Vergleich verschiedener Datenstrukturen zur Verwaltung von Polygonnetzen

Sven Emme

6. Dezember 2011

Polygonnetze

- Nachbarschaftsbeziehungen

Datenstrukturen für Polygonnetze

- Shared Vertex
- Kantenbasierte Datenstrukturen
  - Winged Edge
  - Half Edge

Directed Edges

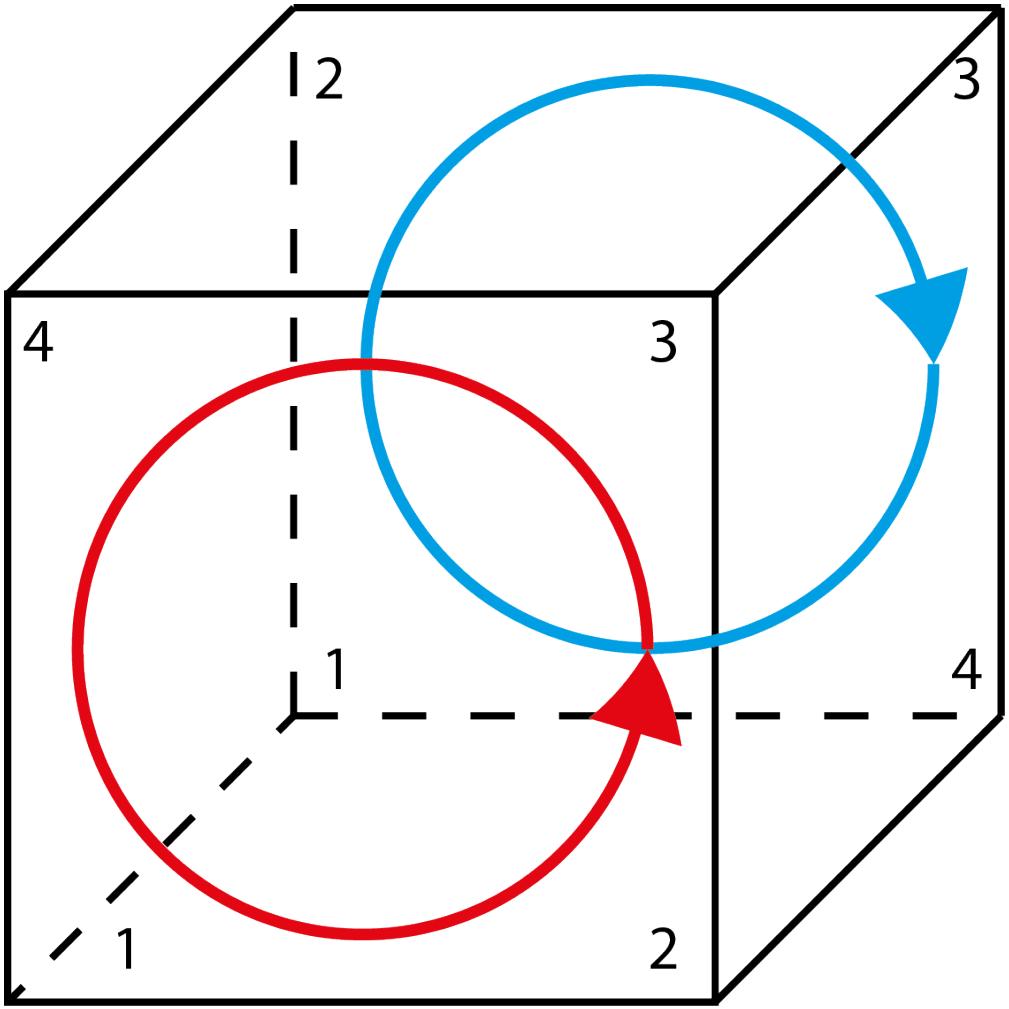
- Aufbau
- Speicherplatz sparende Varianten

Vergleich der Datenstrukturen

Polygonnetze

- Ein Polygonnetz M ist gegeben durch:
- Eine Menge von Kanten E
- Eine Menge von Ecken V
- Und eine Menge von Flächen F
- Wobei eine Kante immer zwei Ecken miteinander verbindet
- Und eine Fläche immer von einer Menge von Kanten beschrieben wird

Orientierbarkeit

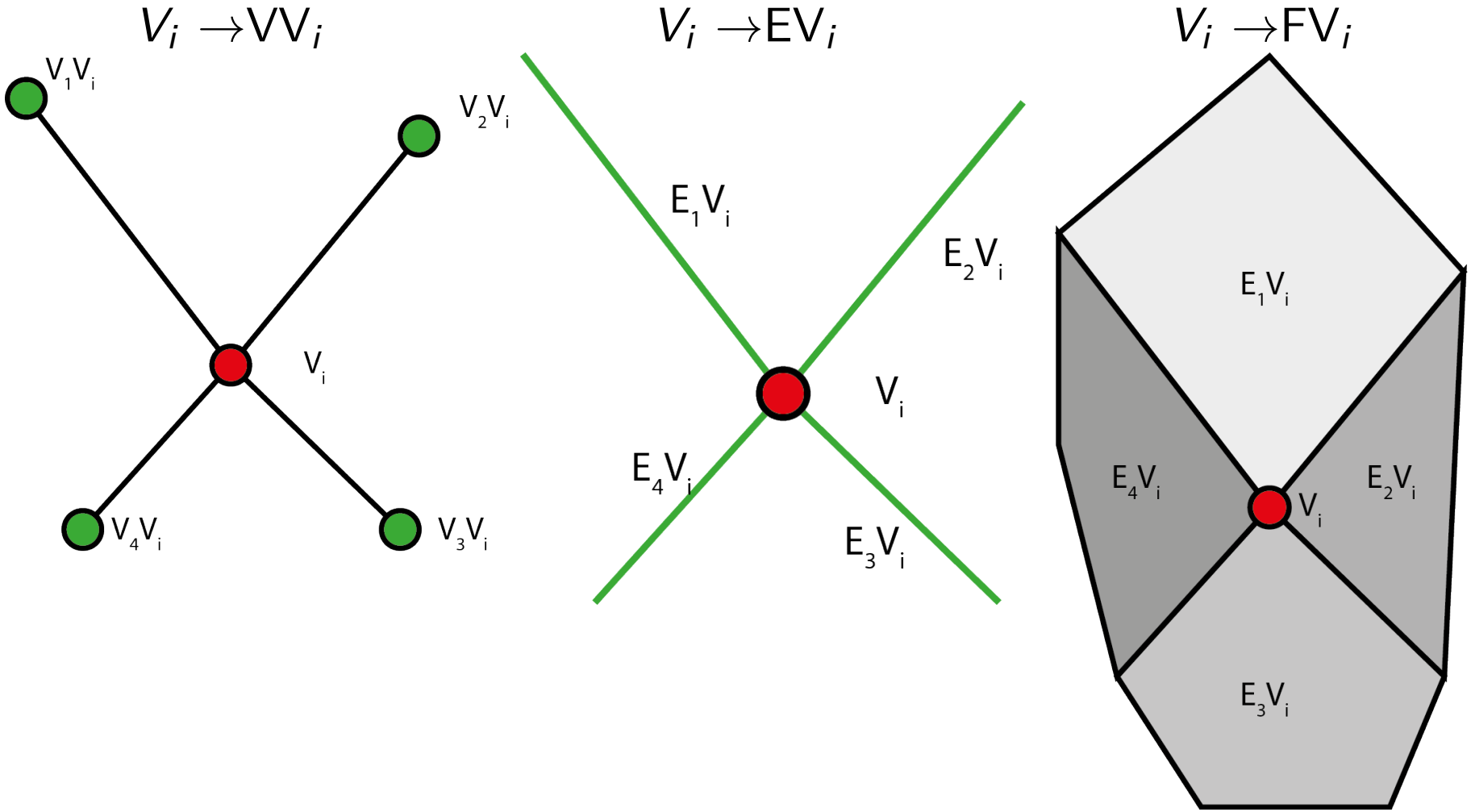


- Um zu bestimmen ob die Vorder- oder die Rückseite einer Fläche betrachtet wird, werden die Ecken orientiert angegeben
- Bei Open GL ist es z.B. immer gegen den Uhrzeigersinn, wenn der Betrachter auf die Vorderseite schaut.

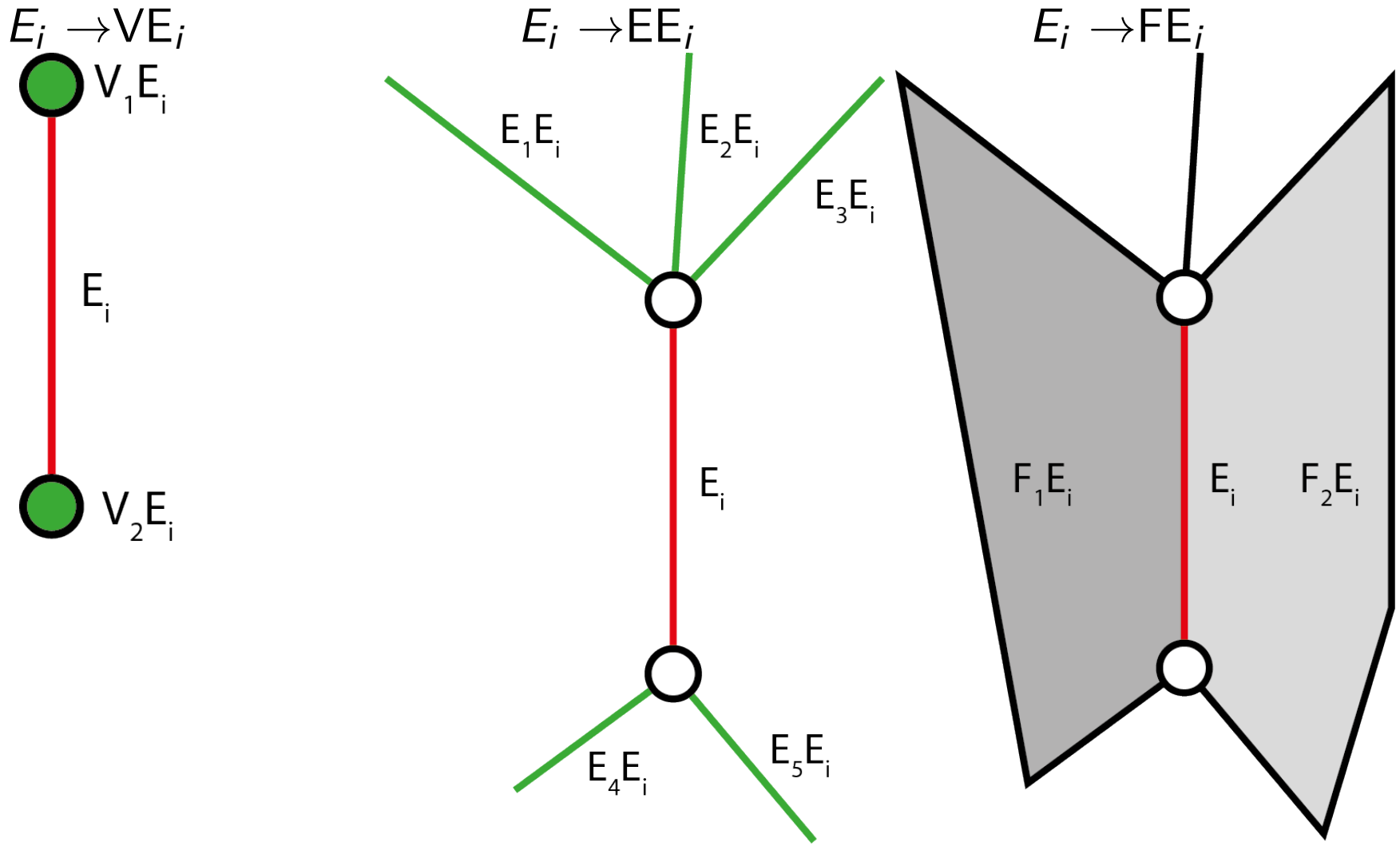
Nachbarschaftsbeziehungen von Polygonnetzen

- Tony Woo hat 1985 Datenstrukturen für Polygonnetze auf ihre Laufzeit und ihren Speicherbedarf analysiert [Woo85]
- Nach Woo gibt es 9 Nachbarschaftsbeziehungen bei Polygonnetzen
- Anfragen auf Nachbarschaft werden immer mit einer bestimmten Kante, Ecke oder Fläche gestellt.
- Die Ausgabe besteht dann aus allen Kanten, Ecken oder Flächen, die mit der übergebenen Instanz benachbart sind.

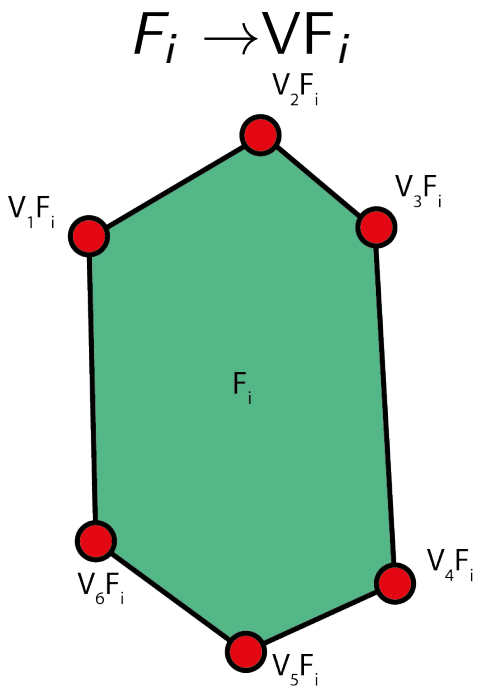
Nachbarschaftsbeziehungen von Ecken

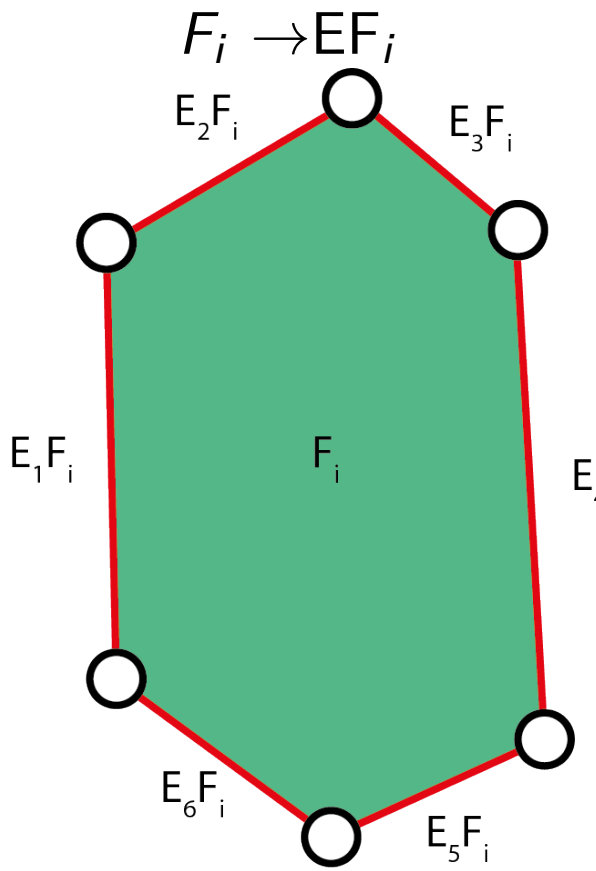


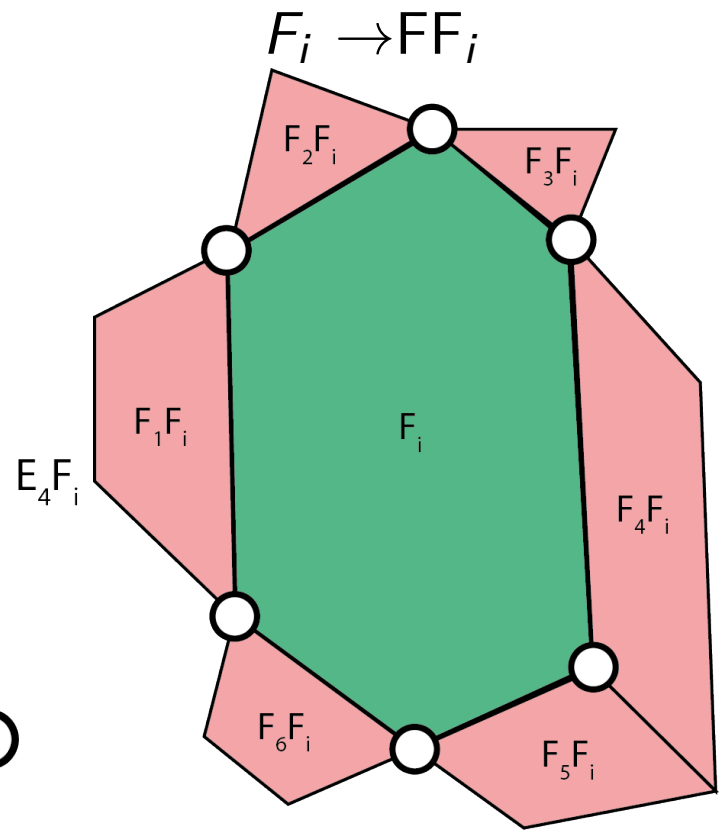
Nachbarschaftsbeziehungen von Kanten



Nachbarschaftsbeziehungen von Flächen



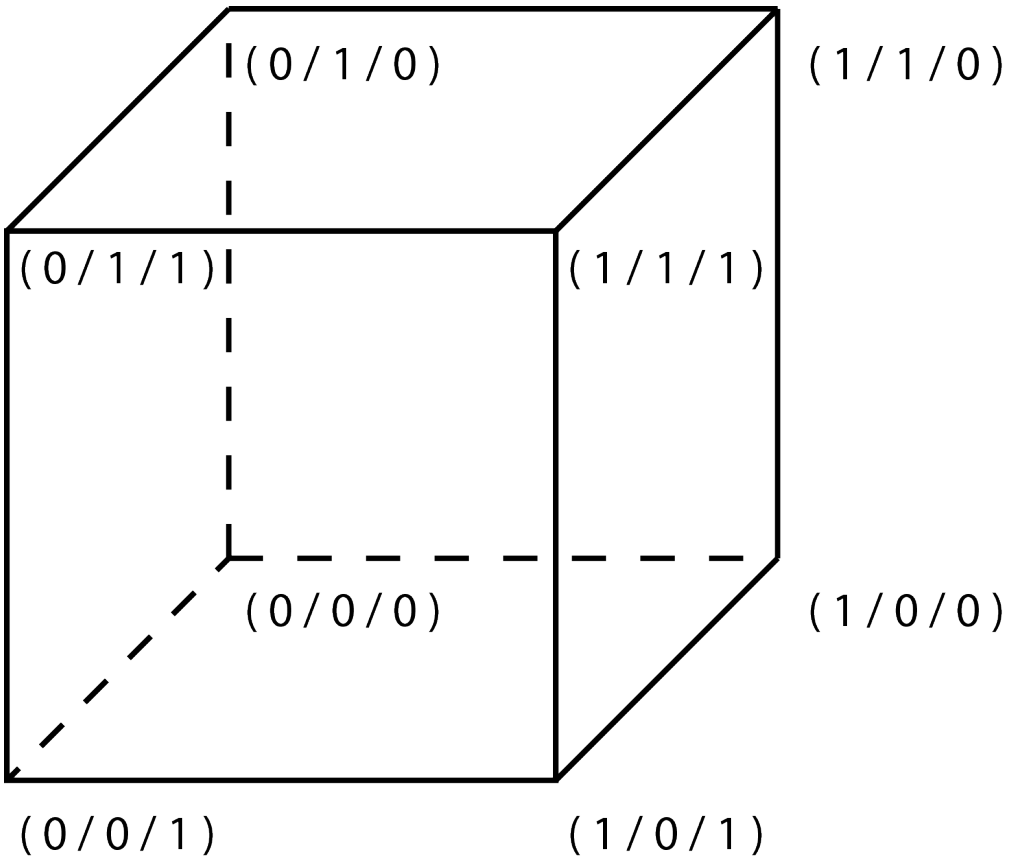




Nachbarschaftsbeziehungen von Polygonnetzen

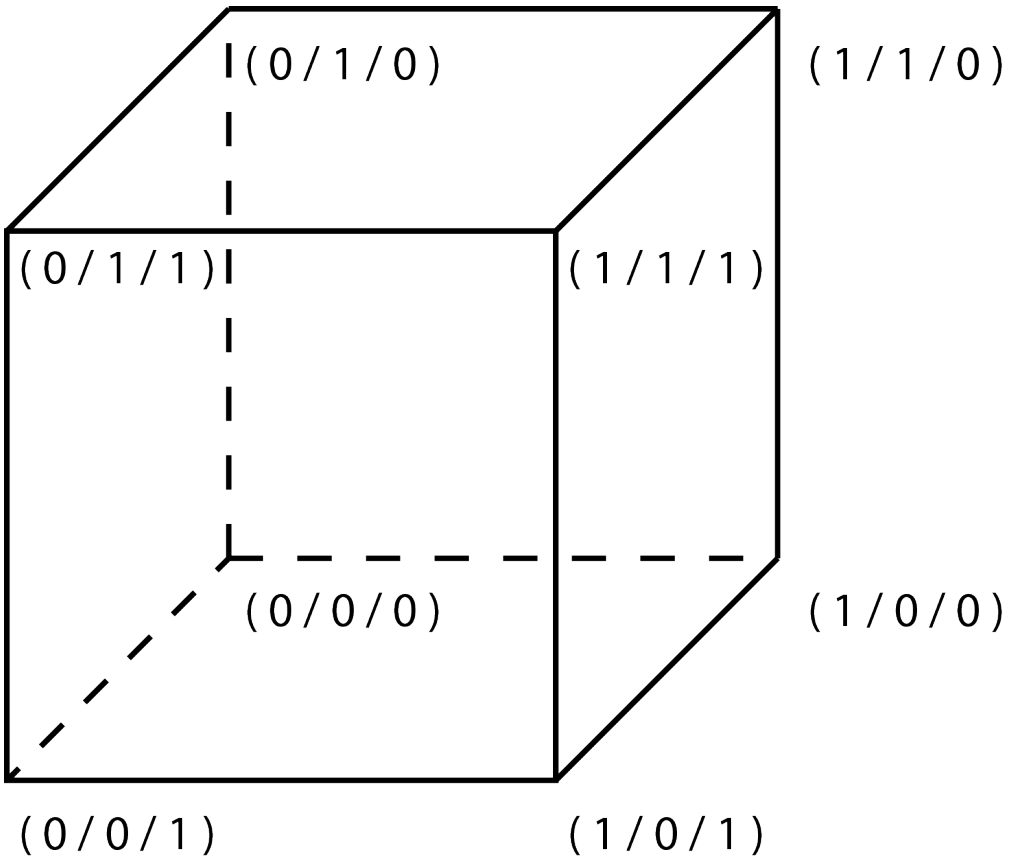
- Relationen die in der Datenstruktur gespeichert werden, können kombiniert werden
- Wenn die Relation allerdings nicht in der Datenstruktur gespeichert ist und es keine Kombination von gespeicherten Relationen gibt muss die Relation gesucht werden
- Diese Operation hat eine Laufzeit von $\mathcal{O}(n)$. Wobei n die Anzahl der gespeicherten Elemente der angeforderten Instanz ist.
- Wenn man beachtet, dass es häufig nötig ist für alle Elemente diese Nachbarschaftsbeziehung zu ermitteln wird daraus schnell ein $\mathcal{O}(n^2)$

Shared Vertex bei Polygonnetzen



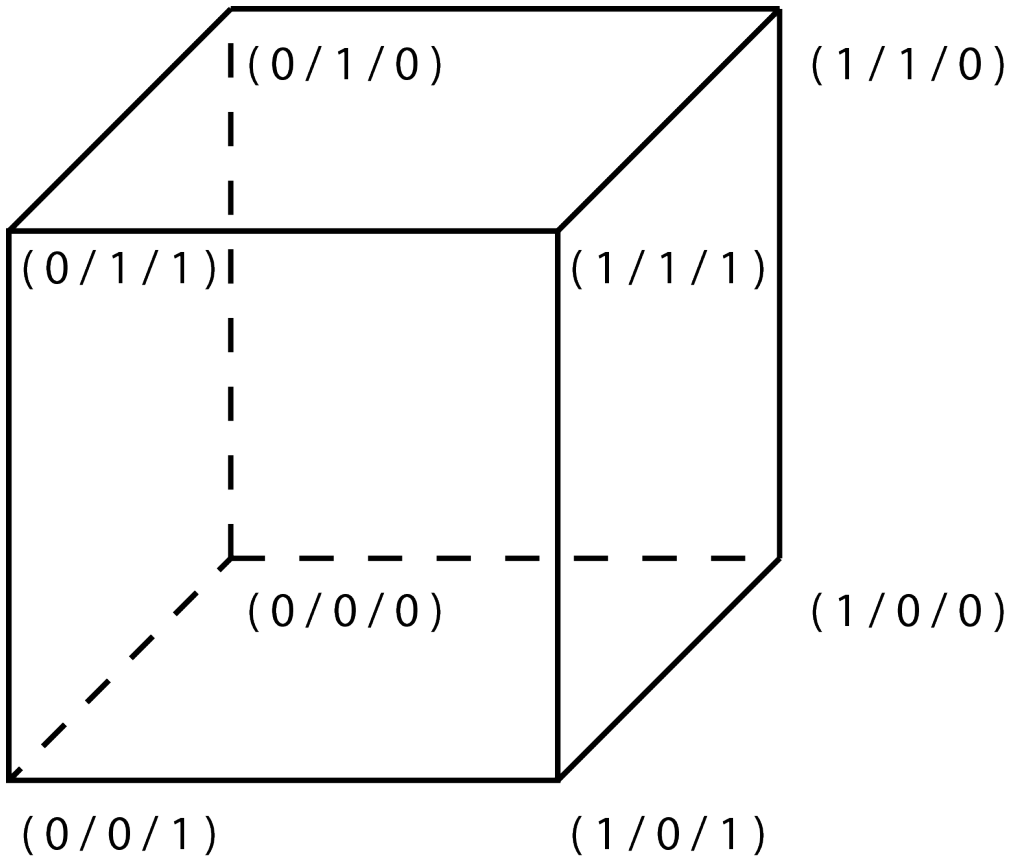
- Einfacher Ansatz: Speichere für jede Fläche ihre Ecken
- Dabei werden viele Koordinaten mehrfach gespeichert

Shared Vertex bei Polygonnetzen



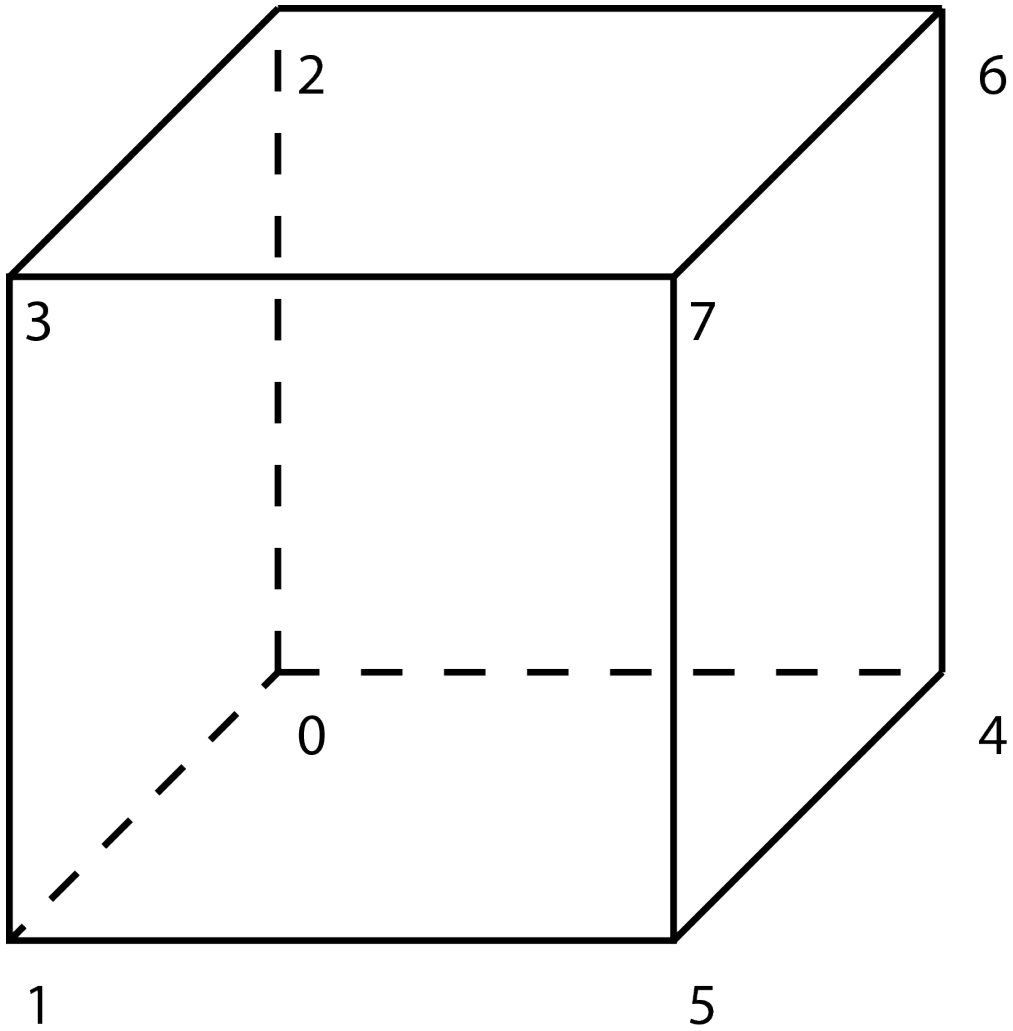
Fläche	Koordinaten
1	0 / 1 / 0
	0 / 1 / 1
	1 / 1 / 1
	1 / 1 / 0
2	0 / 1 / 0
	1 / 1 / 0
	1 / 0 / 0
	0 / 0 / 0

Shared Vertex bei Polygonnetzen



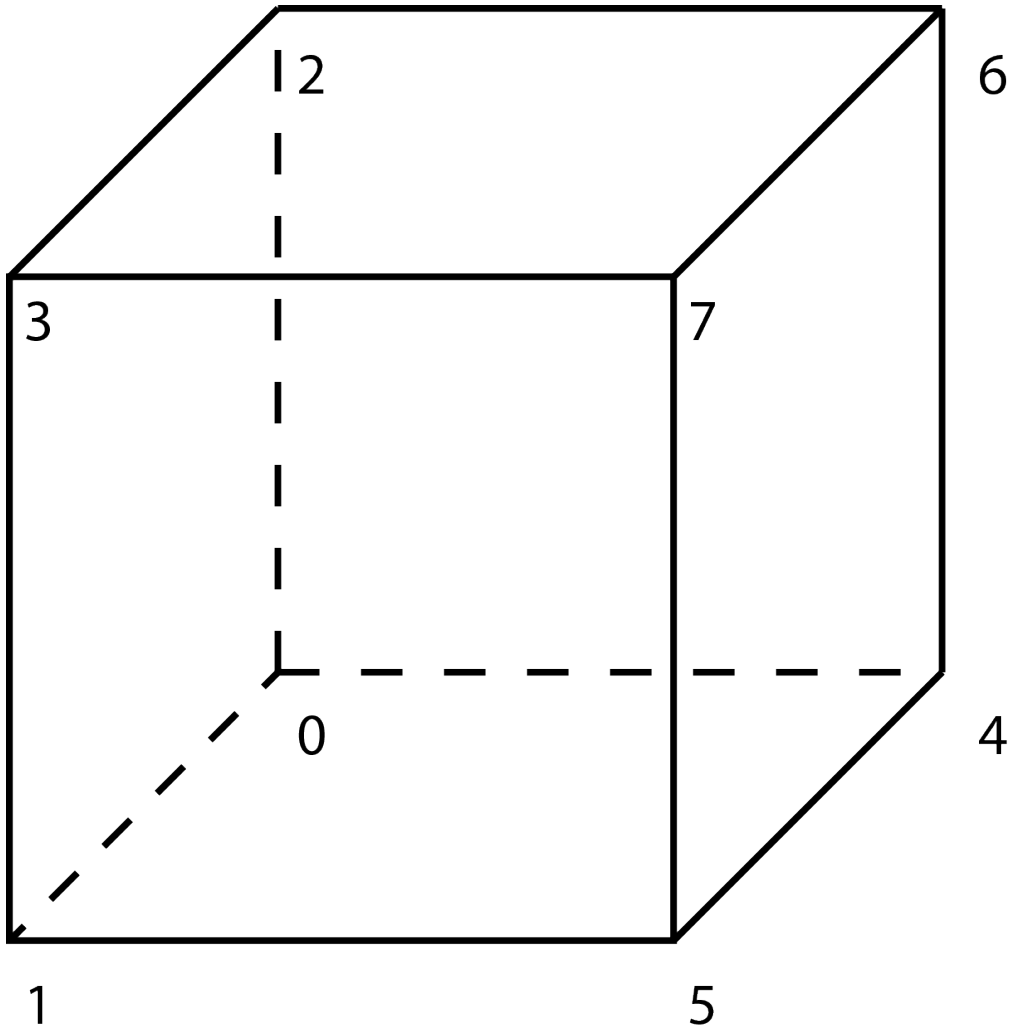
- Durch die redundante Speicherung der Daten wird viel Speicherplatz verschwendet
- Erhöhte Rechenzeit:
- Änderungen an den Koordinaten müssen für jede Fläche aktualisiert werden
- Beim Suchen müssen immer 3 Gleitkommazahlen verglichen werden

Shared Vertex bei Polygonnetzen



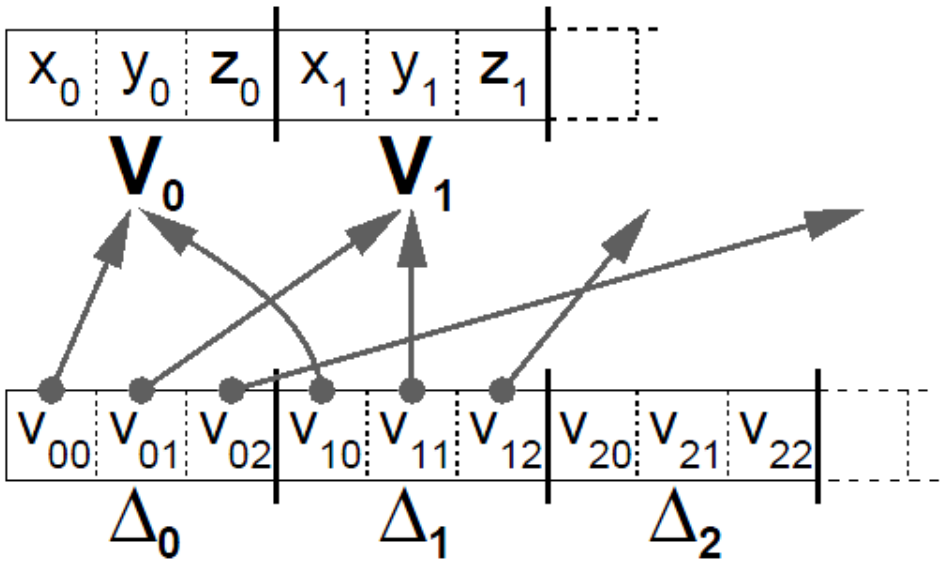
- Lösung: Speichere die Koordinaten der Ecken nur einmal
- Gib die Fläche dann als 3 Zeiger auf die Koordinaten der Ecken an

Shared Vertex bei Polygonnetzen



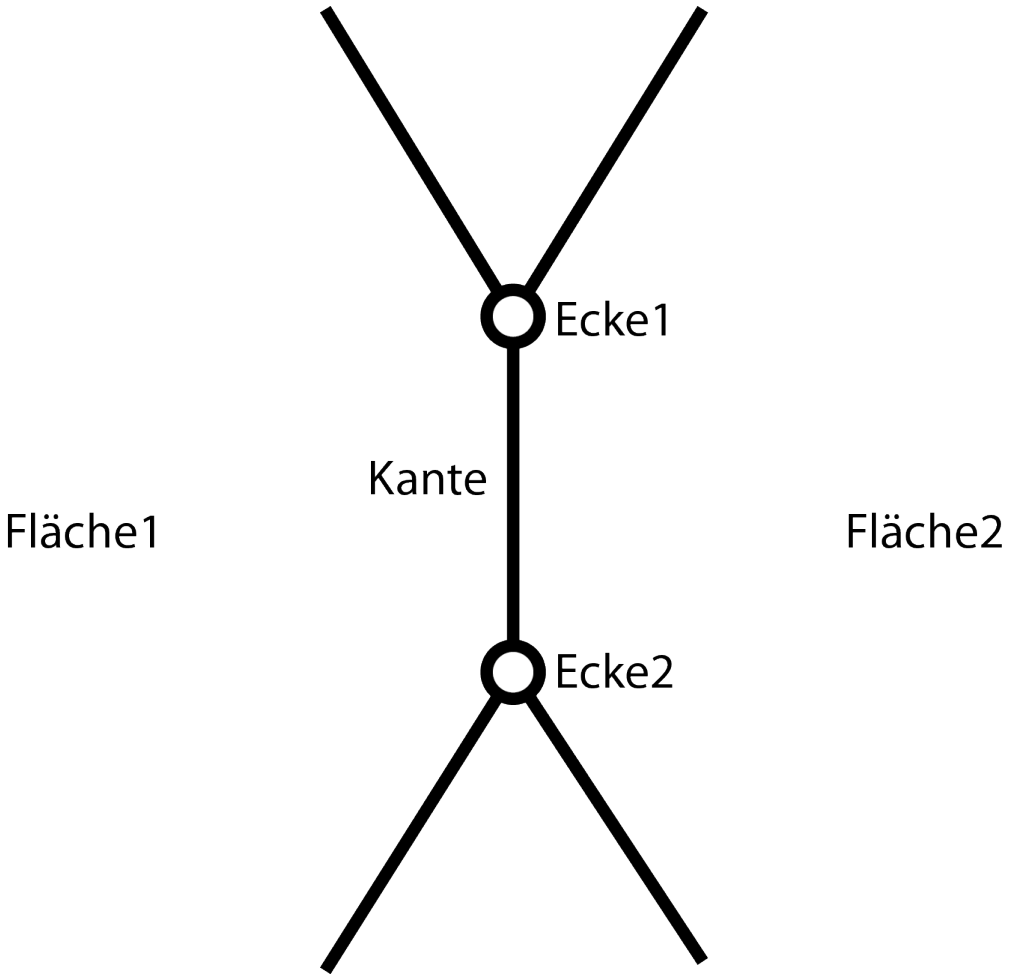
Fläche	Eckenliste
1	{ 2 ; 6 ; 4 ; 0 }
2	{ 2 ; 3 ; 7 ; 6 }
3	{ 3 ; 1 ; 5 ; 7 }
4	{ 7 ; 5 ; 4 ; 6 }

Shared Vertex bei Dreiecksnetzen



- Jedes Dreieck besitzt 3 Zeiger
- Jeweils 1 Zeiger auf einen Eckpunkt des Dreiecks
- Dabei werden die Eckpunkte gegen den Uhrzeigersinn angeordnet

Kantenbasierte Datenstrukturen



- Relationen zwischen Kanten stehen im Vordergrund
- Bei allen diesen Strukturen referenzieren Ecken und Flächen jeweils ein Kantenelement, mit dem sie verbunden sind

Winged Edge Datenstruktur

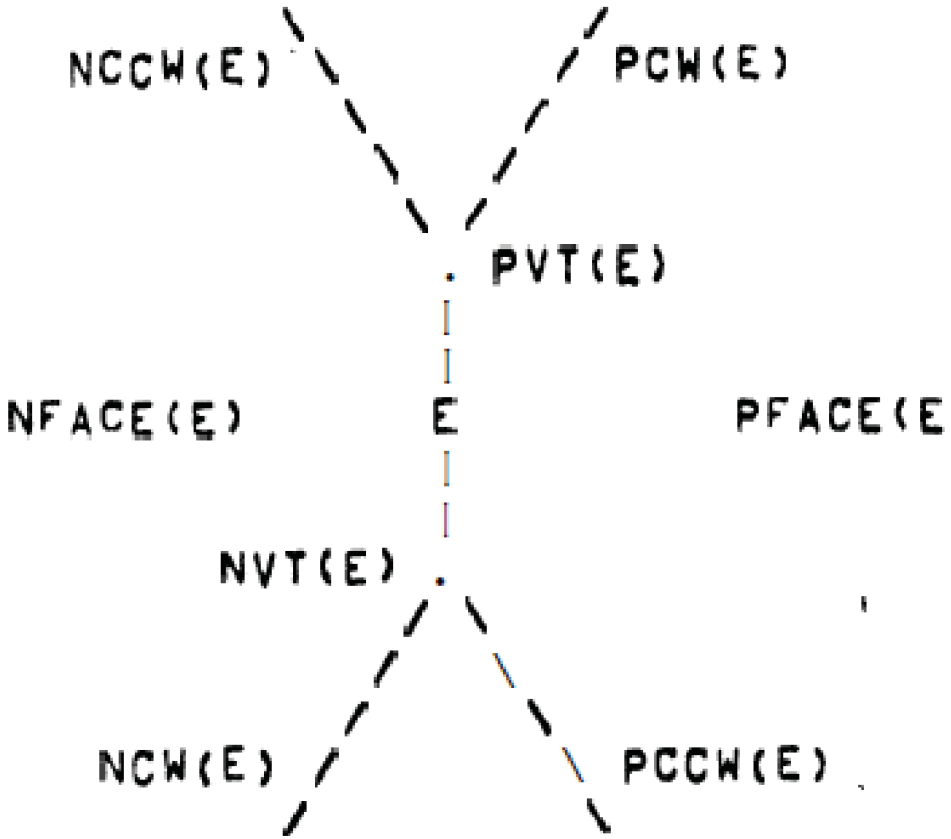
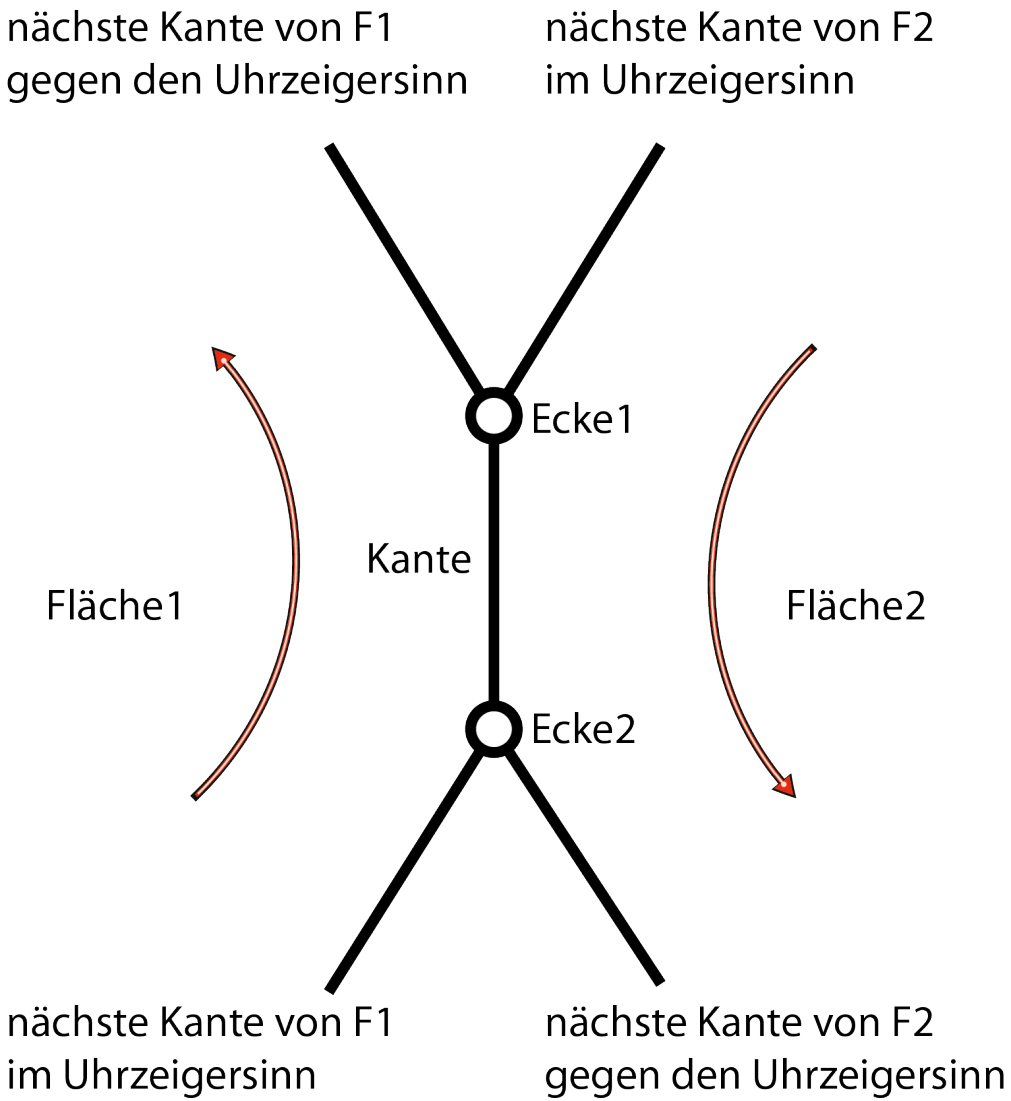


Bild aus [Bau72]

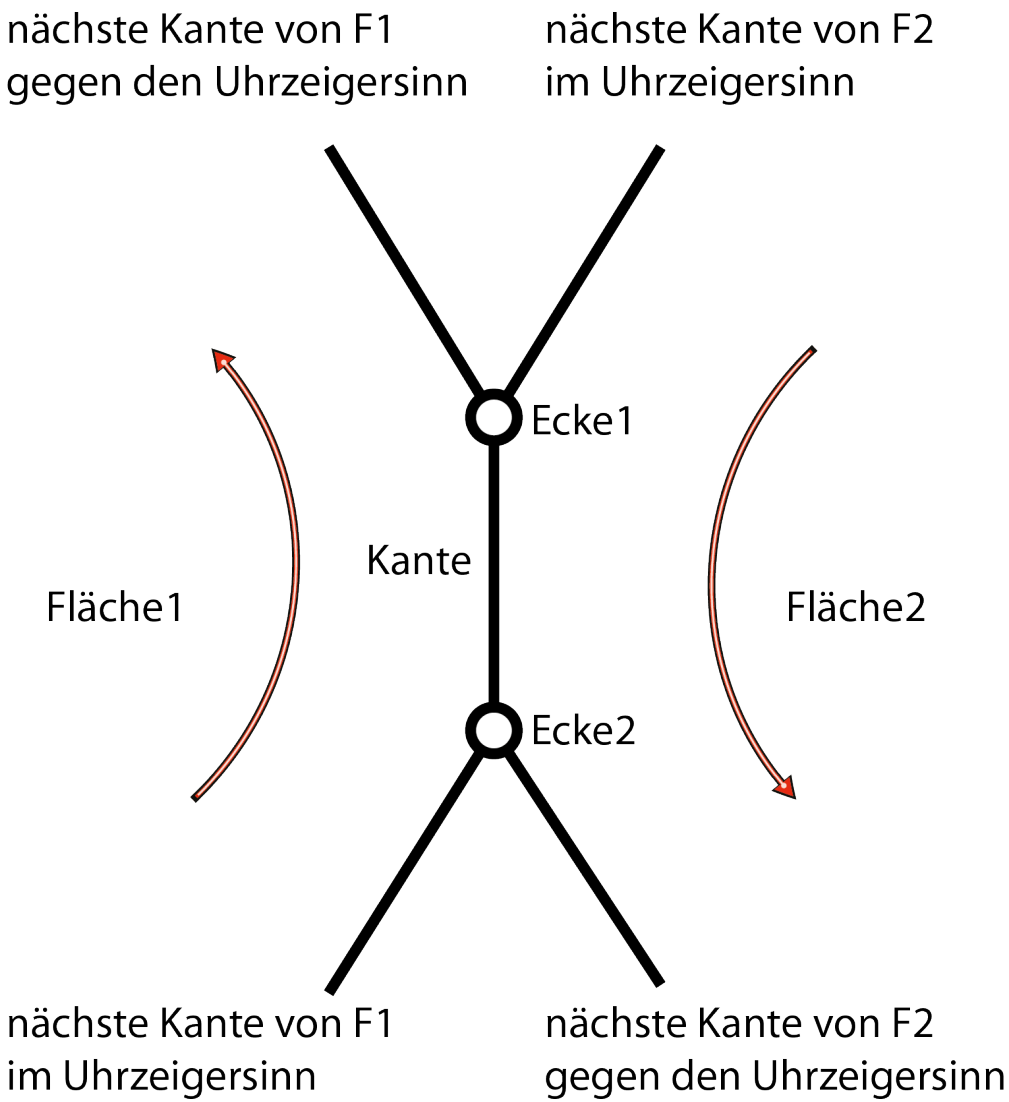
- 1972 von Baumgart entwickelt
- Jeder Ecke ist eine Kante zugeordnet mit der sie verbunden ist
- Jeder Fläche ist eine Kante zugeordnet die sie umläuft

Winged Edge Datenstruktur



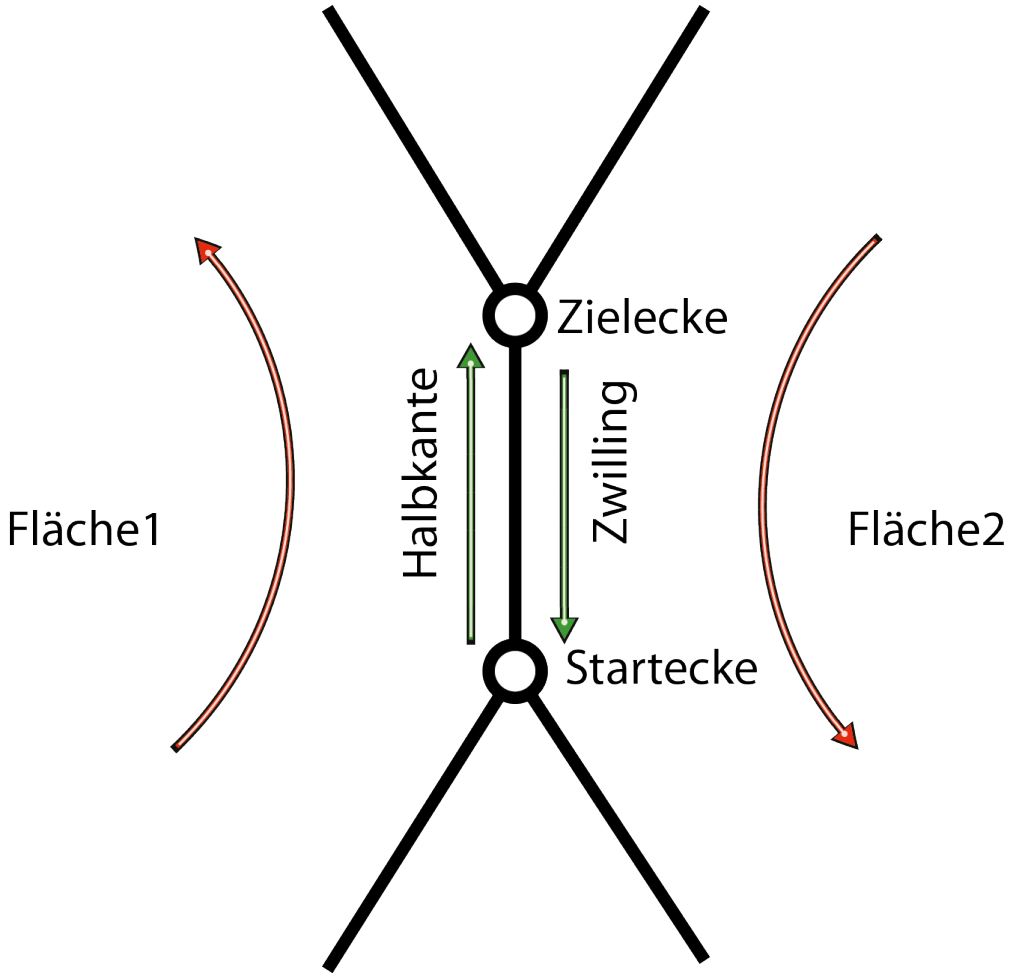
- Zu jeder Kante gehören 8 Zeiger:
- 2 Ecken, die Start- und Endpunkt bestimmen
- 2 Flächen, die sich an der Kante schneiden
- zu jeder Fläche jeweils 2 Nachfolgerkanten
- eine im und eine gegen den Uhrzeigersinn
- 60 byte Speicherplatz pro Dreieck

Winged Edge Datenstruktur



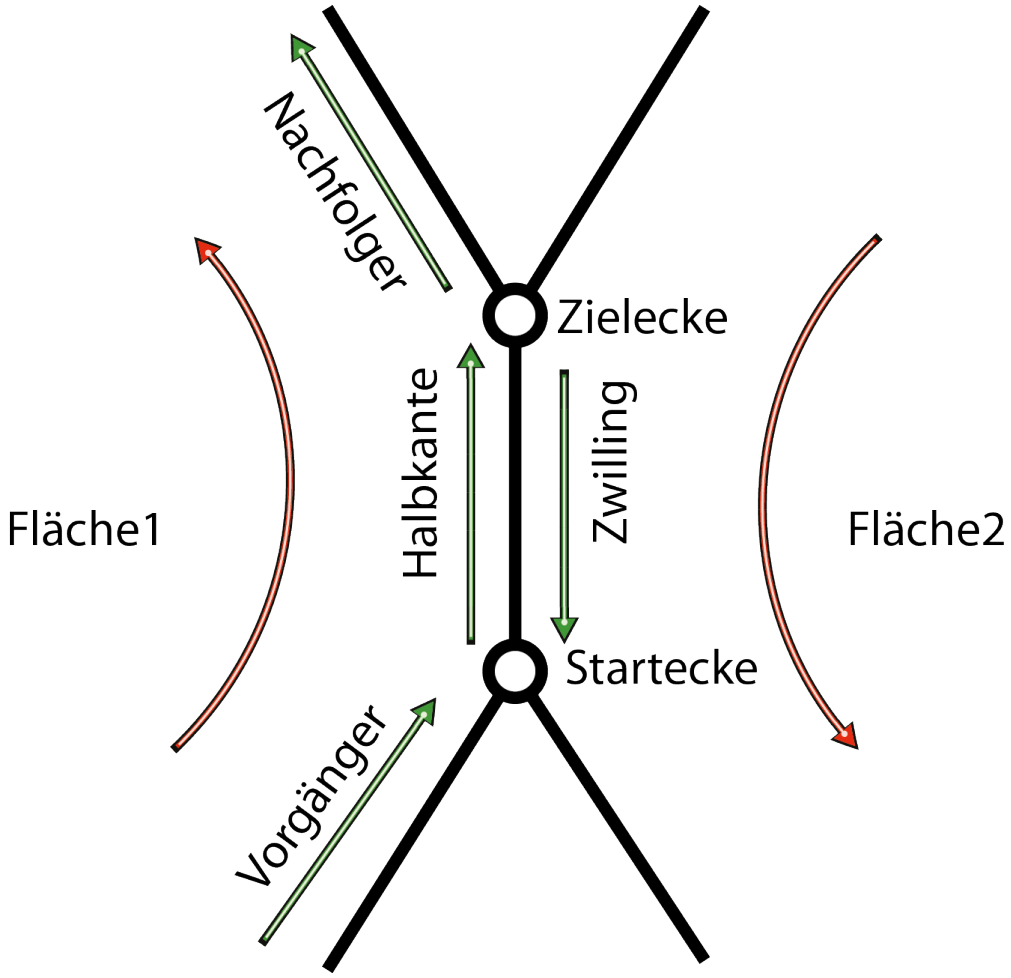
- Die Winged Edge lässt sich auf 6 Zeiger pro Kante reduzieren
- Durch weglassen der Referenzen auf die Nachfolger im Uhrzeigersinn
- 48 byte Speicherplatz pro Dreieck

Half Edge Datenstruktur



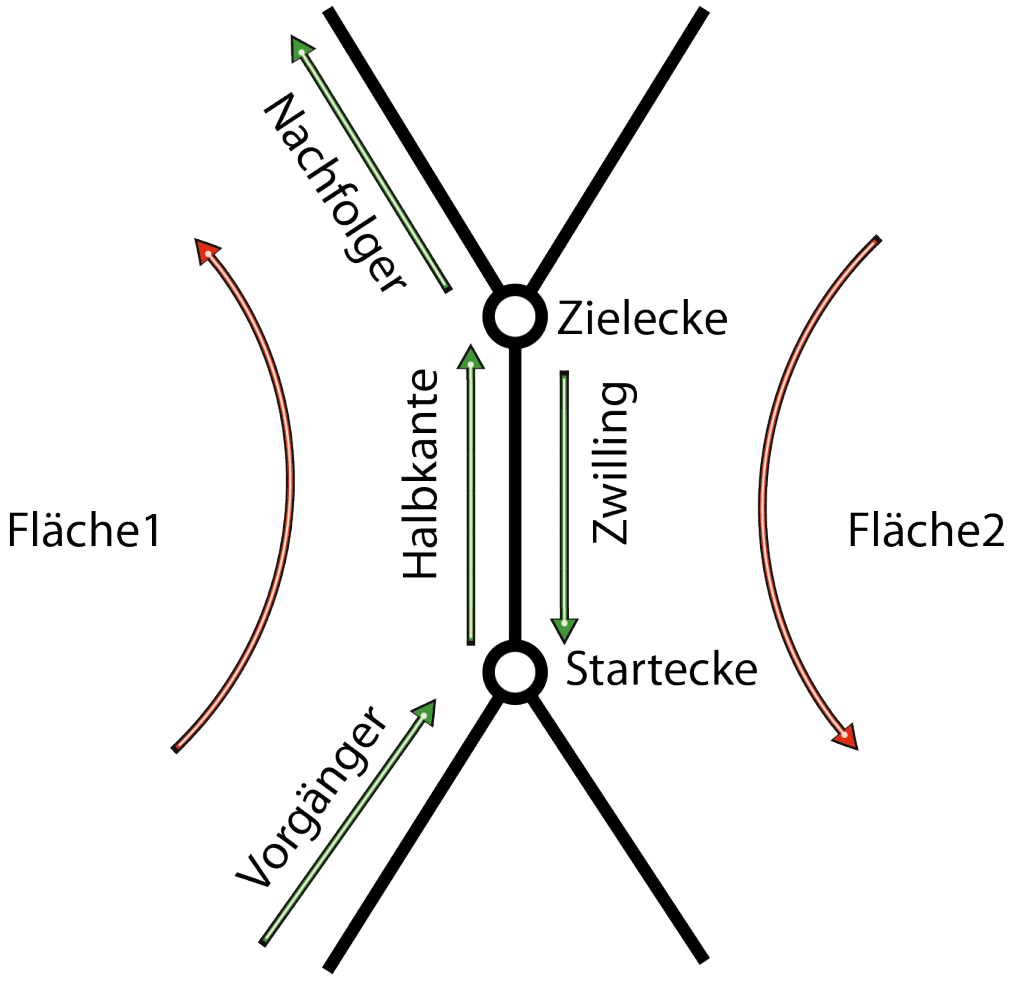
- Kanten werden durch jeweils zwei Halbkanten ersetzt
- jede Halbkante ist eine gerichtete Kante die von einem Startpunkt zu einem Endpunkt zeigt
- Die Halbkante die vom Endpunkt zum Startpunkt läuft wird als Zwilling bezeichnet

Half Edge Datenstruktur



- Jede Halbkante besitzt 6 Zeiger
- 2 Zeiger auf Start und Endpunkt
- 1 Zeiger auf den Zwilling
- Jeweils 1 Zeiger auf die Nachfolger Halbkante und die Vorgängerhalbkante
- 1 Zeiger auf die Fläche der Half Edge

Half Edge Datenstruktur

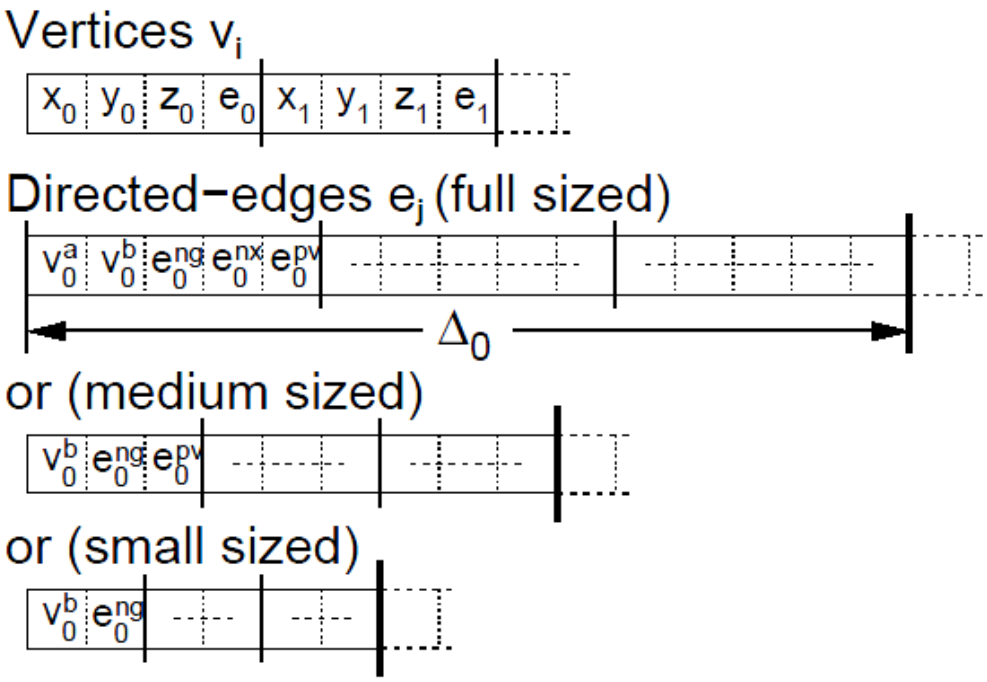


- Die Half Edge lässt sich auf 4 Zeiger Pro Half Edge reduzieren
- Durch weglassen der Referenzen auf die Startecke und den Vorgänger
- 60 byte Speicherplatz pro Dreieck

Directed Edges

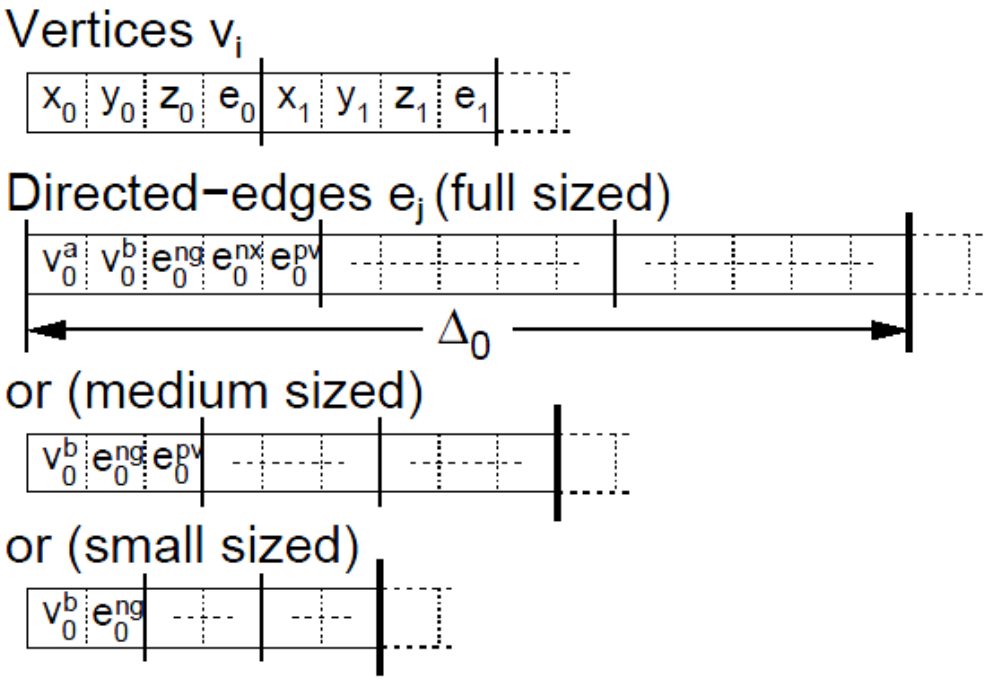
- Wird in dem Artikel *A Scalable Representation for Triangle Meshes* von Swen Campagna et al.
- Basiert auf der Half Edge Datenstruktur
- Ist auf Dreiecke beschränkt
- Ist in 3 Varianten verfügbar, die jeweils den Speicherverbrauch reduzieren
- Bietet eine gute Unterstützung für nicht-mannigfaltige Dreiecksnetze
- Arbeitet mit Array-indices statt mit Zeigern

Aufbau



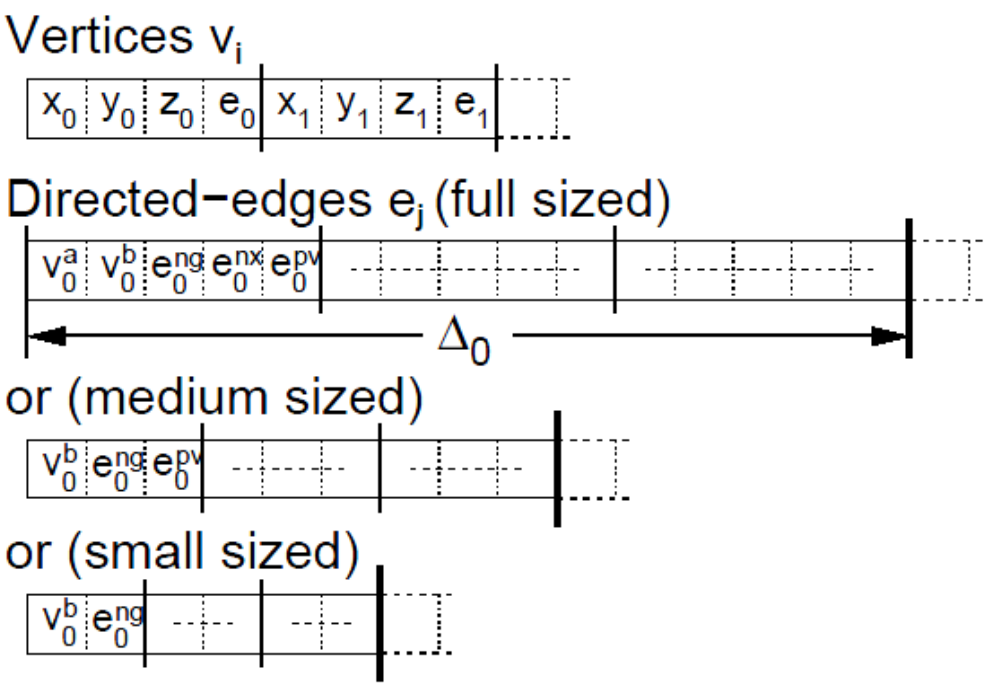
- Nachbarschafts-beziehungen wie bei der Half Edge
- jede Kante referenziert Start- und Endpunkt, Vorgänger, Nachfolger und eine Nachbarkante
- Durch den Array Index lässt sich die Referenz der Fläche bestimmen
- 68 byte Speicherplatz pro Dreieck in der full size Ausführung

medium size



- Die Referenzen auf den Startpunkt und den Nachfolger können weggelassen werden
- Der Startpunkt lässt sich durch den Endpunkt der Nachbarkante beschreiben
- Der Nachfolger lässt sich durch zweimaliges abwandern des Vorgängers beschrieben
- 44 byte Speicherplatz pro Dreieck in der medium size Ausführung

small size



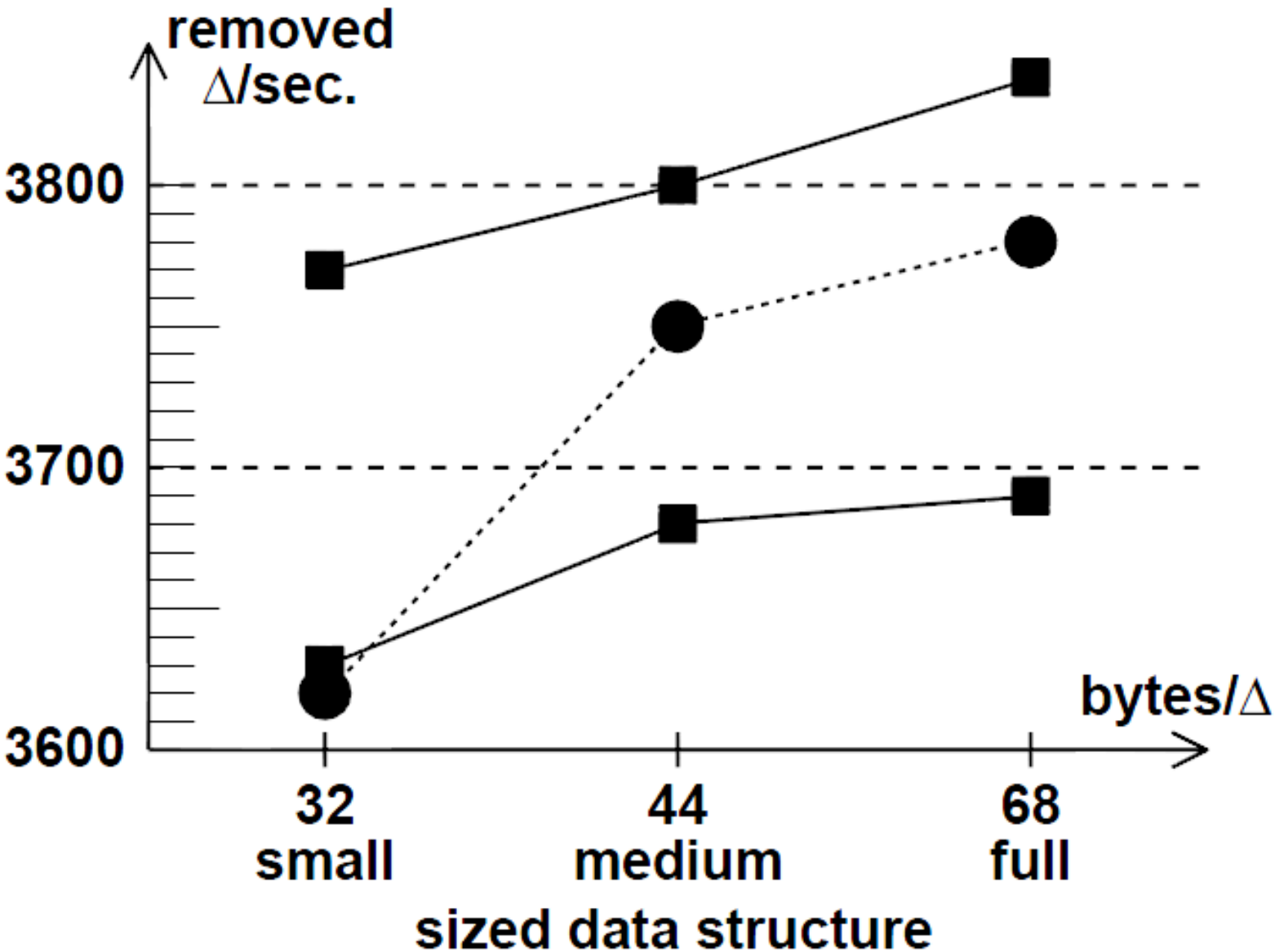
- Die Position des Vorgängers sowie des Nachfolgers lässt sich berechnen
- Dadurch kann die Vorgänger Referenz auch wegfallen
- 32 byte Speicherplatz pro Dreieck in der medium size Ausführung

Vergleich

Daten Struktur	$^{\text{bytes}}/_{\Delta}$	nicht-mannigfaltige Objekte	benach-barte Flächen	Vor-gänger Kante	Nach-folger Kante
einzelne Dreiecke	36	Ja	$\mathcal{O}(n^2)$	mögl.	mögl.
shared vertex	18	Ja	$\mathcal{O}(n^2)$	mögl.	mögl.
winged-edge	60	nein	Ref.	Ref.	Ref.
half-edge	68	Nein	Ref.	Ref.	Ref.
directed-edge (gross)	68	Ja	Ref.	Ref.	Ref.
directed-edge (mittel)	44	Ja	Ref.	Ref.	$\mathcal{O}(1)$
directed-edge (klein)	32	Ja	Ref.	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Daten aus [CKS98]

Performance der 3 Speichervarianten



Fazit

- Die *Directed Edges Datenstruktur ist eine speichereffiziente Möglichkeit um mit Dreiecksnetzen zu arbeiten*
- *In der small size ist sie gerade einmal doppelt so gross wie die shared Vertex*
- *Dazu bietet sie einen konstanten Zugriff auf alle Nachbarschaftsbeziehungen*
- *Bei der Implementation empfielt es sich allerdings eine Schnittstelle bereitzustellen, die die internen Abläufe für den Programmierer versteckt, wie es z.B. bei OpenMesh[Ope] der fall ist.*

Vielen Dank für ihre Aufmerksamkeit


Baumgart, Bruce G.: *Winged Edge Polyhedron Representation*.
, Stanford University, Stanford, CA, USA, 1972.


Campagna, Swen, Leif Kobbelt Hans-Peter Seidel: *Directed Edges - A Scalable Representation for Triangle Meshes*.
Journal of Graphics Tools, 3, 1998.


OpenMesh Documentation.
http://openmesh.org/Documentation/OpenMesh-Doc-Latest/index.html.


Woo, Tony: *A Combinatorial Analysis of Boundary Data Structure Schemata*.
IEEE Comput. Graph. Appl, 5:19–27, 1985.