkten im Polygon.
- Hat der Abstand immer das gleiche Vorzeichen, ist das Polygon konvex.

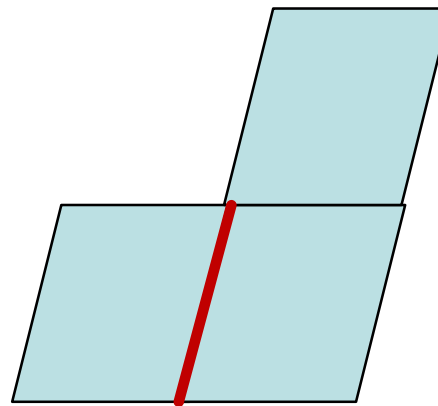
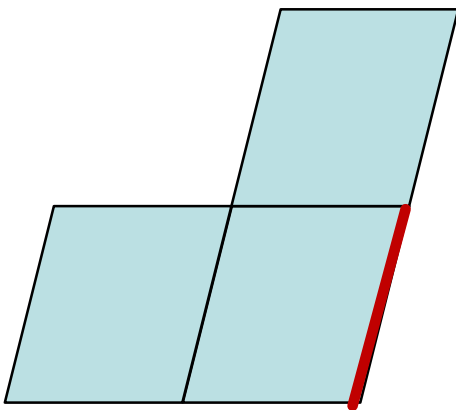
# Polygonale Netze

- Eine Menge von geschlossenen, planaren und einfachen Polygonen (= Facetten) bilden ein polygonales Netz, falls:
  - Je zwei Facetten haben entweder keinen Punkt oder eine Ecke oder eine ganze Kante gemeinsam. Der Schnitt zwischen zwei Facetten ist entweder leer, ein Eckpunkt oder eine Kante.



# Polygonale Netze

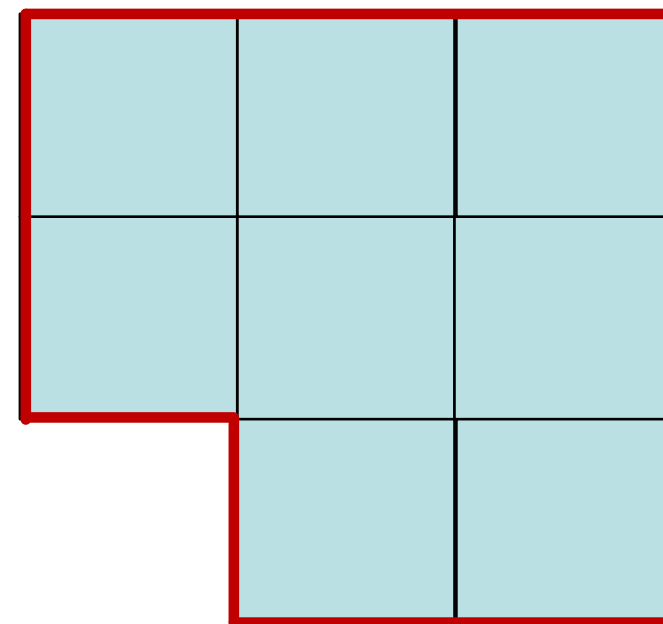
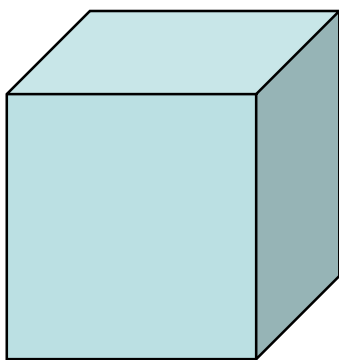
- Eine Menge von geschlossenen, planaren und einfachen Polygonen (= Facetten) bilden ein polygonales Netz, falls:
  - Jede Kante einer Facette gehört zu einer oder höchstens zwei Facetten.





# Polygonale Netze

- Eine Menge von geschlossenen, planaren und einfachen Polygonen (= Facetten) bilden ein polygonales Netz, falls:
  - Die Menge aller Kanten, die nur zu einer Facette gehören, ist entweder leer (Netz geschlossen) oder bildet einen geschlossenen und einfachen Polygonzug (=Rand des Netzes).



# Polygonale Netze

- Eine Menge von geschlossenen, planaren und einfachen Polygonen (= Facetten) bilden ein polygonales Netz, falls:
  - Je zwei Facetten haben entweder keinen Punkt oder eine Ecke oder eine ganze Kante gemeinsam. Der Schnitt zwischen zwei Facetten ist entweder leer, ein Eckpunkt oder eine Kante.
  - Jede Kante einer Facette gehört zu einer oder höchstens zwei Facetten.
  - Die Menge aller Kanten, die nur zu einer Facette gehören, ist entweder leer (Netz geschlossen) oder bildet einen geschlossenen und einfachen Polygonzug (=Rand des Netzes).

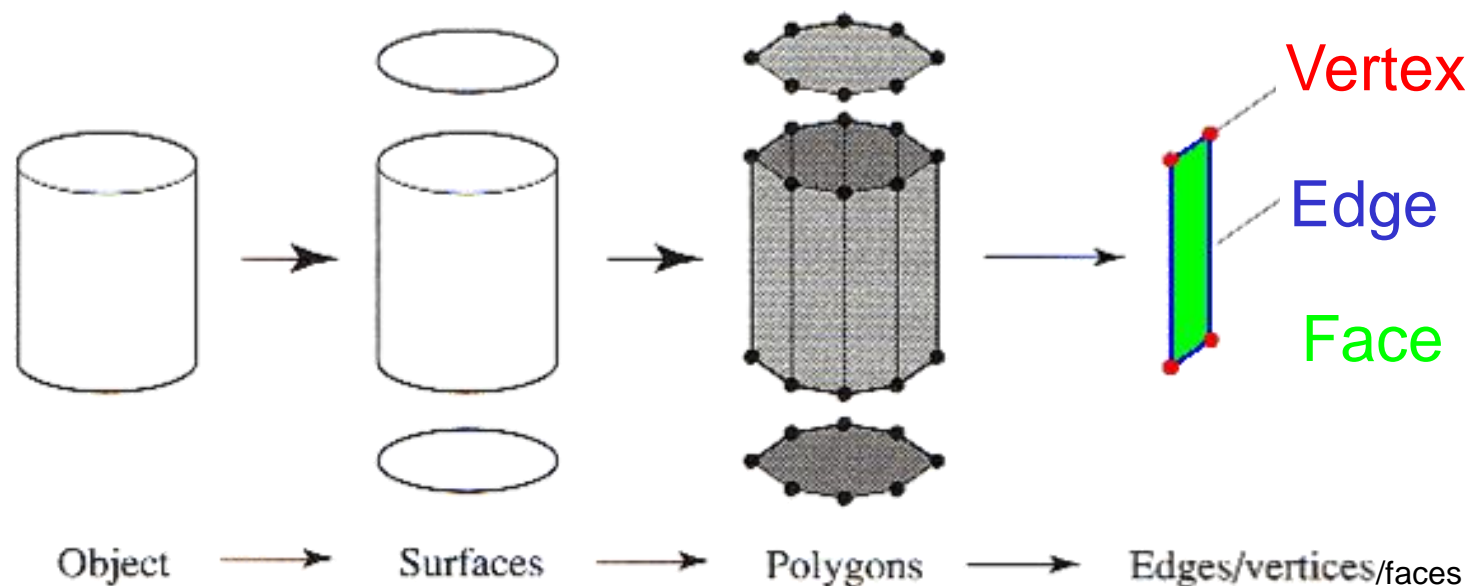
# Boundary Representation

- Allgemein: Darstellung eines dreidimensionalen Objektes durch die das Objekt begrenzenden Flächen, genannt **Boundary Representation**.
- Hier: Nur die einfachste Form der Boundary Representation, die polygonale Repräsentation.
  - Beschreibung eines Objektes durch eine Menge / ein Netz von planaren Polygonen / Facetten.
  - Oft nur Dreiecke, da diese per Konstruktion planar, einfach und konvex.

# Datenstrukturen für polygonale Netze

### ■ Repräsentationshierarchie (konzeptuell)

- Ein Objekt setzt sich aus Oberflächen zusammen
- Eine Oberfläche setzt sich aus Polygonen (**faces**) zusammen.
- Ein Polygon besteht aus Eckpunkten (**vertices**) und Kanten (**edges**).

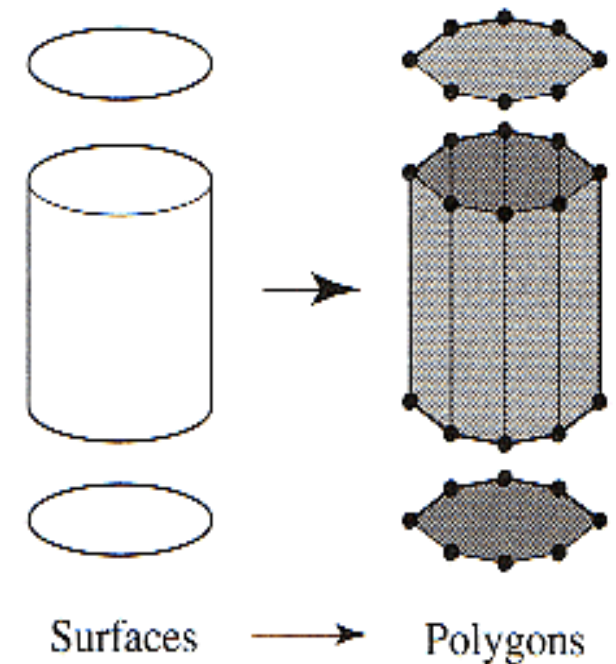


# Datenstrukturen für polygonale Netze

### ■ Offensichtlich existieren in der approximierenden polygonalen Darstellung zwei Arten von Kanten:

- Scharfe Kanten (Feature Lines), diese sollen als Kante sichtbar bleiben.
- Virtuelle Kanten, d.h. Kanten im Inneren glatter Flächen, diese Kanten sollten bei der visuellen Darstellung „verschwinden“.

### ■ Umsetzung: z.B. Mehrfach-Speicherung

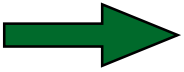


# Repräsentationshierarchie - Datenstruktur

- Was soll an polygonalen Netzen verändert werden?
  - Beseitigen von Mess- und Abtastfehlern durch Glätten.
  - Verfeinern um genauer zu modellieren und mehr Details darzustellen.
  - Netz-Verfeinerung soll lokal und adaptiv möglich sein.
  - Netze Ausdünnen zur Kompression oder zur Level of Detail Darstellung
    - Level of Detail Darstellung: z.B. bei der Datenübertragung erst eine grobe Darstellung übermitteln, dann durch weitere Übermittlungen verfeinern.
    - Entscheidend: Auswahlstrategie beim Entfernen von Polygonen, Rekonstruktion muss wieder möglich sein.

# Wahl der Datenstrukturen

### ■ Anforderungen:

- Trennung Informationen über Topologie und Geometrie?
- Wie Informationen über Ecken, Kanten und Faces speichern?
- Welche Operationen müssen zur Verfügung stehen?
  - Ecken, Kanten Faces auswählen, einfügen, löschen, verschweißen, rotieren, skalieren.
  - Ganze Netze verschweißen.
- Großen Menge Polygone:  Effizienz extrem wichtig.

# Topologische Operatoren auf Netzen

- Finde alle Kanten einer bestimmten Ecke!
- Finde alle Faces, die eine bestimmte Kante oder Ecke gemeinsam haben.
- Bestimme die Ecken, die durch eine Kante verbunden sind.
- Bestimme die Kanten einer Facette!



# Topologische Operatoren auf Netzen

- Bestimme alle Kanten die den Rand des Objekts bilden.
- Konsistenzprüfung:
  - Fehlt etwas? z.B. gibt es Kanten die keine Endpunkte haben?
  - Ist redundante Information enthalten? z.B. Doppelte Faces, doppelt in welchen Sinne (topologisch, geometrisch)?

# Flächen- / Kanten- / Punkt- Attribute

### ■ Zusätzlich zu Topologie und Geometrie verschiedene Attribute möglich:

- Flächen-Attribute: Dreieck?, Fläche(z.B.  $\text{mm}^2$ ), Flächennormale, Koeffizienten der Fläche (d.h. Ebenendefinition), konvex?, Löcher?
- Kanten-Attribute: Länge, Kante zwischen Polygonen oder Oberflächen (scharf/virtuell)?, Abschlusskante (Rand)?
- Eckpunkt-Attribute: beteiligte Polygone, Eckpunktnormale, Texturkoordinaten, geschätzte Krümmung (Biegung) der Fläche in diesem Punkt, ....

# Organisationsprinzipien

- Je nach Anwendungsfall können folgende Fälle auftreten:
  - regelmäßige Strukturen mit impliziter Topologie und Geometrie.
  - gemischte Strukturen mit impliziter Topologie und expliziter Geometrie.
  - „unorganisierte“ Daten mit expliziter Topologie und expliziter Geometrie - (allgemeine) polygonale Netze.
    - implizit: „ohne ausdrücklichen Hinweis in einer Aussage mit enthalten“.
    - explizit: „ausdrücklich, ausführlich“.

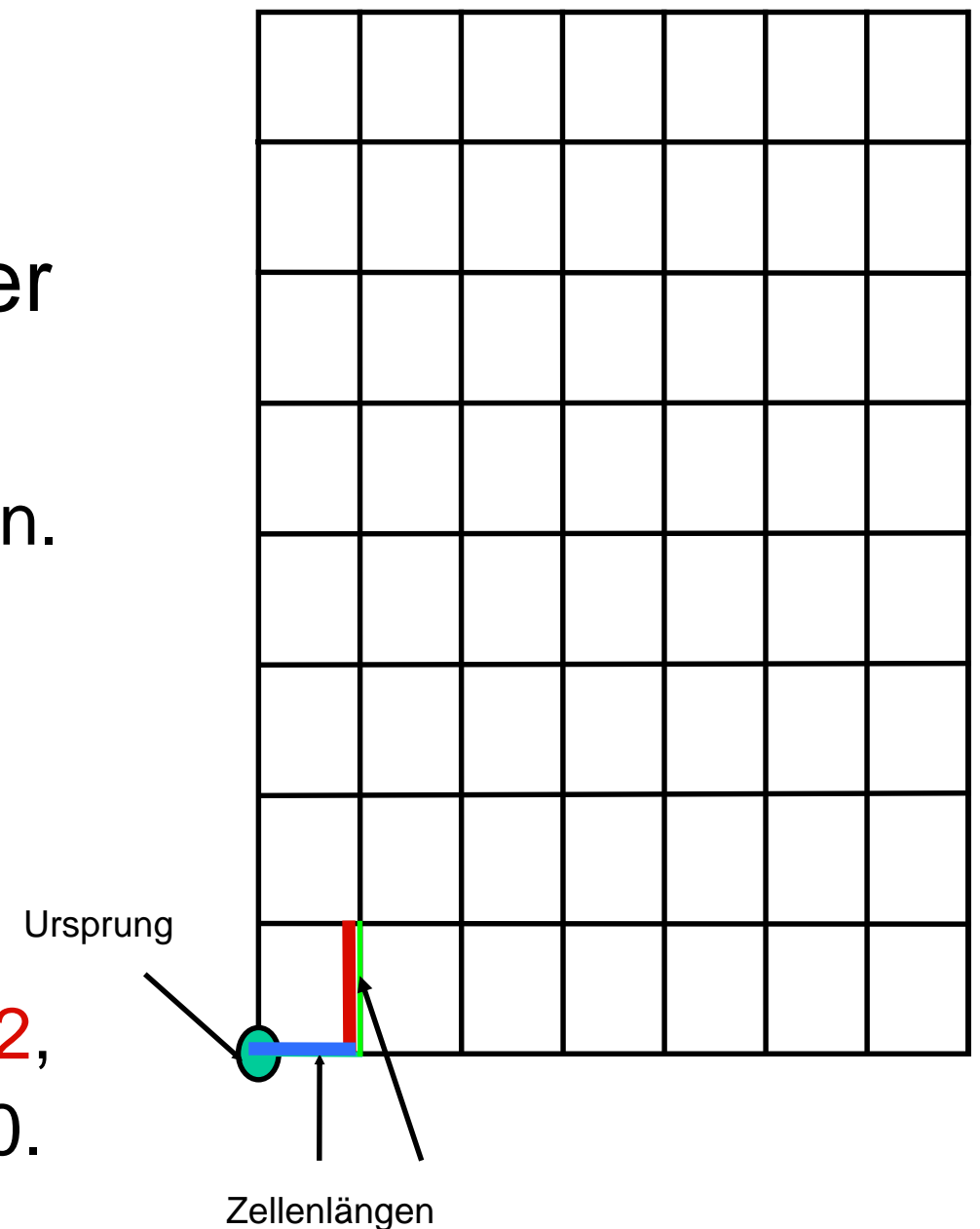
# Regelmäßige Strukturen

## ■ Rechtwinkliges, gleichmäßiges Gitter

- Topologie durch Anzahl der Punkte in x- und y-Richtung (ggf. z-Richtung) gegeben.
- Geometrie durch Angabe von Ursprung und Zellenlängen gegeben.

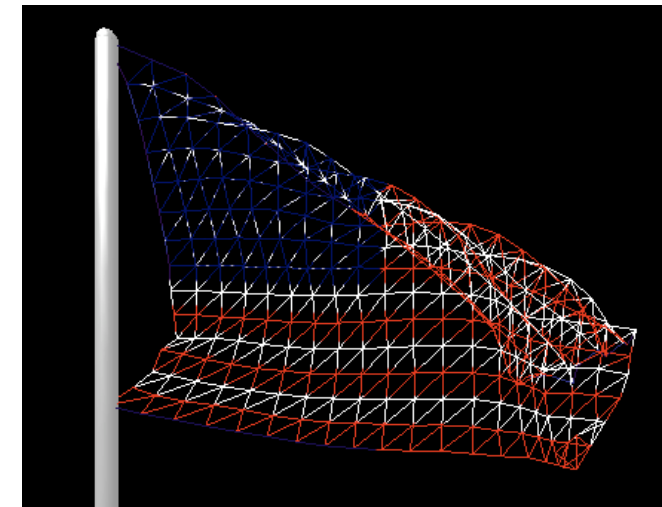
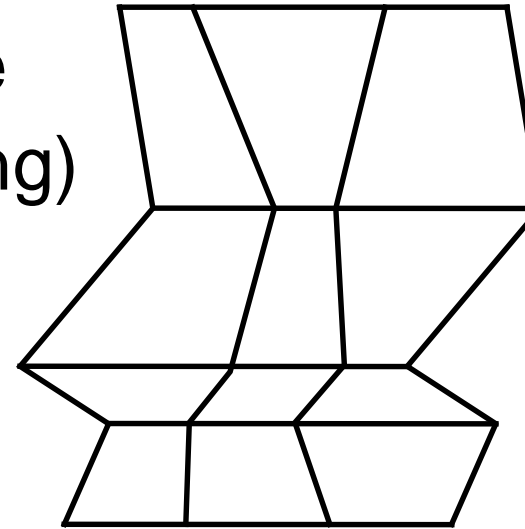
## ■ Beispiel:

- Ursprung =  $(-1, 1)$ ,  $x\text{-Dim} = 0.2$ ,  $y\text{-Dim} = 0.2$ , Anz. x-Richtung = 10, Anz. y-Richtung = 20.
- Koordinaten des rechten oberen Punktes?



# Gemischte Strukturen

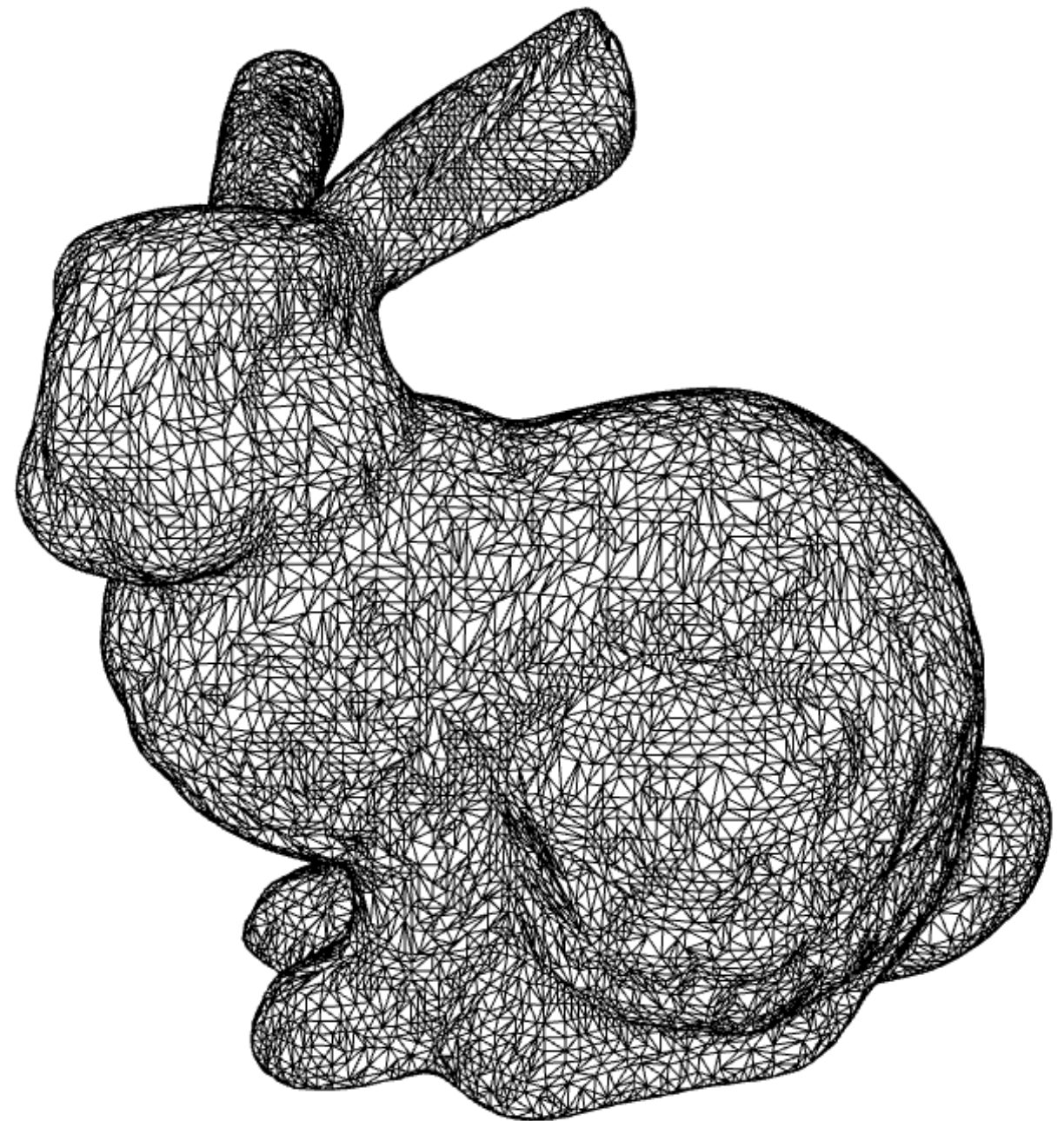
- Topologie durch Anzahl der Punkte in x- und y-Richtung (und z-Richtung) gegeben.
- Geometrie durch explizite Angabe der Punktkoordinaten gegeben.
- Beispiel: Simulation einer Flagge



aber keine adaptive  
Verfeinerung möglich

# Allgemeine polygonale Netze

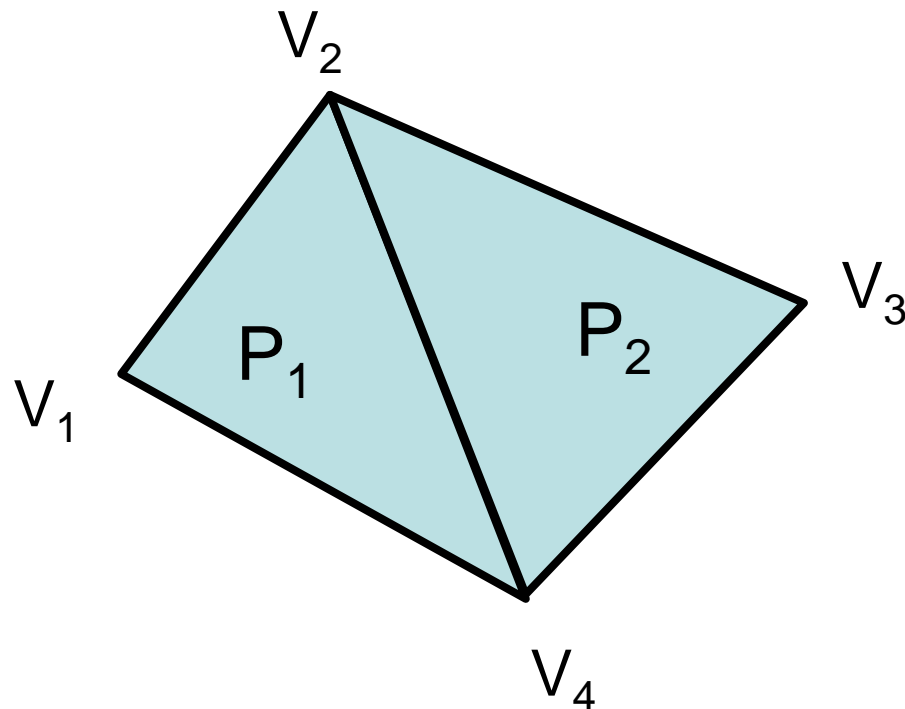
- Topologie und Geometrie muss explizit gespeichert werden.
- Problem ist die Topologie, denn die Geometrie ist „nur“ eine Liste von Punkt-Koordinaten.
- Möglichkeiten: explizite Speicherung, Eckenliste, Kantenliste, winged-edge-Struktur, half-edge Struktur.





# Explizite Speicherung

- Jedes Polygon wird durch eine Liste seiner Eckpunktkoordinaten repräsentiert.
- Zwischen jedem Paar von Ecken ist eine Kante, auch zwischen letzter und erster Kante.



$$P_1 = ((V_{2x}, V_{2y}, V_{2z}), \\ (V_{1x}, V_{1y}, V_{1z}), \\ (V_{4x}, V_{4y}, V_{4z}))$$

$$P_2 = ((V_{4x}, V_{4y}, V_{4z}), \\ (V_{3x}, V_{3y}, V_{3z}), \\ (V_{2x}, V_{2y}, V_{2z}))$$

# Explizite Speicherung

- Speicheraufwendige Darstellung:
  - Die Koordinaten von Ecken werden mehrfach gespeichert.
- Es gibt keine Speicherung gemeinsamer Ecken und Kanten:
  - Wie findet man gemeinsame Kanten?
  - Wie findet man alle Kanten eines Vertex?
- Geometrie kann nicht unabhängig von der Topologie verändert werden, da gemeinsame Ecken gesucht werden müssen.



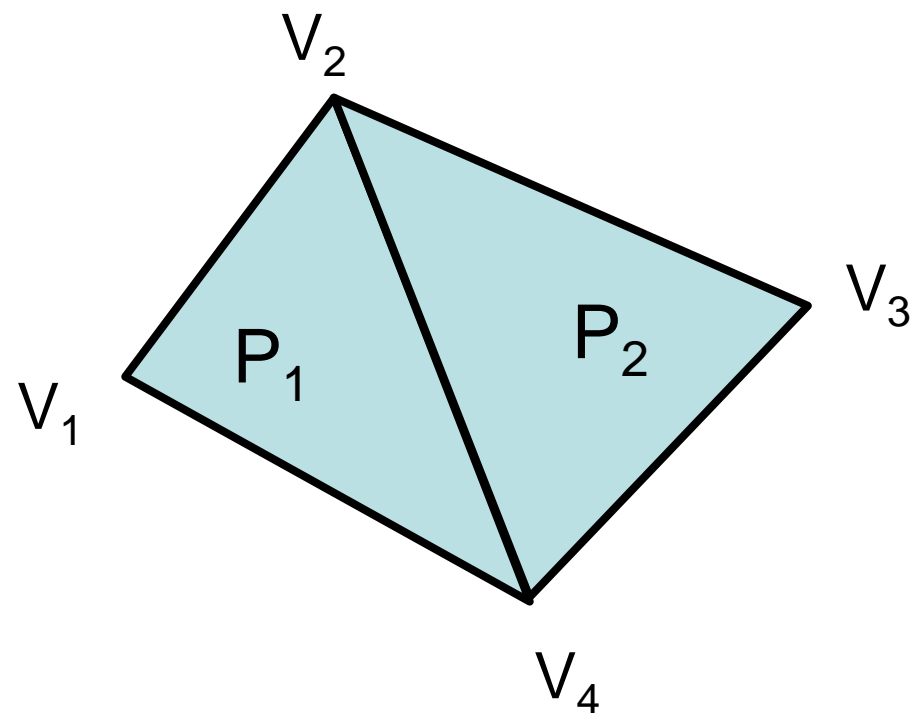
## Eckenliste

- Alle benötigten Punkte werden in einer Punktliste abgelegt
- Ein Polygon wird als Liste von Indices (Verweisen) in die Punktliste definiert

$$V = (V_4, V_2, V_1, V_3)$$

$$P_1 = (1, 4, 2)$$

$$P_2 = (3, 2, 4)$$



Auf Durchlaufrichtung achten!

# Eckenliste

- Jede Ecke wird nur einmal gespeichert.
- Die Geometrie kann unabhängig von der Topologie verändert werden.
- Gemeinsame Kanten und Ecken von Polygonen zu finden, ist immer noch schwer.
- Beim Zeichnen werden Kanten mehrfach dargestellt.
- Beliebt als Speicherformat, z.B. „.obj“-Format

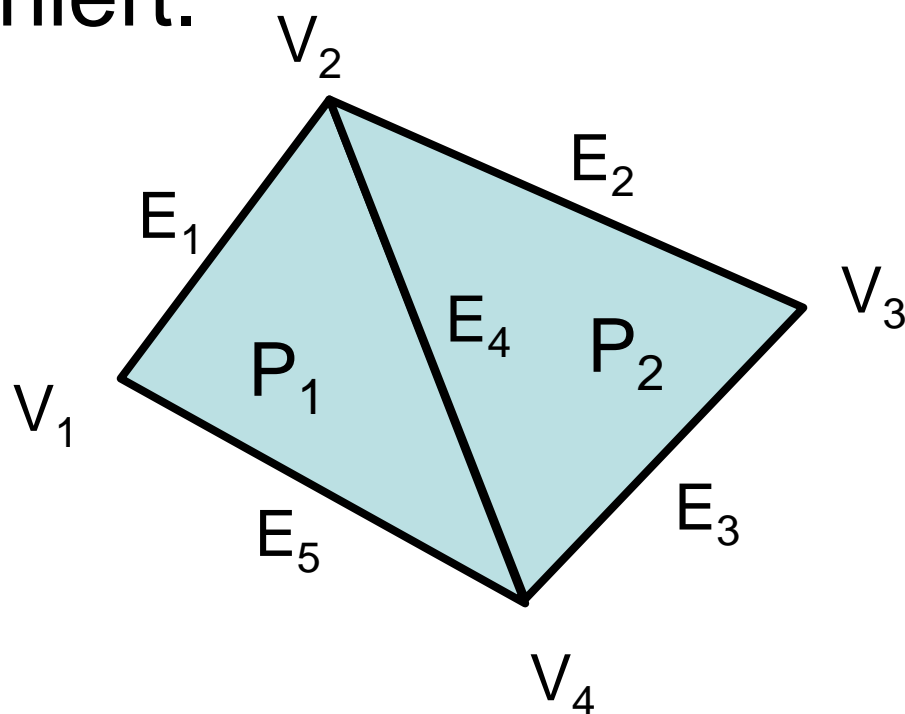
# Eckenliste im Fileformat „.obj“

```
# this is a comment
# Here is the first vertex, with (x,y,z) coordinates.
v 0.123 0.234 0.345
v ...
...
#Texture coordinates
vt ...
...

#Normals in (x,y,z) form; normals might not be unit.
vn ...
..
f v1/vt1/vn1 v2/vt2/vn2 v5/vt5/vn5
f ...
...
```

## Kantenliste

- Ecken in Punktliste, Kanten in einer eigenen Liste.
- Ein Polygon wird als Liste von Indices aus der Kantenliste definiert.



$$V = (V_1, V_2, V_3, V_4)$$

$$E_1 = (2, 1, P_1, N),$$

$$E_2 = (3, 2, P_2, N),$$

$$E_3 = (4, 3, P_2, N),$$

$$E_4 = (4, 2, P_1, P_2),$$

$$E_5 = (1, 4, P_1, N),$$

$$E = (E_1, E_2, E_3, E_5, E_4)$$

$$P_1 = (5, 4, 1), P_2 = (2, 4, 3)$$

Index des Startvertex

Index des Endvertex

Linkes Polygon

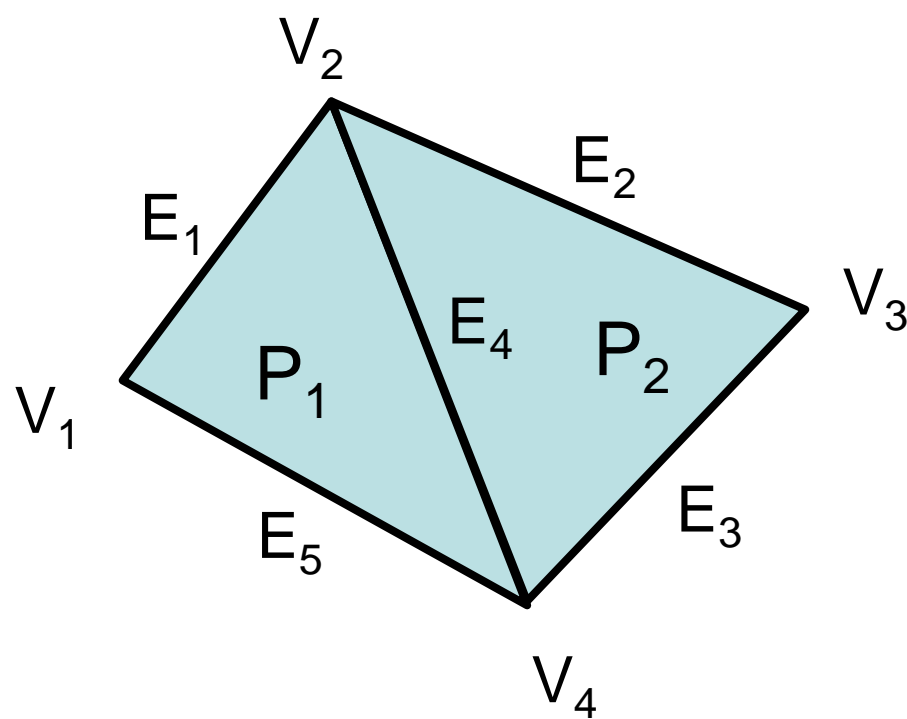
Rechtes Polygon

# Kantenliste

- Die Geometrie kann unabhängig von der Topologie verändert werden.
- Gemeinsame Kanten zu finden ist nun einfacher
  - Prüfung des zweiten Polygonindex in der Kantendefinition.
- Gemeinsame Ecken von Polygonen zu finden ist immer noch schwer.

## Kantenliste

- Beispiel: Finde alle Polygone, die zu  $P_1$  über eine Kante benachbart sind.



$$V = (V_1, V_2, V_3, V_4)$$

$$E_1 = (2, 1, P_1, N),$$

$$E_2 = (3, 2, P_2, N),$$

$$E_3 = (4, 3, P_2, N),$$

$$E_4 = (4, 2, P_1, P_2),$$

$$E_5 = (1, 4, P_1, N),$$

$$E = (E_1, E_2, E_3, E_5, E_4)$$

$$P_1 = (5, 4, 1), P_2 = (2, 4, 3)$$

gegeben  $P_1$ :

$E_5$  : prüfe 2. Index: N

$E_4$  : prüfe 2. Index:  $P_2$

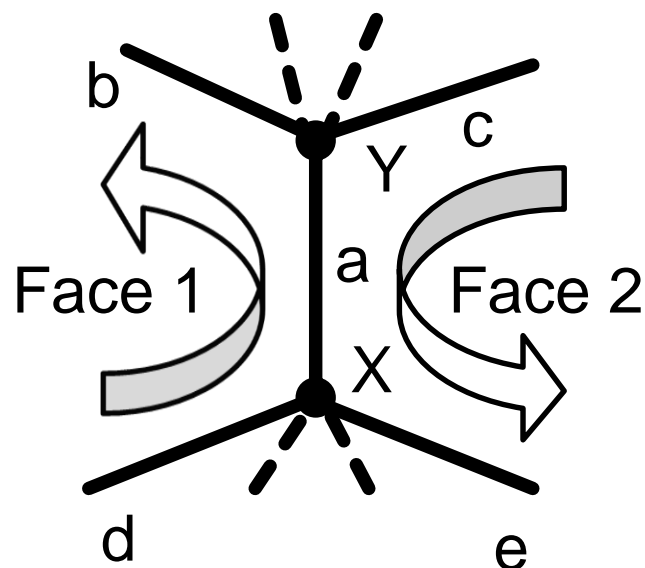
$E_1$  : prüfe 2. Index: N

## Operationen in der Kantenliste

Gegeben	Gesucht	Prozedur	$O(?)$
Ecke (Valenz $k$ )	Ausgehende Kanten	Alle $n$ Kanten nach Ecke absuchen, Kanten in beteiligten Polygonen suchen.	$O(n)$
	Nachbarecken	Vertices in ausgehenden Kanten.	$O(n)$
	Angrenzende Facetten	Polygone der ausgehenden Kanten.	$O(n)$
Kante	Endpunkte	Anfangs-/Endvertex in Kante.	$O(1)$
	Nachfolger/Vorgänger	Polygon ( $k$ -gon) ablaufen.	$O(k)$
	Angrenzende Facetten	Polygone in Kante.	$O(1)$
Facette ( $k$ -gon)	Eckpunkte	Vertices in Kanten.	$O(k)$
	Kanten	Kanten in Polygon.	$O(k)$
	Nachbarfacetten	Polygone der Kanten.	$O(k)$

# Doppelt verkettete Kantenliste

- Ecken, Kanten und Polygone wie in der Kantenliste
- Eine Kante hat Zeiger auf Nachfolger- und Vorgängerkanten in beiden angrenzenden Polygonen (winged edge).

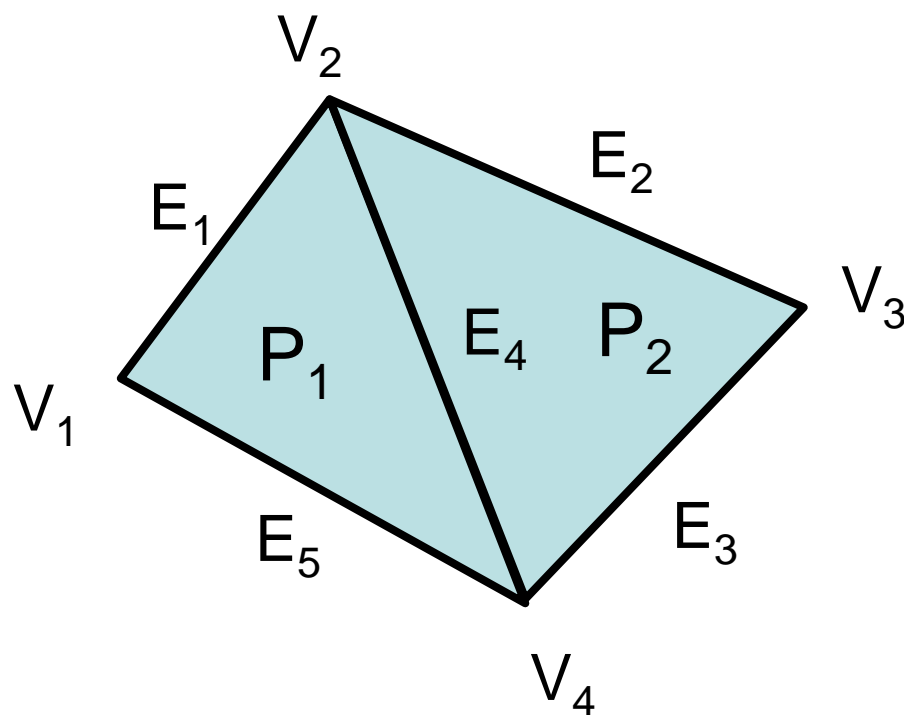


Kante	Punkte		Faces		Left Traverse		Right Traverse	
Name	Start	End	Left	Right	Pred	Succ	Pred	Succ
a	X	Y	1	2	d	b	c	e
...								



## Doppelt verkettete Kantenliste

- Ecken, Kanten und Polygone wie in der Kantenliste
- Eine Kante hat Zeiger auf Nachfolger- und Vorgängerkannten in beiden angrenzenden Polygonen (winged edge).



$$V = (V_1, V_2, V_3, V_4)$$

$$E_1 = (2, 1, P_1, N, 4, 5, N, N),$$

Vorg. links

$$E_2 = (3, 2, \quad \quad \quad),$$

Nachf. links

$$E_3 = (4, 3, \quad \quad \quad),$$

Vorg. rechts

$$E_4 = (4, 2, \quad \quad \quad),$$

Nachf. rechts

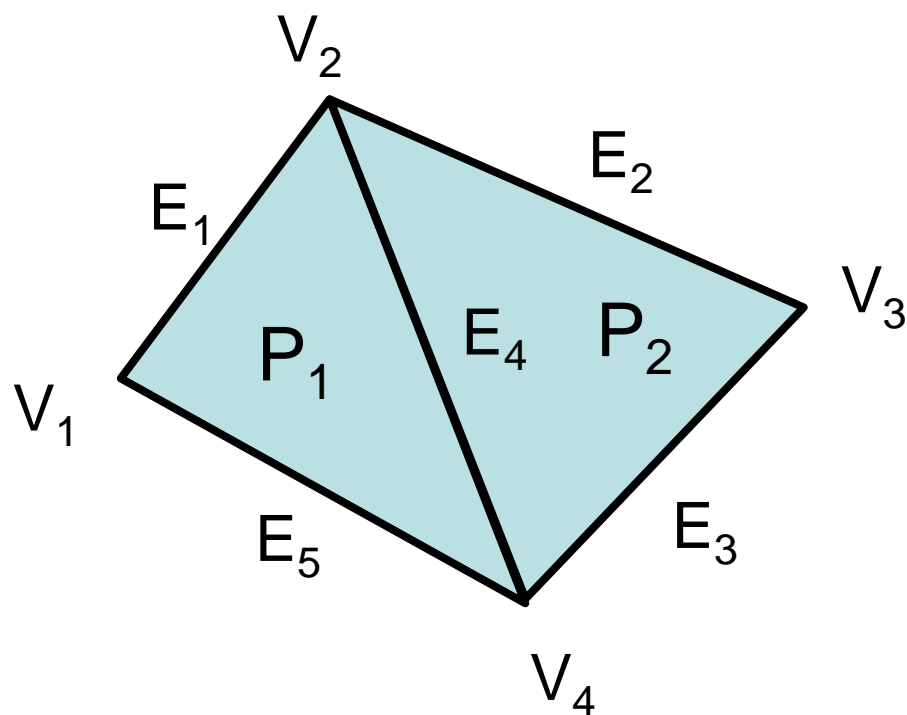
$$E_5 = (1, 4, \quad \quad \quad),$$

$$E = (E_1, E_2, E_3, E_5, E_4)$$

$$P_1 = (5, 4, 1), P_2 = (2, 4, 3)$$

# Doppelt verkettete Kantenliste

- Ecken, Kanten und Polygone wie in der Kantenliste
- Eine Kante hat Zeiger auf Nachfolger- und Vorgängerkannten in beiden angrenzenden Polygonen (winged edge).



$$V = (V_1, V_2, V_3, V_4)$$

$$E_1 = (2, 1, P_1, N, \text{4, 5, N, N}),$$

$$E_2 = (3, 2, P_2, N, \text{3, 4, N, N}),$$

$$E_3 = (4, 3, P_2, N, \text{4, 2, N, N}),$$

$$E_4 = (4, 2, P_1, P_2, \text{5, 1, 2, 3}),$$

$$E_5 = (1, 4, P_1, N, \text{1, 4, N, N}),$$

$$E = (E_1, E_2, E_3, E_5, E_4)$$

$$P_1 = (5, 4, 1), P_2 = (2, 4, 3)$$

Vorg. links

Nachf. links

Vorg. rechts

Nachf. rechts

# Doppelt verkettete Kantenliste

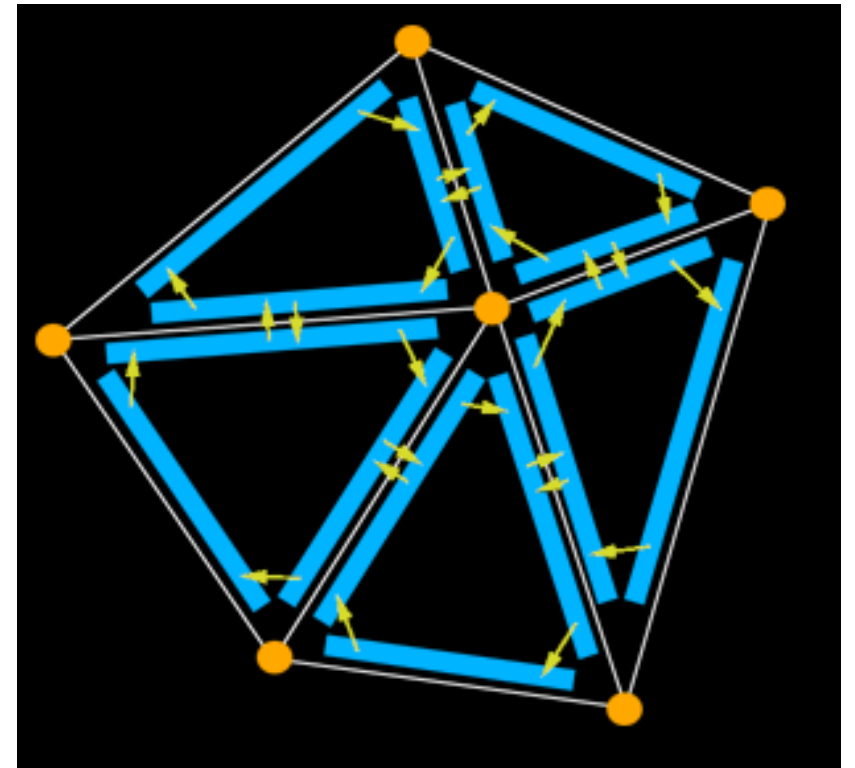
- Geometrie kann unabh. von der Topologie verändert werden.
- Neun Nachbarschaftsbeziehungen können abgefragt werden
  - Welche Facette, Kante oder Ecke gehört zu jeder Facette, jeder Kante oder zu jeder Ecke.
- Ecken und Facetten zu einer Kante zu finden ist mit konstantem Aufwand möglich.
- Alle Kanten oder Facetten zu einer Ecke zu finden ist immer noch aufwändig.

## winged-edge Struktur - Zusammenfassung

Gegeben	Gesucht	Prozedur	$O(?)$
Ecke (Valenz $k$ )	Ausgehende Kanten	Alle $n$ Kanten nach Ecke absuchen, über Nachfolger/Vorgänger handeln.	$O(n)$
	Nachbarecken	Vertices in ausgehenden Kanten.	$O(n)$
	Angrenzende Facetten	Polygone der ausgehenden Kanten.	$O(n)$
Kante	Endpunkte	Anfangs-/Endvertex in Kante.	$O(1)$
	Nachfolger/Vorgänger	Vorgänger/Nachfolger in Kante.	$O(1)$
	Angrenzende Facetten	Polygone in Kante.	$O(1)$
Facette ( $k$ -gon)	Eckpunkte	Vertices in Kanten.	$O(k)$
	Kanten	Kanten in Polygon.	$O(k)$
	Nachbarfacetten	Polygone der Kanten.	$O(k)$

# half-edge Darstellung

- Statt Kanten werden Halb-Kanten (half-edges) gespeichert.
  - Eine Kante ist ein Paar von entgegengesetzt gerichteten Halbkanten
  - Die Halbkanten, die ein Face begrenzen, bilden eine zyklisch verlinkte Struktur.
  - Die Orientierung kann mit oder gegen den Uhrzeigersinn laufen, muss aber auf der ganzen Struktur gleich gewählt werden.



# half-edge Darstellung

- Jede Halbkante speichert einen Zeiger auf ihren Nachfolger, auf den Partner und auf die Facette, die von ihr begrenzt wird und auf ihre ausgehende Ecke (in der Abbildung nicht gezeigt).
- Die Ecken der half-edge Datenstruktur speichern die x,y, und z-Koordinaten und einen Zeiger zu genau einer Halbkante, die diese Ecke als Start-Punkt benutzt.
  - Für eine Ecke gibt es in der Regel mehr als eine Kante zur Auswahl, es ist jedoch egal welche gewählt wird.
- Eine Facette speichert nur einen Zeiger auf eine der Halbkanten, die die Facette begrenzt.

# half-edge Darstellung - Datenstruktur

```
struct HE_edge {
    HE_vert* vert;    // start-vertex of the half-edge
    HE_edge* pair;    // oppositely oriented adjacent half-edge
    HE_face* face;    // face the half-edge borders
    HE_edge* next;    // next half-edge around the face
};

struct HE_vert {
    float x;
    float y;
    float z;

    HE_edge* edge;    // one of the half-edges emanating from the vertex
};

struct HE_face{
    HE_edge* edge;    // one of the half-edges bordering the face
};
```

# half-edge Darstellung - Nachbarschaften

- Anfangs- und End-Punkt einer gegebenen Kante finden.

```
HE_vert* vert1 = edge->vert;  
HE_vert* vert2 = edge->pair->vert;
```

- Alle Halb-Kanten eines Faces durchlaufen

```
HE_edge* edge = face->edge;  
  
do {  
    // do something with edge  
    edge = edge->next;  
} while (edge != face->edge);
```



# half-edge Darstellung - Nachbarschaften

- Laufe über alle Kanten, die zu einem gegebenen Vertex gehören.

```
HE_edge* edge = vert->edge;  
do {  
    // do something with edge, edge->pair or edge->face  
    edge = edge->pair->next;  
} while (edge != vert->edge);
```

## ■ Bemerkungen:

- Scharfe Kanten können als Rand-im-Inneren modelliert werden.
- Alle Beispiele verzichten auf NULL-Pointer Checks. Falls Netze mit Rand oder scharfen Kanten benutzt werden, müssen die Abfrage-Routinen entsprechend verändert werden.

# half-edge Struktur - Zusammenfassung

Gegeben	Gesucht	Prozedur	$O(?)$
Ecke (Valenz $k$ )	Ausgehende Kanten	Start-Kante und Partner der Vorgänger hangeln.	$O(k)$
	Nachbarecken	Vertices in Partner in ausgehenden Kanten.	$O(k)$
	Angrenzende Facetten	Polygone der ausgehenden Kanten.	$O(k)$
Kante	Endpunkte	Startecke in Halb-Kante und Partner.	$O(1)$
	Nachfolger/Vorgänger	Vorgänger in Kante und Vorgänger in $k$ -gon entlang hangeln für Nachfolger.	$O(k)$
	Angrenzende Facetten	Polygone in Halb-Kante und Partner.	$O(1)$
Facette ( $k$ -gon)	Eckpunkte	Vertices in Kanten.	$O(k)$
	Kanten	Vorgänger der Start-Kante hangeln.	$O(k)$
	Nachbarfacetten	Polygone der Partner in Kanten.	$O(k)$

# Verfeinerung von Netzen

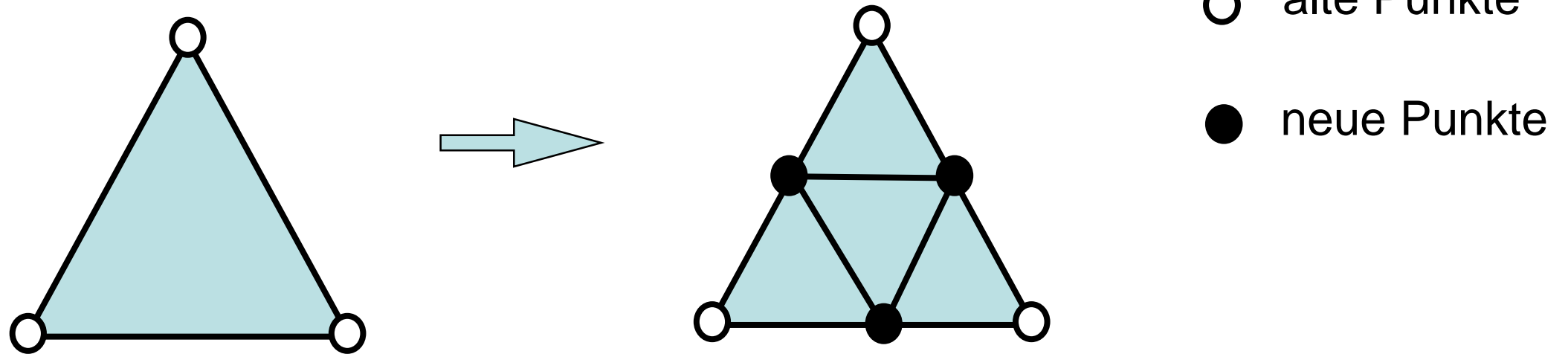
- Interaktive Manipulation aufwändig, wenn viele Dreiecke „von Hand“ manipuliert werden müssen.
- Idee:
  - Beginne mit einem einfachen Objekt, das gut manipulierbar ist.
  - Füge dann automatisiert nach und nach mehr Polygone ein, um ein feiner aufgelöstes Modell zu erreichen. (Subdivision Modelling).
  - Auf jeder Detailstufe kann weiter manipuliert werden.
  - Wurde bereits in den 70er Jahren entwickelt.
  - Wird heute vor allem bei Trickfilmen (Findet Nemo, ToyStory etc.) verwendet.

# Subdivision für Netze

- Wir beginnen zunächst mit Netzen, die nur aus Dreiecken bestehen.
- Füge zunächst einen neuen Punkte in die Mitte einer Kante ein und verbinde die neuen Punkte.
- Diese Methode wird edge-split genannt

# Subdivision für Dreiecksnetze

### ■ edge-split für Dreiecksnetze:



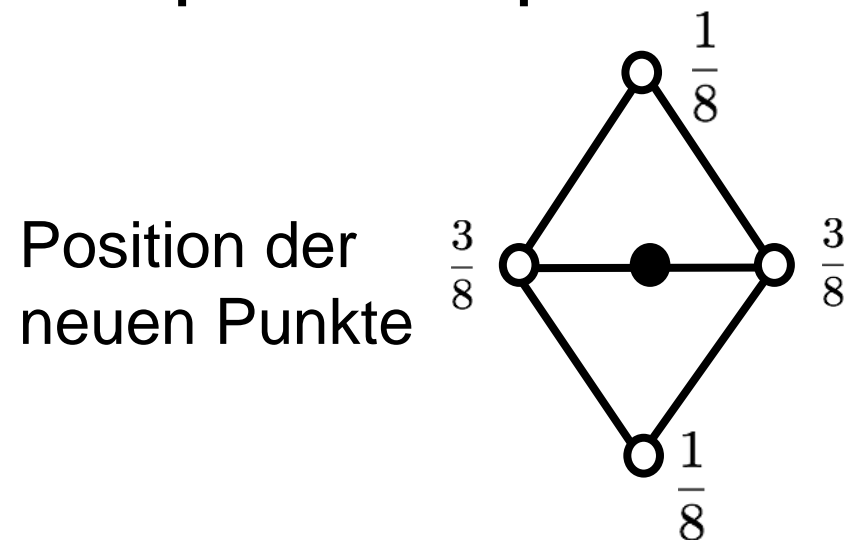
- Der edge-split zerteilt ein Dreieck in vier Dreiecke.
- Bisher liegen die vier Dreiecke noch in derselben Ebene.

# Average-Operation für Netze

- In dem bisherigen topologischen Unterteilungsschritt wurde aus einer Menge  $F^{k-1}$  von Faces und der Eckenmenge  $V^{k-1}$  durch teilen der Faces die Menge  $F^k$  erzeugt.
  - $k$  gibt also die Unterteilungsstufe an.
- Anschließend wird jetzt die Geometrie des Netzes durch die Berechnung neuer Koordinaten sowohl für die alten als auch für die neu erzeugten Punkte geändert.

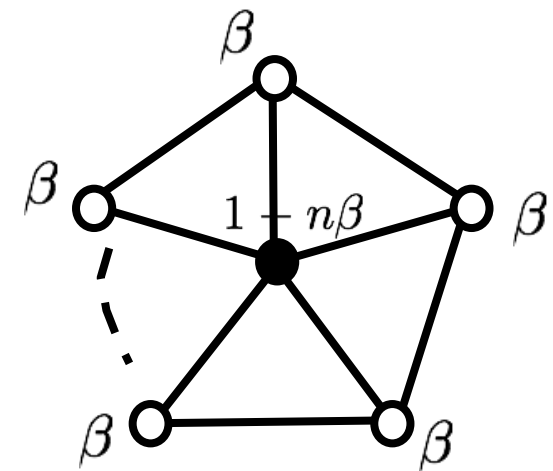
## Loop-Subdivision

- In der Praxis wird die Berechnungsvorschrift mit so genannten Stencils angegeben.
- Beispiel: Loop-Subdivision



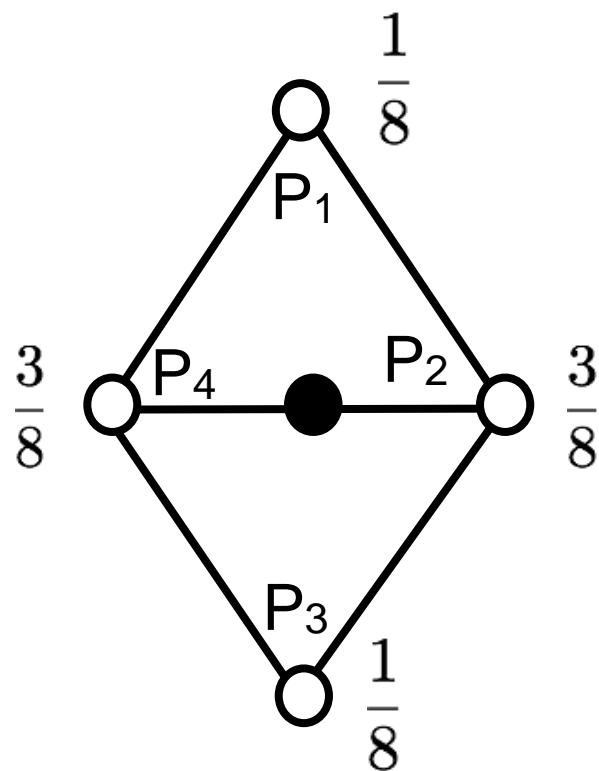
Neue Position  
der alten Punkte

$$\beta = \frac{1}{n} \left( \frac{5}{8} - \left( \frac{3}{8} + \frac{1}{4} \cos\left(\frac{2\pi}{n}\right) \right)^2 \right)$$



## Loop Subdivision

### ■ Erläuterung:



- Halbiere jede Kante durch Einfügen eines neuen Punktes.

- Berechne die Position als:

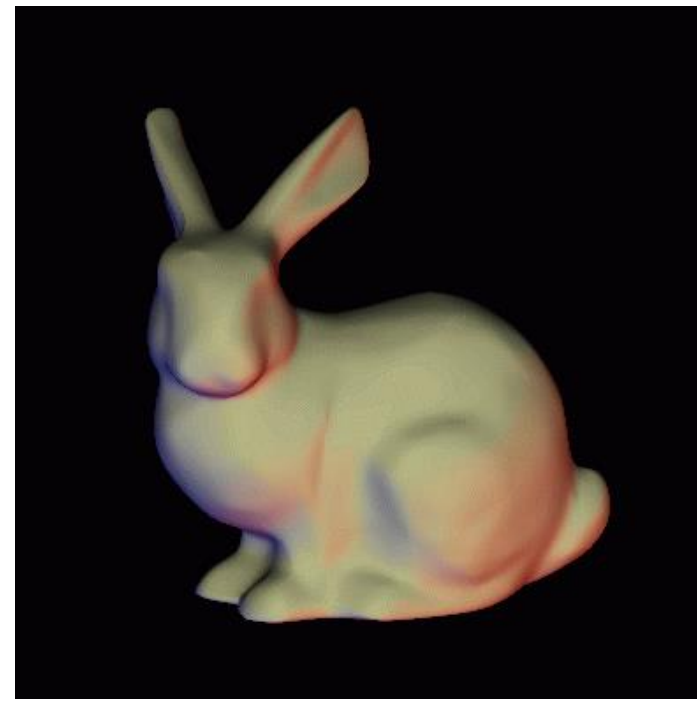
$$P = \frac{1}{8}P_1 + \frac{3}{8}P_2 + \frac{1}{8}P_3 + \frac{3}{8}P_4 +$$

- Wende dies für jede Kante auf dem Netz an.



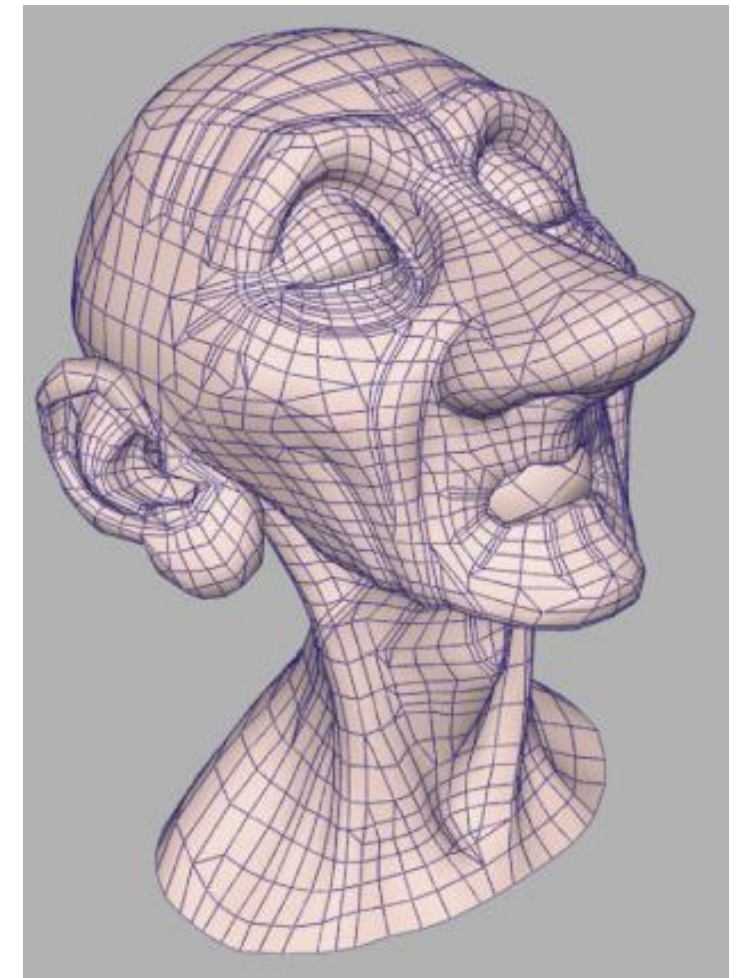
# Loop Subdivision

- Die Stencils enthalten also neben der Mittelungs-Vorschrift auch die Anleitung für die Topologische Bearbeitung des Netzes.



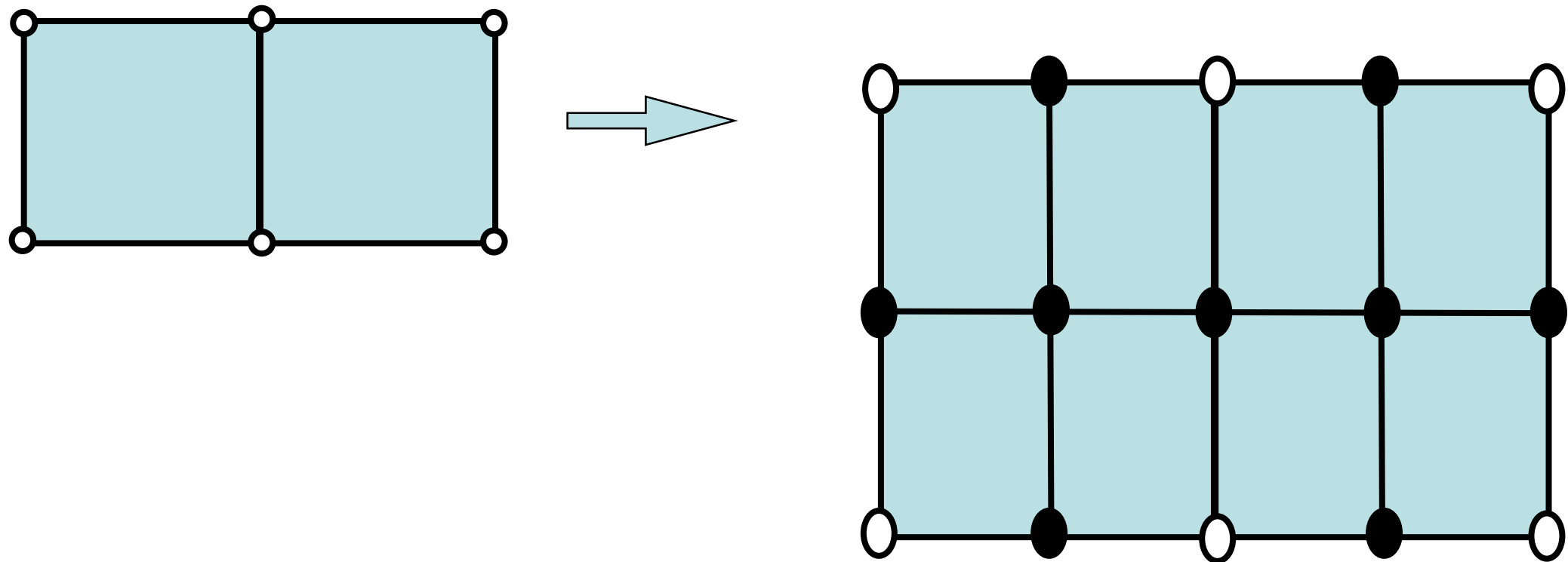
# Subdivision auf Vierecksnetzen

- Die Modelle in Pixar`s Trickfilmen sind größtenteils mit Vierecksnetzen modelliert.
- Der Catmull-Clark Algorithmus überführt zunächst jedes beliebige Polygone Netz in ein Vierecksnetz und führt dann eine Unterteilung durch.



# Unterteilung von Vierecksnetzen

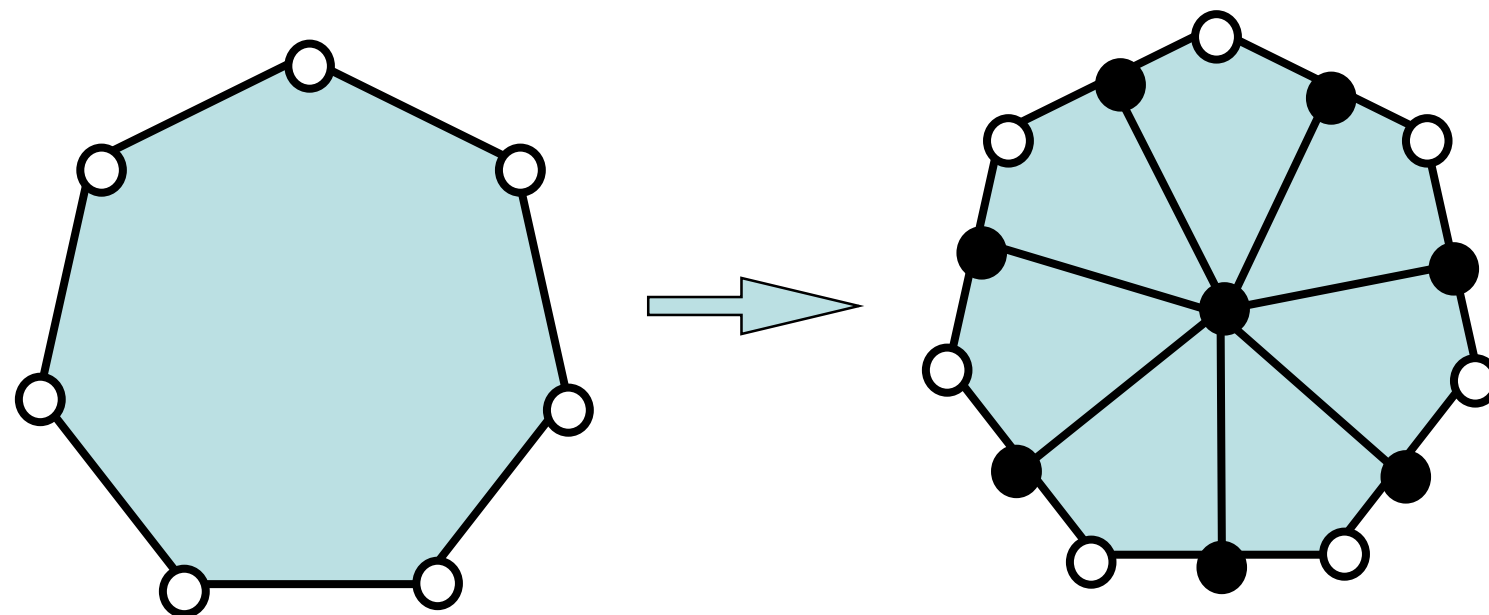
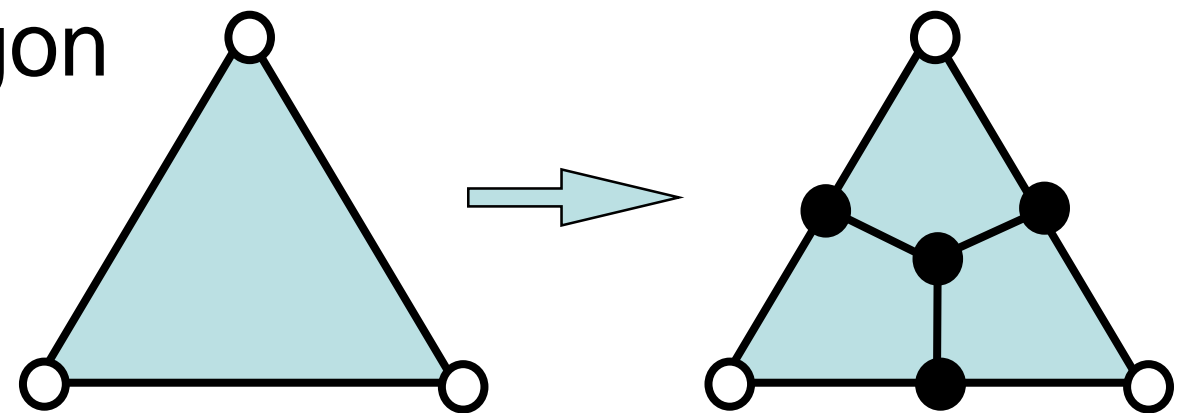
- Der Catmull-Clark Algorithmus führt einen kombinierten Edge- und Face-Split auf Vierecksnetzen durch.



- Vorschrift: Füge auf jede Kante und in die Mitte von jedem Face einen neuen Punkt ein und verbinde zu einem feinen Vierecksnetz.

# Unterteilung von Vierecksnetzen

- Idee: Nutze die gleiche Vorschrift um aus einem Netz mit beliebigen Faces ein Vierecksnetz zu machen.
- Damit wird aus jedem Polygon mit  $n$  Ecken ein Punkt mit  $n$  Nachbarn im feinen Netz.
- Das feine Netz besteht nur aus Vierecken.



## Catmull-Clark Unterteilung

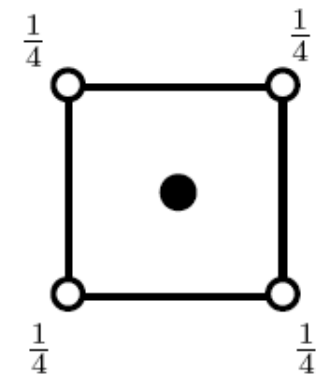
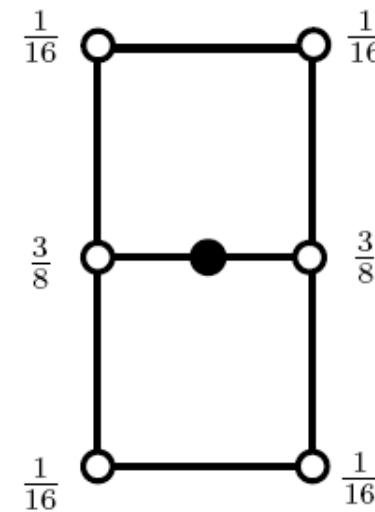
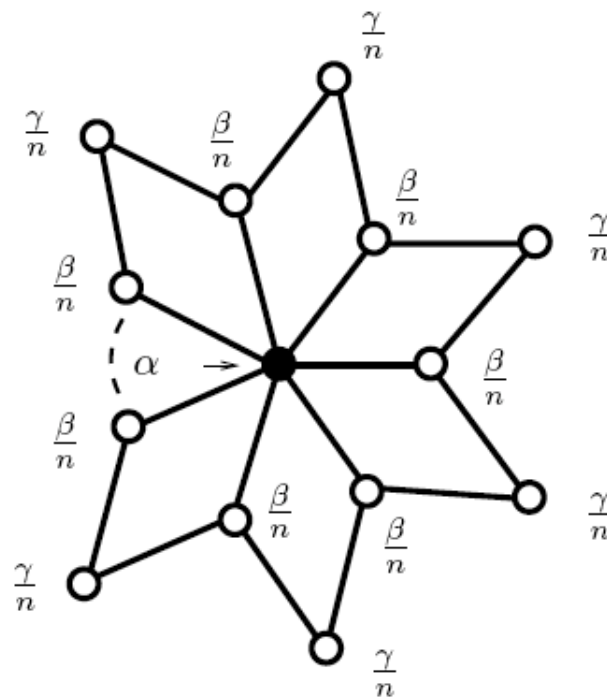
- Der Catmull-Clark Algorithmus wird durch folgende Stencils beschrieben:

$$\alpha = 1 - 7/(4n)$$

$$\beta = 3/(2n)$$

$$\gamma = 1/(4n)$$

$$\alpha + \beta + \gamma = 1$$



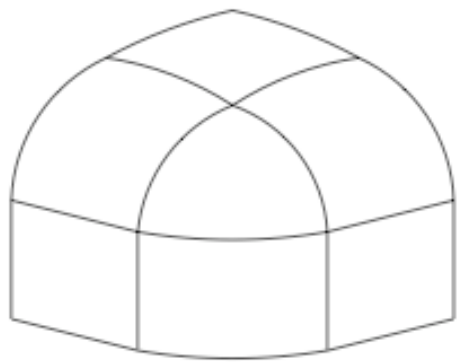
# Catmull Clark Unterteilung

- Catmull-Clark Flächen sind unter anderem deshalb beliebt, weil sie in Regionen, in denen nur Punkte mit vier Nachbarn auftreten, sehr schöne Flächen erzeugen.
- Diese Flächen haben einen Zustand minimaler Biegeenergie ähnlich einer dünnen Metallplatte (thin plate).
- Beobachtung: Die Bereiche mit Punkten mit 4 Nachbarn werden mit fortschreitender Unterteilung immer größer, Punkte mit Nachbarschaften  $\neq 4$  werden isoliert.
- Software, die Catmull-Clark Unterteilung verwendet:

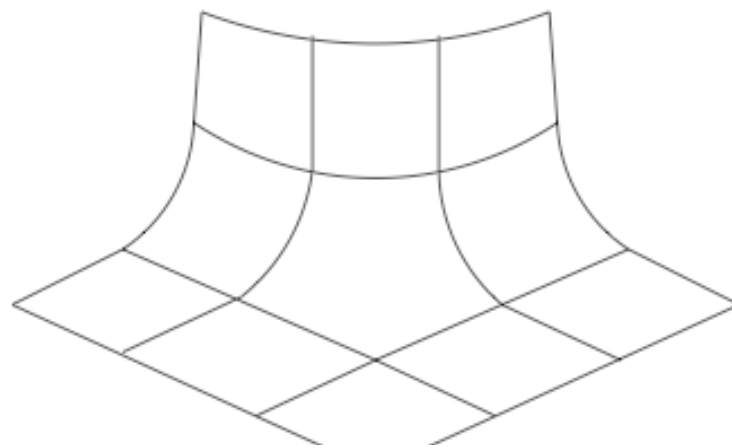
[3ds max](#), 3D-Coat, [AC3D](#), Anim8or, [Blender](#), [Carrara](#), Cheetah3D, [Cinema4D](#), DAZ [Studio](#), 2.0, [Gelato](#), [Hexagon](#), JPatch, [K-3D](#), LightWave [3D, version 9](#), Maya, modo, Mudbox, [Silo](#), [SketchUp](#) -Requires [a Plugin.](#), Softimage XSI, [Strata 3D CX](#), [Wings 3D](#), Zbrush

# Unterteilung von Vierecksnetzen

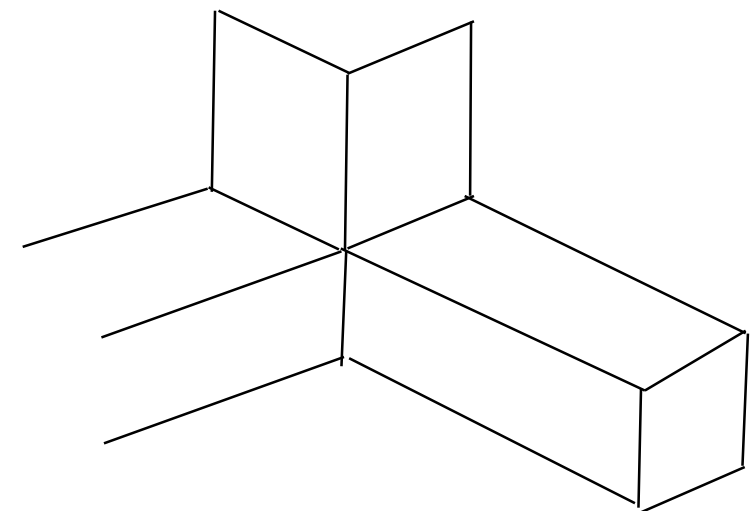
- Vierecksnetze sind beliebt zum Modellieren sind, weil viele Dinge des täglichen Lebens  $90^\circ$  Winkel beinhalten.
- Typische Situationen:



Kofferecke:  
3-er Nachbarschaft



Hausecke:  
5-er Nachbarschaft



Affensattel:  
6-er Nachbarschaft



# Zusammenfassung Subdivision

### ■ Vorteile:

- Die Regeln zur Verfeinerung sind geometrisch eher einfach.
- Verfahren können adaptiv nach Bedarf verfeinern.
- Die Qualität der Flächen kann durch mathematische Analyse vorausgesagt werden.

### ■ Nachteile:

- Die Unterteilung kann nicht rückgängig gemacht werden, d.h. eine Vereinfachung ist mit den bisherigen Techniken nicht möglich.
- Damit müssen für Level-of-Detail-Verfahren (z.B. für Flugsimulator) alle Stufen der Unterteilung gespeichert werden (aufwendig).