für meine Eltern

Repräsentation und Modellierung von regulären Objekten

Diplomarbeit
von
Wolfgang Collet

nach einem Thema von Prof. Dr. Günter Hotz
im Fachbereich 14, Informatik, der Universität des Saarlandes

Hiermit versichere ich an Eides Statt, daß ich diese Arbeit nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Saarbrücken, im Oktober 1994

Inhaltsverzeichnis

Abbildungsverzeichnis

Einleitung

Seit jeher ist der Mensch bestrebt, kulturelle Gegenstände herzustellen, aus der Motivation heraus, sich das Leben erträglicher zu gestalten.

Anfänglich waren es einfache Gegenstände: Jagdwerkzeuge, Kleider, Geschirr usw. Empirisches Wissen oder Wissen aus mündlicher Überlieferung reichte dazu aus, die Gegenstände direkt herzustellen.

Von Generation zu Generation wurden diese Objekte immer vielfältiger und komplexer in ihrem Aufbau. Man denke beispielsweise an die imposanten Gebäude alter Kulturen : Pyramiden, Paläste, Tempel, Theater, Wettkampfeinrichtungen, Brücken, Aquadukte usw.

Um nicht den Überblick in der *Konstruktionsphase* zu verlieren, erkannten die damaligen Bauherren, daß es wichtig war, sich Konstruktionspläne bereits im Vorfeld anzufertigen. Um aber das menschliche Gehirn nicht unnötig mit der Information dieser Pläne zu belasten, speicherte man sich diese — extern — in Schrift und Skizze. Diese Entlastung ermöglichte exaktes Arbeiten und sparte Zeit und Geld.

Diese soganannte *Planungsphase* hat sich zwar bis ins 20. Jahrhundert stetig weiterentwickelt und verbessert, jedoch blieb das Speichermedium prinzipiell das gleiche. Wer kennt beispielsweise nicht noch das Zeichenbrett?

Erst in den letzten Jahrzehnten unseres Jahrhunderts kam es auf diesem Gebiet zu einer Revolution. In den Planungs- und Ingenieurbüros traten die Zeichenbretter allmählich in den Hintergrund. Graphikterminals standen nun an ihrer Stelle. Ein neues Medium war geboren — **CAD** (**c**omuter **a**ided **d**esign). Die Vorteile des CAD gegenüber den traditionellen Methoden liegen u.a. in der virtuellen räumlichen Modellierung und Darstellung der herzustellenden Objekte, in der Datenverwaltung und der Datenverarbeitung.

Mit einem an einen Rechner angeschlossenen Graphikterminal und einer entsprechenden Software ist es dem Ingenieur möglich geworden, das zu konstruierende Objekt virtuell zu modellieren. Er kann es dann räumlich von allen Seiten betrachten. Wenn er es wünscht, kann er sogar virtuelle *Spaziergänge* in seinem Objekt (z.B. Gebäude) durchführen, um die Konstruktion im Detail zu untersuchen. Tausende von Daten über das Konstruktionsobjekt können überschaubar und schnell zugänglich im Rechner verwaltet werden. Durch Änderung von nur wenigen Parametern können Objekte leicht zu anderen Objekten modifiziert werden. Maßberechnungen wie z.B. Volumenberechnung an räumlichen Objekten können durch den Rechner in Sekundenschnelle durchgeführt werden. Der Ingenieur kann bewegte Objekte simulieren und diese auf Kollision untereinder untersuchen. Nicht zuletzt können mit Hilfe des CAD gleich ganze Arbeitsgänge eingespart werden.

Man denke beispielsweise an die Fertigung von Maschinenbauteilen oder Werkzeugen durch NC-Maschinen (Abk. für engl. **n**umerical **c**ontrol). Ursprünglich wurde für das zu fertigende Objekt zunächst eine technische Zeichnung angefertigt. Aus der Zeichnung und ihren Daten wurde dann ein für die NC-Maschine lesbares Programm durch einen erfahrenen Techniker geschrieben. Mit diesem Programm *gefüttert*, konnte dann die NC-Maschine das gewünschte Objekt fertigen. Heute kann das zu fertigende Objekt am Rechner modelliert werden. Die Daten des Modells werden durch den Rechner automatisch zu einem Programm für eine Fertigungsmaschine weiterverarbeitet, welche das Objekt unmittelbar nach Erhalt des Programms über eine an den Rechner angeschlossene Datenleitung, konstruieren kann. Solche Systeme faßt man unter dem Begriff *computerunterstützte Arbeitsvorbereitung* (**CAP** Abk. für engl. **c**omputer-**a**ided production **p**lanning) zusammen.

Ein weiterer Einsatzbereich des CAD finden wir auf dem Gebiet der Montageplanung. Diesbezüglich wurden — am Lehrstuhl von Prof. Dr. G. Hotz, an der Universität des Saarlandes — insbesondere auf Spezialgebieten wie Kollisionstests, Robotik und Motion Planning, räumliche Objekte benötigt, welche sich im Rechner durch eine *Boundary-Representation* (Randbeschreibung) eindeutig beschreiben lassen.

Aus diesem Anlaß beschäftigte ich mich zunächst in einem Fortgeschrittenen – Praktikum mit der Implementierung eines Konstruktionseditors zur Modellierung von primitiven Objekten (Rotationskörper, Prismen). In meiner Diplomarbeit schließlich habe ich mich ganz allgemein mit der *Repräsentation und Konstruktion von räumlichen Objekten* beschäftigt. Wir werden diese Objekte als reguläre Objekte [1] bezeichnen, welche wir durch konvexe Polyeder approximieren können. Dabei unterscheiden wir zwei Arten der Repräsentation für reguläre Objekte:

- Die **BSP-Representation** (**b**inary **s**pace **p**artitioning) wird mit Hilfe eines Bisektionsverfahrens erstellt. Sie bietet u.a. Vorteile in der geometrischen Darstellung und der Konstruktion regulärer Objekte.

- Die **Boundary-Representation** (Randdarstellung) beschreibt reguläre Objekte durch ihre Begrenzungselemente (Ecken, Kanten, Flächen). Sie bietet gegenüber der BSP-Representation u.a. Vorteile in der Volumenberechnung und in Kollisionstests regulärer Objekte. Nicht zuletzt ist diese Art der Repräsentation eine Standardschnittstelle .

[1] Definition *reguläres Objekt* siehe Kapitel 4 S. 60

Der Kern der Arbeit wird bestimmt durch zwei Basisoperationen.

- Die Operation **Split-Polyeder** splittet ein konvexes Polyeder bei gegebener Schnittebene in zwei konvexe Teilpolyeder in Zeit $O(n)$. Dabei ist n die Komplexität des zu splittenden Polyeders, gemessen in der Anzahl der Kanten des Polyeders.

- Mit Hilfe der Operation **Paste-Polyeder** (paste engl. kleben) werden konvexe Polyeder zu einem regulären Objekt zusammengeklebt.

Beide Operationen dienen als **Konvertierungsoperationen** zwischen den beiden Repräsentationsarten Boundary und BSP.

Die Algorithmen zu diesen beiden Operationen sind einfach und überschaubar. Dies garantiert eine robuste und leichte Implementierung. Weiterhin ist es uns möglich mit diesen beiden Operationen ein mächtiges Werkzeug auf dem Gebiet des CAD auf einfache Art und Weise zu entwickeln. Denn beide Elementaroperationen, *Split-* und *Paste-Polyeder*, sind mächtig genug zur Realisierung von **Booleschen Operationen** auf Polyedern. Anstöße und Anregungen zu dieser Arbeit stammen u.a. aus folgenden Veröffentlichungen: [FKN80], [TN87], [PY89], [Nay90], welche sich im wesentlichen mit der Repräsentation und Darstellung von geometrischen Objekten mit Hilfe des BSP-Trees beschäftigen. Definition und Analyse des BSP-Trees erfolgen in Kapitel 4.

Gliederung der Arbeit

In *Kapitel* 1 wird mit Hilfe einleitender Definitionen und Sätze bezüglich konvexer Mengen im $\mathbb{R}^d$, die Bezeichnung konvexes *Hyperpolyeder* (d = 1 : Strecke; d = 2 Polygon; d = 3 Polyeder) eingeführt. Auf dieser Grundlage wird gezeigt wie Polygon und Polyeder durch planare Graphen repräsentiert werden können. Schließlich wird in diesem Kapitel noch theoretisch gezeigt, wie eine Strecke, ein Polygon bzw. ein Polyeder durch eine Ebene gesplittet werden können. *In Kapitel* 2 werden zwei Datenmodelle – ein externes und ein internes Datenmodell – vorgestellt, um das Polyeder im Rechner zu repräsentieren. Weiterhin werden hier grundlegende Algorithmen bereitgestellt, zur Handhabung der Datenmodelle und einige grundlegende Algorithmen bzgl. Polygone und Polyeder. *In Kapitel* 3 wird ein effizientes Verfahren zur Implementation der Operation *Split-Polyeder* entwickelt. Es stellt einen wesentlichen Teil meiner Arbeit dar. *In Kapitel* 4 definieren wir den Begriff *reguläres Objekt* und den Begriff *Polyeder*, durch welchen sich ein beliebiges reguläres Objekt approximieren läßt. Weiterhin beschäftigen wir uns in diesem Kapitel mit *BSP-Bäumen*, mit welchen wir Polyeder (als Vereinigung von konvexen Teilpolyedern) repräsentieren können. Weiter werden einige wichtige Eigenschaften zu diesem BSP-Baum untersucht und einige grundlegende Algorithmen bzgl. dieses Baumes bereitgestellt. *In Kapitel* 5 beschäftigen wir uns primär mit einem effizienten Verfahren zur Implementation der Operation *Paste-Polyeder*, welches ebenfalls zum Kern meiner Arbeit gehört. Dabei soll gleichzeitig deutlich werden, daß durch dieses Verfahren ein in einer BSP-Representation gegebenes reguläres Objekt in die Boundary-Representation umgewandelt wird. Im letzten Abschnitt dieses Kapitels wird die umgekehrte Konvertierungsrichtung Boundary $\longrightarrow$ BSP mit Hilfe der Operation Split-Polyeder besprochen. In Kapitel 6 schließlich wird gezeigt, wie wir, mit Hilfe der beiden Basisoperationen Split-Polyeder und Paste-Polyeder, zwei Polyeder durch eine *Boolesche Operation* zu einem neuen Polyeder miteinander verknüpfen können. Im Anhang dieser Arbeit wird das Ergebnis einer Objektmodellierung, welche mit Hilfe des für diese Arbeit implementierten Konstruktionseditors erstellt wurde, demonstriert.

Implementation

Um den praktischen Aspekt dieser Arbeit deutlich zu machen, habe ich einen Konstruktionseditor zur geometrischen Modellierung von regulären Objekten programmiert. Dabei habe ich Wert darauf gelegt, die Arbeitsweise mit den beiden Basisoperationen Split- und Paste- Polyeder in den Vordergrund zu stellen.

Die Implementation wurde auf einer Workstation der Firma Hewlett - Packard durchgeführt. Als graphische Oberfläche diente X-Windows, darauf aufgesetzt OSF-Motif. Als Programmiersprache diente C++. Weiterhin bediente ich mich der Programmierbibliothek LEDA, welche mir eine effiziente Verwaltung von Suchbäumen und einfachen Listen ermöglichte. Der Quellcode meiner Implementation erreichte eine Größe von 1,4 MB was ca. 40000 Programmierzeilen entspricht.

Danksagungen

Ich möchte mich bei Herrn Prof. Dr. G. Hotz für die Vergabe der Diplomarbeit und für die interessante Themenstellung bedanken. Insbesondere möchte ich mich bei Herrn Dr. E. Schömer bedanken für die intensive Betreuung meiner Arbeit. Weiterer Dank gilt allen Studenten und Informatikern, die mir im Laufe meiner Tätigkeit am Lehrstuhl mit Rat und Tat zur Seite standen.

Kapitel 1

Einführung

Wie wir später in Kaptiel 4 sehen werden, lassen sich reguläre Objekte (Definition 4.1 S. 60) durch konvexe Polyeder approximieren, mit welchen wir uns zunächst beschäftigen wollen. Nach einigen einleitenden Definitionen und Sätze über Konvexität, orientierte Ebene, Räume, Umgebungen und Rand, definieren wir den Begriff **Hyperpolyeder** im Raum $\mathbb{R}^d$. ($d = 1$: Strecke; $d = 2$ Polygon; $d = 3$: Polyeder) Grob gesehen ist ein Hyperpolyeder ein geometrisches Gebilde, welches durch den Schnitt positiver Halbräume definiert wird. Anschließend folgen Definitionen und Sätze bzgl. des Randes eines Hyperpolyeders. Dabei stellen wir fest daß sich der rand eines Hyperpolyeders der Dimension d aus der vereinigung von Hyperpolyedern der Dimension $d - 1$ zusammensetzt. Im nachfolgenden Abschnitt zeigen wir, wie wir Polygone bzw. Polyeder durch eine Randbeschreibung repräsentieren können, welche wiederum durch Graphen repräsentiert werden können. Schließlich beschäftigen wir uns in diesem Kapitel noch mit dem Split eines Hyperpolyeders durch eine Hyperebene. Insbesondere wird theoretisch der Split einer Strecke, eines Polygones und eines Polyeders besprochen.

1.1 Theoretische Grundlagen

Definition 1.1 (Konvexität)
Eine Menge $M \subseteq \mathbb{R}^d$ heißt **konvexe** *Menge, wenn für 2 beliebige Elemente $\vec{x}_1$, $\vec{x}_2 \in M$ gilt:*
$\{ \ \vec{x} \ : \ \vec{x} = \vec{x_1} + \lambda \ (\vec{x}_2 - \vec{x}_1); \quad \lambda \in [0, 1] \ \} \subseteq M$.

Satz 1
$M_1, ..., M_n$ seien n konvexe Untermengen aus $\mathbb{R}^d$ $(d \in \mathbb{N})$. Dann ist auch ihr Durchschnitt $\bigcap_{i=1}^n M_i$ eine konvexe Untermenge aus $\mathbb{R}^d$.

Definition 1.2 (Hyperebene)
Sei $\vec{n} \in \mathbb{R}^d$ ein Vektor, $c \in \mathbb{R}$ ein Skalar, $(d \in \mathbb{N})$.
Dann verstehe man unter: $H := \{ \ \vec{x} \in \mathbb{R}^d \ : \ < \vec{n}, \vec{x} >= c \ \}$ eine **orientierte Hyperebene** *in $\mathbb{R}^d$;*

Diese Art der Ebenendarstellung bezeichnet man auch als allgemeine Normalform (**ANF** *). Dabei bezeichne man $\vec{n}$ als* **Normale** *zur Hyperebene H. Wir wollen in diesem Zusammenhang für die Ebene H folgende abkürzende Schreibweise verwenden. $H\ :\ (\vec{n}, c)$*

Definition 1.3 (Halbraum)
Sei $H \subseteq I\!R^d$ eine orientierte Hyperebene, spezifiziert durch die Normale $\vec{n} \in I\!R^d$ und dem Skalar $c \in I\!R$. Dann verstehe man unter:

$H_+ := \{\ \vec{x} \in I\!R^d \ :\ <\vec{n}, \vec{x}> \ > c\ \}$ **offener positiver Halbraum**
$H_- := \{\ \vec{x} \in I\!R^d \ :\ <\vec{n}, \vec{x}> \ < c\ \}$ **offener negativer Halbraum**
$H_+^\circ := \{\ \vec{x} \in I\!R^d \ :\ <\vec{n}, \vec{x}> \ \geq c\ \}$ **positiver Halbraum**
$H_-^\circ := \{\ \vec{x} \in I\!R^d \ :\ <\vec{n}, \vec{x}> \ \leq c\ \}$ **negativer Halbraum**

Bezeichnung 1
Speziell für $d = 2$ bezeichnet man in Definition 1.3 die orientierte Hyperebene als **orientierte (Rand-)Gerade** *und den Halbraum als* **Halbebene** *.*

Speziell für $d = 3$ bezeichnet man in Definition 1.3 die Hyperebene als **orientierte (Rand-)Ebene** *und den Halbraum als* **Halbraum** *.*

Bemerkung 2
Aus den beiden zuletzt aufgeführtenten Definitionen und der Definition für Konvexität (Def. 1.1) kann man leicht folgern, daß die oben aufgeführten Mengen : $H, H_+, H_+^\circ, H_-, H_-^\circ$ aus $I\!R^d$ konvexe Mengen sind.

Definition 1.4 (Umgebung)
Sei $\vec{p} \in I\!R^d$ ein beliebiger Raumpunkt und $\epsilon \in I\!R$ mit $\epsilon > 0$, dann versteht man unter der **Epsilon - Umgebung** *von $\vec{p}$ die Menge :*

$$U_\epsilon(\vec{p}) := \{\ \vec{x} \in I\!R^d \quad : \quad |\vec{x} - \vec{p}| < \epsilon\}$$

Definition 1.5 (Rand)
Sei $M \subseteq I\!R^d$ eine konvexe Menge, dann verstehe man unter dem **Rand** *(Boundary) von M die Menge :*

$$B_M = \{\ x \in I\!R^d \quad : \quad U_\epsilon(x) \cap M \neq \emptyset; \quad U_\epsilon(x) \cap \overline{M} \neq \emptyset; \quad \epsilon > 0\ \}$$

Dabei ist $\overline{M}$ die komplementäre Menge zu M derart, daß : $(M \dot\cup \overline{M}) = I\!R^d$

Definition 1.6 (konvexes Hyperpolyeder)

$H_1, ..., H_n$ seien n orientierte Hyperebenen aus $\mathbb{R}^d$, spezifiziert durch die Normalen $\vec{n}_1, ..., \vec{n}_n$ aus $\mathbb{R}^d$ und den Skalaren $c_1, ..., c_n$ aus $\mathbb{R}^d$.
$P \subseteq \mathbb{R}^d$ ist ein aus $H_1, ..., H_n$ konstruiertes, **konvexes Hyperpolyeder** *, genau dann wenn*

1. $P := \bigcap_{i=1}^{n} (H_i)_+^\circ$ *Polyeder*
2. $P \setminus \bigcup_{i=1}^{n} H_i \neq \emptyset$ *innere Punkte*
3. $\forall\ x \in P$ *gilt* $\exists\ \epsilon > 0$ *derart, daß* $P \subseteq U_\epsilon(x)$ *beschränkt*

Bezeichnung 3

Sei $P \subseteq \mathbb{R}^d$ ein Hyperpolyeder. für:

- $d = 1$ *dann bezeichne man P als (konvexe)* **Strecke**
- $d = 2$ *dann bezeichne man P als* **konvexes Polygon**
- $d = 3$ *dann bezeichne man P als* **konvexes Polyeder**

Definition 1.7
Sei $P \subseteq \mathbb{R}^d$ ein aus den orientierten Hyperebenen $H_1, ..., H_n$ konstruiertes, konvexes Hyperpolyeder.

Sei $x \in P \setminus \bigcup_{i=1}^{n} H_i$, dann heiße x **innerer Punkt** *des Hyperpolyeders P.*
Sei $x \in \overline{P} := \bigcup_{i=1}^{n} (H_i)_-$, dann heiße x **äußerer Punkt** *des Hyperpolyeders P.*

Satz 2
Sei $P \subseteq \mathbb{R}^d$ ein aus den orientierten Hyperebenen $H_1, ..., H_n$ konstruiertes, konvexes Hyperpolyeder.
B_P sei der Rand von P. Dann gilt:

$$B_P = \left(\bigcup_{i=1}^{n} H_i \right) \cap P$$

Satz 3
Sei $P \subseteq \mathbb{R}^d$ ein aus den orientierten Hyperebenen $H_1, ..., H_n$ konstruiertes, konvexes Hyperpolyeder.
Sei $F := f_1, ..., f_n = f_{H_1}, ..., f_{H_n} = (H_1 \cap P), ..., (H_n \cap P)$ die Menge der **Berandungsgebiete** *von P. Dann gilt:*

$$B_P = \bigcup_{i=1}^{n} f_i$$

Beweis zu Satz 3

$$B_P = \bigcup_{i=1}^{n} H_i \cap P = \bigcup_{i=1}^{n} (H_i \cap P) = \bigcup_{i=1}^{n} f_i$$

■

Satz 4
Sei $P \subseteq \mathbb{R}^d$ ein aus den orientierten Hyperebenen $H_1, ..., H_n$ konstruiertes, konvexes Hyperpolyeder.
Sei H_{n+1} eine weitere orientierte Hyperebene aus $\mathbb{R}^d$ derart, daß (nach Def. 1.6) $P^* := P \cap (H_{n+1})^\circ_+$ ein aus $H_1, ..., H_n, H_{n+1}$ konstruierter Hyperpolyeder ist. Wir definieren $H := H_{n+1}$.
Dann gilt:

$$B_{P^*} := \Big(B_P \cap (H^\circ_+\Big) \cup f_H$$

Beweis zu Satz 4

$$\begin{aligned} B_{P^*} &= \bigcup_{i=1}^{n+1} f_i = \bigcup_{i=1}^{n+1} (H_i \cap P^*) = \bigcup_{i=1}^{n} (H_i \cap P^*) \cup \underbrace{(H \cap P^*)}_{f_H} \\ &= \bigcup_{i=1}^{n} (H_i \cap (P \cap H^\circ_+)) \cup f_H \\ &= \bigcup_{i=1}^{n} ((H_i \cap P) \cap H^\circ_+) \cup f_H \\ &= \left(\bigcup_{i=1}^{n} (H_i \cap P) \cap H^\circ_+ \right) \cup f_H \\ &= \left(\bigcup_{i=1}^{n} f_i \cap H^\circ_+ \right) \cup f_H \\ &= (B_P \cap H^\circ_+) \cup f_H \end{aligned}$$

■

Korollar 1
Seien P, P^* wie in Satz 4 , dann gilt:

$$B_{P^*} = \left(\bigcup_{i=1}^{n} f_i \cap H_+^\circ \right) \cup f_H$$

Satz 5
Sei $P \subseteq \mathbb{R}^d$ ein aus den orientierten Hyperebenen $H_1, ..., H_n$ konstruiertes, konvexes Hyperpolyeder. F sei die Menge der Berandungsgebiete von P. Dann gilt:

$$B_P = \bigcup_{f \in F'} f$$

$$(F' := \{ f \in F : f \textit{ ist Hyperpolyeder der Dimension } d-1 \})$$

Bemerkung 4
Mit Satz 5 gilt speziell:

Eine (konvexe) Strecke ist eine von 2 Punkten A und B begrenzte konvexe Punktmenge auf einer Geraden.

Abbildung 1.1: Strecke

Ein konvexes Polygon ist eine durch eine endliche Anzahl von Strecken $e_1, ..., e_n$ begrenzte konvexe Punktmenge in der Ebene.

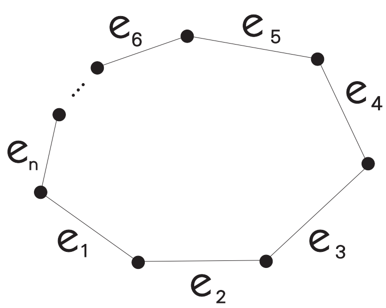

Abbildung 1.2: Konvexes Polygon

Ein konvexes Polyeder ist eine durch eine endliche Anzahl von konvexen Polygonen $Q_1, ..., Q_m$ begrenzte konvexe Punktmenge im Raum.

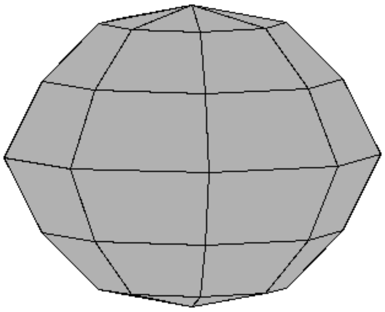

Abbildung 1.3: Konvexes Polyeder

Bezeichnung 5
Im folgenden bezeichne man die Begrenzungsstrecken eines Polygones als **Kanten** *des Polygones und deren begrenzende Punkte als* **Ecken** *des Polygones.*

1.2 Repräsentation eines konvexen Polygones durch einen Graphen

Sei Q ein Polygon wie in Abb. 1.2 mit den Begrenzungskanten $e_1, ..., e_n$ und den Ecken $v_1, ..., v_n$ derart, daß:

$$\begin{aligned} e_1 &= \{ \ \vec{x} \ : \ \vec{x} = \vec{v}_1 + \lambda \ (\vec{v}_2 - \vec{v}_1) \ \lambda \in [0,1] \} \\ &\vdots \\ e_n &= \{ \ \vec{x} \ : \ \vec{x} = \vec{v}_n + \lambda \ (\vec{v}_1 - \vec{v}_n) \ \lambda \in [0,1] \} \end{aligned}$$

Mit Abb. 1.2 wird deutlich, daß die Kanten bzw. die Ecken des Polygones den Rand von Q in einer bestimmten **zyklischen Ordnung** repräsentieren. Diesbezüglich können genau zwei zyklische Ordnungen definiert werden:

$$\begin{aligned} Knotenordnung1 : VO1_Q &:= (\underbrace{v_1, v_2}_{e_1}, ..., \underbrace{v_{n-1}, v_n}_{e_n}) \\ Kantenordnung1 : EO1_Q &:= (e_1, ..., e_n) \end{aligned}$$

$$Knotenordnung2 : VO2_Q := (\underbrace{v_n, v_{n-1}}_{e_n}, ..., \underbrace{v_2, v_1}_{e_1})$$

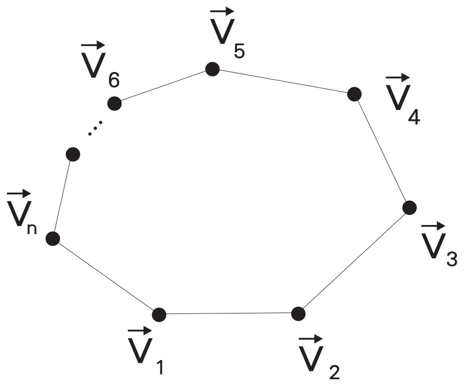

Abbildung 1.4: Knoten eines Polygones

$$Kantenordnung2 : EO2_Q \quad := \quad (e_n, ..., e_1)$$

Bezeichnung 6
Die oben auf den Ecken bzw. den Kanten definierten Ordnungen **VO** *bzw.* **EO** *bezeichne man als* **Knoten** - *bzw.* **Kanten-Ordnung** *eines Polygones. Knoten bzw. Kanten, welche in der Ordnung unmittelbar aufeinander folgen, heißen* **benachbart** .

Satz 6
Von einem Polygon Q sei eine Knotenordnung $VO_Q = (v_1, ..., v_n)$ gegeben,dann gilt: Der Rand B_Q von Q und Q werden eindeutig durch VO_Q repräsentiert.

Beweis

1. Bestimme $e_1, ..., e_n$; dann ist: $B_Q = \bigcup_{i=1}^{n} e_i$

2. Die Vektoren

$$\vec{t}_1 := (\vec{v}_2 - \vec{v}_1), \vec{t}_2 := (\vec{v}_3 - \vec{v}_2), \cdots, \vec{t}_n := (\vec{v}_1 - \vec{v}_n)$$

   sind in dieser Ordnung paarweise aufeinanderfolgend linear unabhängig.

3. Die aus den Vektorprodukten

$$\vec{n}_1 := (\vec{t}_1 \times (-\vec{t}_2)), \vec{n}_2 := (\vec{t}_2 \times (-\vec{t}_3)), \cdots, \vec{n}_n := (\vec{t}_n \times (-\vec{t}_1))$$

   entstehenden Vektoren sind orthogonal zu der Ebene, in welcher das Polygon Q liegt. Sei diese die orientierte Ebene H, dann zeigen alle diese Vektoren entweder

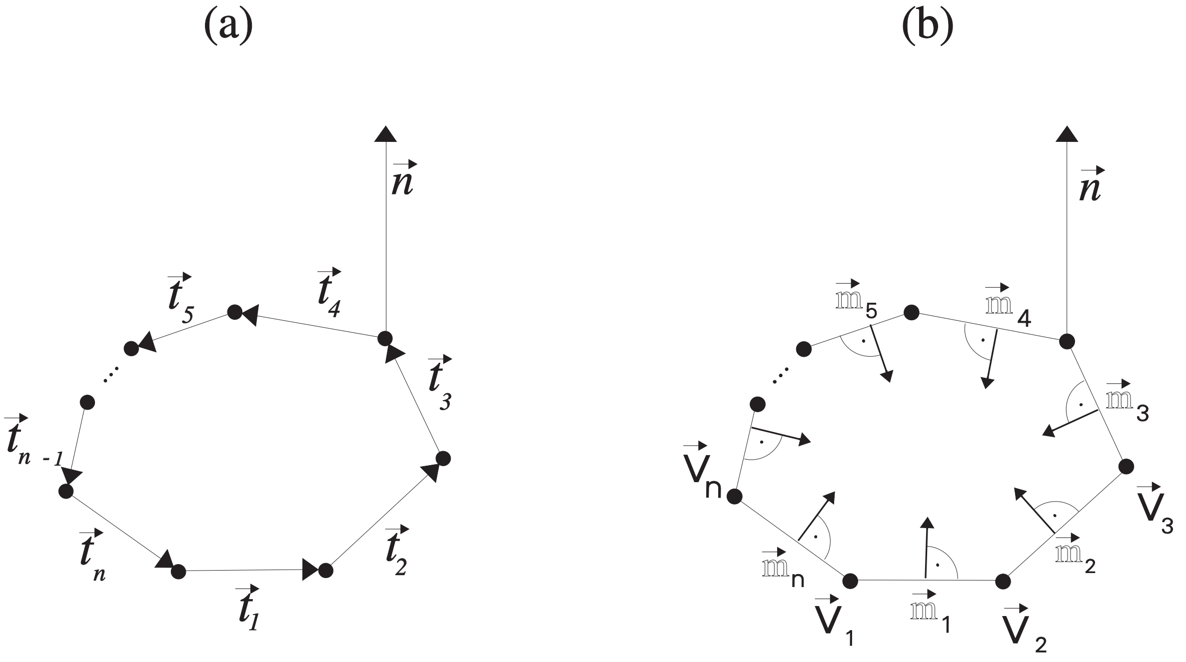

Abbildung 1.5: Orientiertes Polygon

in den Halbraum H_+ oder in den Halbraum H_-. Sei $\vec{n} \in \{\vec{n}_1, ..., \vec{n}_n\}$ einer dieser Vektoren. Wir bezeichnen diesen Vektor $\vec{n}$ als **Normalenvektor der Knotenordnung** VO_Q **des Polygones Q**. Die Ebene H sei dann gegeben durch den Normalenvektor $\vec{n}$ und dem Skalar $c := (\vec{n}, \vec{v}_1)$.

> Rechte Hand Regel:
> Zeigen die gekrümmten Finger der rechten Hand in die Richtung der geordneten Knoten bzgl. VO_Q, dann zeigt der Daumen der rechten Hand – orthogonal zum Zeigefinger derselben Hand – in die Richtung der Normalen der Knotenordnung des Polygones Q.

4. Die Vektoren

$$\vec{m}_1 := (\vec{n} \times (\vec{t}_1)), \vec{m}_2 := (\vec{n} \times (\vec{t}_2)), \cdots, \vec{m}_n := (\vec{n} \times (\vec{t}_n))$$

sind je orthogonal zu den Vektoren $\vec{t}_1, ..., \vec{t}_n$. Sie liegen in H und zeigen in das *Innere* des Polygones Q.

5. Wir erhalten dann die orientierten Geraden:

$$g_1 \quad = \quad \{ \; \vec{x} \; : \; < \vec{m}_1, \vec{x} > = < \vec{m}_1, \vec{v}_1 > \} \quad \cap \quad H$$

$$\begin{aligned} g_2 &= \{\ \vec{x}\ :\ <\vec{m}_2, \vec{x}> = <\vec{m}_2, \vec{v}_1>\} \quad \cap \quad H \\ &\vdots \\ g_n &= \{\ \vec{x}\ :\ <\vec{m}_n, \vec{x}> = \underbrace{<\vec{n}_n, \vec{v}_n>}_{1}\} \quad \cap \quad H \end{aligned}$$

Dann ist :

$$Q = \bigcap_{i=1}^{n} g_{i_+^\circ}$$

Somit bietet die Repräsentation eines Polygones durch eine bestimmte Knotenordnung, zusätzlich die Möglichkeit, auch Informationen bzgl. der Lage zu Halbräumen im $\mathbb{R}^3$ zu repräsentieren. Dies gibt den Anlaß zu der nun folgenden Definition.

Definition 1.8 (orientierte Knotenordnung)
Sei Q ein Polygon, repräsentiert durch die Knotenordnung $VO_Q = (v_1, ..., v_n)$. Q ist eingebettet in einer orientierten Ebene $H \in \mathbb{R}^3$.
$\vec{n}$ sei der Normalemvektor der Knotenordnung VO_Q gemäß Satz 6, und somit ist $\vec{n}$ auch orthogonal zur Ebene H. Zeigt $\vec{n}$, aufgehängt in einem Punkt der Ebene H, in den positiven Halbraum H_+ von H, dann bezeichne man VO_Q als die **zur Ebene H positiv orientierte Knotenordnung** *. Andernfalls bezeichne man VO_Q als die* **zur Ebene H negativ orientierte Knotenordnung** *.*

Bemerkung 7 (Kantenpartitionierung)
Sei $Q \subseteq \mathbb{R}^3$ ein Polygon, welches in der orientierten Ebene H liegt. Q werde durch die zu H positiv (negativ) orientierte Knotenordnung $VO = (v_1, ..., v_i, v_{i+1}, ..., v_m)$ repräsentiert. $(1 \leq i \leq m)$
Sei $x \in \mathbb{R}^3$, ein Punkt der Kante $e = (v_i, v_{i+1})$ wobei $x \neq v_i \wedge x \neq v_{i+1}$.
Dann wird Q auch durch die zu H positiv (negativ) orientierte Knotenordnung $VO^* = (v_1, ..., v_i, x, v_{i+1}, ..., v_m)$ repräsentiert.

Definition 1.9 (konvexes Polygon als planarer Graph)
Sei Q ein Polygon mit der Knoten-Ordnung $VO_Q = (v_1, ..., v_n)$,so läßt sich aus VO_Q ein planarer Graph G_Q wie folgt definieren, welcher Q eindeutig repräsentiert.

$$G_Q := (V, E, \varphi)$$

wobei:

[1]Skalar der Geraden g_n

$$\begin{array}{rcl} \varphi : E & \rightarrow & V \times V \\ & & \\ e_1 & \mapsto & (v_1, v_2) \\ e_2 & \mapsto & (v_2, v_3) \\ & \vdots & \\ e_{n-1} & \mapsto & (v_{n-1}, v_n) \\ e_n & \mapsto & (v_n, v_1) \end{array}$$

$$\begin{array}{lcl} Knotenmenge \; V & := & \{v_1 \cdots v_n\} \subseteq I\!R^3 \\ Kantenmenge \; E & := & \{e_1 \cdots e_n\} \end{array}$$

1.3 Repräsentation eines konvexen Polyeders durch einen Graphen

Satz 7
Sei $P \subseteq I\!R^3$ ein aus den orientierten Ebenen $\mathcal{H} = H_1, ..., H_n$ konstruiertes, konvexes Polyeder. $\mathcal{M} = \{Q_1 \cdots Q_m\}$ sei die Menge der Begrenzungspolygone, welche P von $\overline{P}$ abgrenzen. Weiterhin sei $\alpha : \mathcal{M} \rightarrow I\!N$ die Abbildung, welche jedem Polygon eindeutig die Anzahl seiner Knoten zuordnet.

Sei $H_{Q_i} \in \mathcal{H}$ die Ebene, in welcher Q_i eingebettet ist derart, daß $Q_i = P \cap H_{Q_i}$. $VO_{Q_i} = \{v_1, \cdots, v_{\alpha(Q_i)}\}$ sei die zur Ebene H_{Q_i} negativ orientierte Knotenordnung von Q. Dann läßt sich P eindeutig durch die Knotenordnungen $VO_{Q_1} \cdots VO_{Q_n}$ bzw. durch die daraus resultierenden planaren Graphen $G_{Q_1} \cdots G_{Q_n}$ repräsentieren.

Beweis : zu Satz 7

- Es genügt, den Beweis für die Knotenordnungen $VO_{Q_1} \cdots VO_{Q_n}$ durchzuführen, da Knotenordnung und planarer Graph äquivalente Polygonrepräsentationen sind.

- $VO_{Q_i} = (v_1, \cdots, v_{\alpha(Q_i)})$ sei die zur Ebene H_{Q_i} negativ orientierte Knotenordnung von Q_i. $(-\vec{n}_i)$ sei der nach Satz 6 konstruierte Normalenvektor von VO_{Q_i}, welcher aufgehängt in einem Punkt der Ebene H_{Q_i} in deren negativen Halbraum zeigt. Somit zeigt $\vec{n}_i$ in den positiven Halbraum der Ebene H.

- Mit v_i, einem beliebigen Knoten aus VO_{Q_i}, und dem Normalenvektor $\vec{n}_i$, läst sich die Ebene H_{Q_i} bestimmen derart, daß:

$$H_{Q_i} = \{\vec{x} \; : \; <\vec{n}_i, \vec{x}> = <\vec{n}_i, \vec{v}>\}$$

- Für den Polyeder P gilt dann:

$$P = \bigcap_{i=1}^{n} H^{\circ}_{+Q_i}$$

Satz 8
Sei $P \subseteq \mathbb{R}^3$ ein durch die planaren Graphen $G_{Q_1}, \cdots, G_{Q_n}$ repräsentiertes Polyeder. Dann läßt sich daraus ein zusammenhängender, gerichteter Graph $G_P = (V_P, E_P, \Delta)$ konstruieren. Darüber hinaus gilt :

- *In G_P gibt es keine Kantenüberschneidungen.*
- *G_P ist auf einer Kugel einbettbar.*

Lemma 1
Sei P ein Polyeder, Q_1, Q_2 seien 2 beliebige ungleiche Begrenzungspolygone von P. Dann trifft genau eine der folgenden 3 Aussagen zu:

1. $Q_1 \cap Q_2 = \emptyset$
2. $Q_1 \cap Q_2 = s$
   Dabei ist s Ecke von Q_1 und Q_2.
3. $Q_1 \cap Q_2 = e$
   Dabei ist e Kante von Q_1 und Q_2.

Beweis
Zu Q_1, Q_2 existieren 2 Ebenen H_{Q_1}, H_{Q_2} derart, daß $Q_1 = H_{Q_1} \cap P$ und $Q_2 = H_{Q_2} \cap P$.
Dabei gilt:

$$(Q_1 \neq Q_2) \;\Leftrightarrow (H_{Q_1} \cap P \neq H_{Q_2} \cap P) \Leftrightarrow (H_{Q_1} \neq H_{Q_2})$$

1. Fall: $H_{Q_1} \| H_{Q_2}$
   $\Rightarrow \; Q_1 \cap Q_2 = \emptyset \qquad$ (da $Q_1 \neq Q_2$)
2. Fall: $H_{Q_1} \not\| H_{Q_2}$
   $\Rightarrow \; g := H_{Q_1} \cap H_{Q_2}$ ist eine Gerade $\Rightarrow \; H_{Q_1} \cap H_{Q_2} \cap P = (Q_1 \cap Q_2)$ ist entweder eine Strecke oder ein Punkt, oder eine leere Punktmenge, da P durch eine Kugel begrenzt ist.

   (a) Sei $e := \overbrace{H_{Q_1} \cap H_{Q_2}}^{g} \cap P$ eine Strecke
   Annahme:
   $Q_1/g = \{Q_{1_+}, Q_{1_-}\}$
   $\Rightarrow Q_1/H_{Q_2} = \{Q_{1_+}, Q_{1_-}\}$
   $\Rightarrow Q_{1_-} \subseteq H_{(Q_2)^{\circ}_{-}}$
   $\Rightarrow Q_{1_-}$ ist kein Begrenzungspolygon von $P \qquad \Rightarrow$ Widerspruch

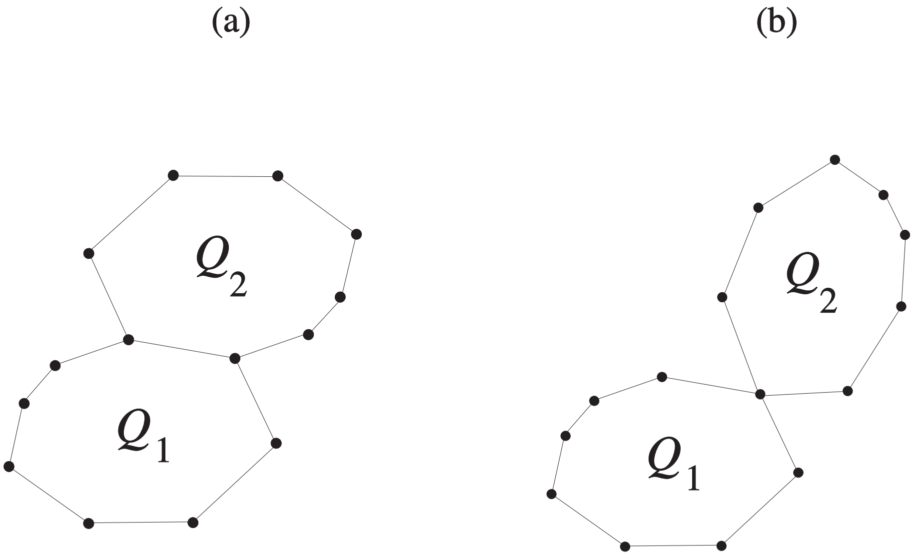

Abbildung 1.6: Benachbarte Begrenzungspolygone

$\Rightarrow e$ liegt auf dem Rand von Q_1 $(e \subseteq B_{Q_1}$
$\Rightarrow \exists\, e_{Q_1} \subseteq B_{Q_1} \quad \because \quad e \subseteq e_{Q_1}$
Weiterhin gilt:
$e_{Q_1} \subseteq g \qquad \text{und} \qquad e_{Q_1} \subseteq P$

$\Rightarrow e_{Q_1} \subseteq (g \cap P) = e$
$\Rightarrow e$ ist Kante von Q_1
analog gilt e ist Kante von Q_2

(b) Sei $s := (\overbrace{H_{Q_1} \cap H_{Q_2}}^{g} \cap P)$ ein Punkt

Analog wie oben leitet man her, daß: $s \subseteq B_{Q_1}$

$\Rightarrow \exists\, e_{Q_1} \subseteq B_{Q_1} \quad \because \quad s \subseteq e_{Q_1}$

Angenommen s ist weder Anfangs- noch Endpunkt der Kante e_{Q_1} dann gilt:

1. Fall $e_{Q_1} \| g$
   $\Rightarrow e_{Q_1} \subseteq (g \cap P) = s$
   $\Rightarrow e_{Q_1}$ ist ein Punkt — Widerspruch!
2. Fall $e_{Q_1} \nparallel g$
   $\Rightarrow Q_1/g = \{Q_{1+}, Q_{1-}\}$ — Widerspruch! (s.o. in Teil (a))

$\Rightarrow s$ ist Ecke von Q_1
(Analog beweist man s ist Ecke von Q_2.)

Lemma 2
Sei Q ein Polygon aus der Menge der das Polyeder P begrenzenden Polygone. Dann gilt:
An jede Kante des Polygones Q grenzt ein von diesem verschiedenes Begrenzungspolygon des Polyeders P.

Beweisidee:

Nach Lemma 1 grenzen Polygone höchstens in ihren Kanten aneinander. Annahme: es gibt eine Kante e in Q derart, daß kein anderes Begrenzungspolygon sich in dieser Kante schneidet. Damit ist jedoch der Rand B_P von P in unmittelbarer Nähe der Kante e *unstetig*. Dies führt zu einem Widerspruch.

■

Beweis zu Satz 8

$G_{Q_1}, \cdots, G_{Q_n}$ seien die planaren Graphen welche den Polyeder P repräsentieren derart, daß:

$$G_{Q_i} = (V_{Q_i}, E_{Q_i}, \varphi_{Q_i})$$

.
Sei $G_P = (V_P, E_P, \varphi)$ der zu konstruierende Graph, welcher P repräsentiert.
dann setze man:

$$V_P = \bigcup_{i=1}^{n} V_{Q_i} \qquad E_P = \bigcup_{i=1}^{n} E_{Q_i}$$

$$\varphi : E_P \to V_P \times V_P$$
$$e \mapsto \varphi(e)$$

wobei gilt: für $e \in E_P \quad \exists\, Q_i$ derart, daß $e \in E_{Q_i}$ und es gilt: $\varphi(e) = \varphi_{Q_i}(e)$.

■

Bezeichnung 8
Sei P ein konvexes Hyperpolyeder, repräsentiert durch den Graphen $G_P = (V_P, E_P, \varphi)$. Betrachten wir die Kanten $e_1, e_2 \in E_P$, welche aufgrund ihrer Richtung eindeutig zu einem Begrenzungspolygon Q des Polyeders P gehören. $v_a, v_b, v_c \in V_P$ seien ihre zugehörigen Knoten derart, daß:

$$\varphi(e_1) = (v_a, v_b) \quad \varphi(e_2) = (v_b, v_c)$$

Dann gilt:

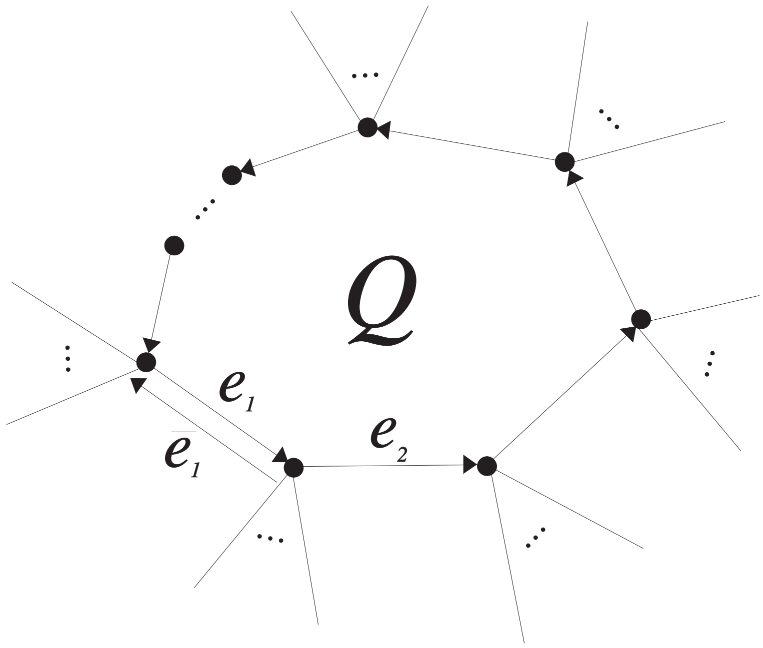

Abbildung 1.7: Graphenmodell des Polyeders P an der Begrenzungsfläche Q

- e_1 *ist die von* v_a *nach* v_b *gerichtete Kante*
- v_a *ist der* **Anfangsknoten** *von* e_1
- v_b *ist der* **Endknoten** *von* e_1
- v_a *ist* **Vorgängerknoten** *von* v_b — $pred(v_b) = v_a$
- v_b *ist* **Nachfolgerknoten** *von* v_a — $succ(v_a) = v_b$
- $\exists\ \overline{e_1} \in E_P$ *mit* $\varphi(\overline{e_1}) = (v_b, v_a)$;*i*
  $\overline{e_1}$ *heißt die zur Kante* e **korrespondierende Kante** — $co(e_1) = \overline{e_1}$
- e_1 *ist* **Vorgängerkante** *von* e_2 — $pred(e_2) = e_1$
- e_2 *ist* **Nachfolgerkante** *von* e_1 — $succ(e_1) = e_2$

1.4 Split

Definition 1.10
Sei $P \subseteq I\!R^d$ *ein aus den orientierten Hyperebenen* $H_1, ..., H_n$ *konstruiertes konvexes Hyperpolyeder.* H *sei eine weitere orientierte Hyperebene derart, daß:*

$$P_+ := H_+^\circ \cap P \neq P \quad \text{und} \quad P_- := H_-^\circ \cap P \neq P$$

sind zwei weitere konvexe Hyperpolyeder. Dann bezeichne man H *als* **Splitebene**, *welche* P *in die beiden Polyeder* $\mathbf{P}_+$ *und* $\mathbf{P}_-$ *splittet.*

kurz: $P/H := \{P_+, P_-\}$

Satz 9
Sei $P \subseteq \mathbb{R}^d$ *ein aus den orientierten Hyperebenen* $H_1, ..., H_n$ *konstruiertes, konvexes Hyperpolyeder.* H *sei eine weitere orientierte Hyperebene. Dann gelten die beiden folgenden, zueinander äquivalenten Aussagen:*

1.	$\begin{array}{ll} & \forall\, Knoten\; v\; von\; P\; gilt:\; v \in H_+^\circ \\ \vee & \forall\, Knoten\; v\; von\; P\; gilt:\; v \in H_-^\circ \end{array}$	$\Leftrightarrow$	*P wird nicht gesplittet durch H*
2.	$\begin{array}{l} \exists \quad Knoten\; v_1, v_2\; derart,\; daß \\ \quad v_1 \in H_+ und v_2 \in H_- \end{array}$	$\Leftrightarrow$	$P/H = \{P_+, P_-\}$

Beweisidee für d = 3
Es reicht aus, die erste der beiden Behauptungen zu zeigen, da ihre Äquivalenz offensichtlich ist.
"$\Rightarrow$"

$$
\begin{array}{lll}
Annahme: & & \forall\, Knoten\; v\; von\; P\; gilt:\; v \in H_+^\circ \\
 & & \\
 & \Rightarrow & \forall\, Kanten\; v\; von\; P\; gilt:\; e \in H_+^\circ,\; da\; H_+^\circ\; konvex \\
 & \Rightarrow & \forall\, Begrenzungspolygone\; f\; von\; P\; gilt\;\; B_f \subset H_+^\circ \\
 & \Rightarrow & \forall\, Begrenzungspolygone\; f\; von\; P\; gilt\;\; f \subset H_+^\circ,\; da\; H_+^\circ\; konvex \\
 & \Rightarrow & P \subset H_+^\circ \\
 & \Rightarrow & P\; wird\; durch\; H\; nicht\; gesplittet
\end{array}
$$

"$\Leftarrow$"
Wird andererseits P von H nicht gesplittet, dann gilt:
entweder $P \subset H_+^\circ$ oder $P \subset H_-^\circ$.
Somit liegen auch alle Knoten von P entweder in H_+° oder in H_-°.

■

Es folgen drei Splitbeschreibungen, speziell für die konvexen Hyperpolyeder Strecke, Polygon und Polyeder.

1.4.1 Streckensplit

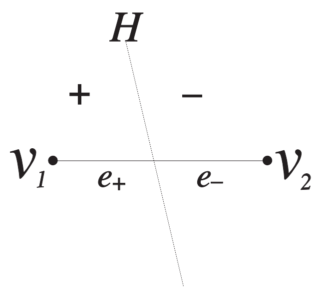

Abbildung 1.8: Streckensplit

Sei $e = (v_1, v_2) \subseteq \mathbb{R}^3$ eine Strecke. $H\ :\ (\vec{n}, c)$ sei eine orientierte Ebene in $\mathbb{R}^3$, welche die Strecke e in die beiden Teilstrecken e_+ und e_- gemäß Definition 1.10 splittet. ($e/H\ =\ \{e_+, e_-\}$). Damit existiert ein Schnittpunkt $\vec{s}\ =\ \{\vec{v}_1 + \lambda(\vec{v}_2 - \vec{v}_1)\}$, wobei $\lambda \in [0, 1]$. Der Schnittpunkt $\vec{s} = (e \cap H)$ berechnet sich dann wie folgt.

$$H = \{\vec{x} \in \mathbb{R}^3\ :< \vec{n}, \vec{x} >= c\,\} \tag{1}$$

$$e = \{\vec{x} \in \mathbb{R}^3\ :\ \vec{x} = \vec{v}_1 +\ \lambda\,(\vec{v}_2 - \vec{v}_1)\ ;\ \lambda\ \in [0, 1]\ \} \tag{2}$$

dann gilt:

$$< \vec{n}, \vec{s} > \quad = c \quad \Leftrightarrow \quad < \vec{n},\ \vec{v_1} + \lambda \overbrace{(\vec{v}_2 - \vec{v}_1)}^{\vec{v}} > \quad = c \quad \Leftrightarrow \quad \lambda = \frac{c- < \vec{n}, \vec{v}_1 >}{\underbrace{< \vec{n}, \vec{v} >}_{\neq 0}}$$

Annahme $< \vec{n}, \vec{v} > \ = \ 0$

> Dann ist $\vec{n}$ orthogonal zu $\vec{v} = (\vec{v_2} - \vec{v_1})$. Somit liegt e parallel zur Ebene H, was der Ausgangssituation **widerspricht**.

Schließlich erhält man $\vec{s}$ aus der Kantengleichung(2):

$$\vec{s} \;=\; \vec{v}_1 + \frac{c- < \vec{n}, \vec{v}_1 >}{< \vec{n}, \vec{v} >} \cdot \vec{v}$$

1.4.2 Polygonsplit

Sei $Q \subseteq \mathbb{R}^2$ ein Polygon, repräsentiert durch die Knotenordnung $VO_Q = (v_1, \cdots, v_n)$. Q werde durch die orientierte Gerade g in Q_+ und Q_- gesplittet. Dann schneidet g den Rand von Q in zwei verschiedenen Kanten $e_i := (v_i, succ(v_i))$, $e_j := (v_j, succ(v_j))$ von Q. Man erhält die beiden Schnittpunkte:
$\vec{x_1} := \{\vec{v_i} + \lambda_1(succ(\vec{v_i} - \vec{v_i}\}$ $\vec{x_2} := \{\vec{v_j} + \lambda_1(succ(\vec{v_j} - \vec{v_j}\}$ wobei $\lambda_1, \lambda_2 \in [0, 1]$

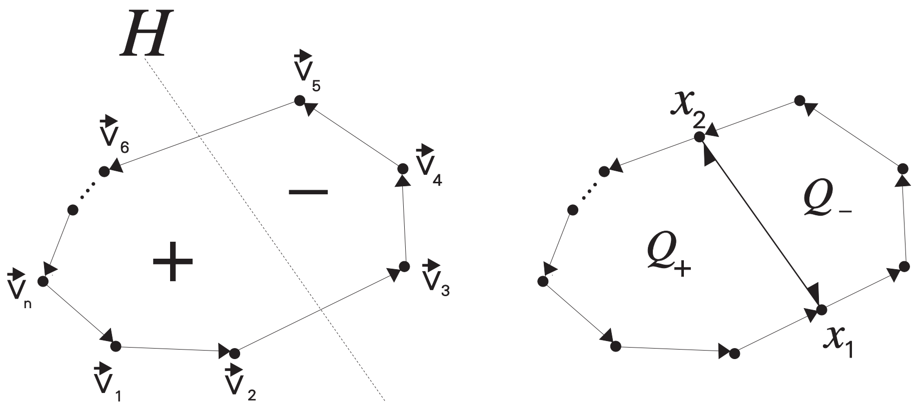

Abbildung 1.9: Polygonsplit

Falls $\vec{x_1}$ nicht identisch ist mit einem Knoten von Q, dann erweitere man die Knotenordnung VO_Q um den Knoten $\vec{x_1}$. Analog verfahre man mit $\vec{x_2}$. Man erhält die modifizierte Knotenordnung : $VO_Q^* := \{v_1, \cdots, x_1, \cdots, x_2, \cdots, v_n\}$
In Bezug auf Abbildung 1.9 erhält man somit:

$$\begin{aligned} VO_{Q_+} &:= (v_1, v_2, x_1, x_2, \cdots, v_n) \\ VO_{Q_-} &:= (x_1, v_3, v_4, v_5, x_2) \end{aligned}$$

Damit splittet die Gerade g die Knotenordnung VO_Q in die Knotenordnungen VO_{Q_+} bzw. VO_{Q_-}. Den zugehörigen Algorithmus finden wir in Abschnitt 2.5 auf Seite 30

$$kurz: \quad VO_Q/g = \{VO_{Q_+}, VO_{Q_-}\}$$

Bemerkung 9
Mit Satz 4 Seite 8 gilt:

$$
\begin{aligned}
B_{Q_+} &= (B_Q \cap g_+^\circ) \cup \underbrace{f_g}_{(x_1,x_2)} \\
B_{Q_-} &= (B_Q \cap g_-^\circ) \cup \underbrace{f_g}_{(x_1,x_2)}
\end{aligned}
$$

Bemerkung 10 (Flächenpartitionierung)
Sei P ein aus den orientierten Ebenen $\mathcal{H} = H_1, ..., H_n$ konstruiertes Polyeder. P werde durch die Knotenordnungen $VO_1, VO_2..., VO_n$ repräsentiert. Q sei das Polygon, welches durch die Knotenordnung VO_1 repräsentiert wird. Gemäß Abschnitt 1.4.2 werde Q durch eine Gerade g in die Polygone Q_1 und Q_2 gesplittet, indem die Knotenordnung VO_1 in die beiden Knotenordnungen $VO_{1.1}$ und $VO_{1.2}$ gesplittet werden. Dabei sind die beiden neuen Knotenordnungen gleich orientiert wie VO_1. Dann wird P ebenfalls repräsentiert durch die Knotenordnungen $VO_{1.1}, VO_{1.2}, VO_2, ..., VO_n$.

1.4.3 Polyedersplit

Sei $P \subseteq \mathbb{R}^3$ ein Polyeder, repräsentiert durch die Knotenordnungen $VO_{Q_1}, \cdots, VO_{Q_n}$ der Polygone $Q_1, \cdots, Q_n$. Sei H die Ebene, welche P in die Polyeder P_+ und P_- splittet. Dann gilt:

$$f_H := H \cap P \textit{ ist ein konvexes Polygon}$$

.
Denn P entspricht dem Schnitt positiver Halbräume. Diese bewirken in H einen Schnitt positiver Halbebenen. Damit folgt obige Behauptung unmittelbar aus der Definition 1.6 für Hyperpolyeder.
Weiterhin gilt:

$$f_H = H \cap P_+ (= H \cap (P \cap H_+^\circ)) = H \cap P$$

Mit Hilfe der Fallunterscheidung in Abbildung1.10 können wir die Knotenordnungen der Begrenzungspolygone von P in 3 disjunkte Mengen PGN_+, PGN und PGN_- wie folgt einteilen:

$$B_P \; : \; PGN_+ \cup PGN \cup PGN_- = \bigcup_{i=1}^{n} VO_{Q_i}$$

PGN_+ : *Knotenordnungen der Begrenzungspolygone von P, welche in H_+° liegen. (Fälle 1 bis 3 in Abbildung 1.10)*

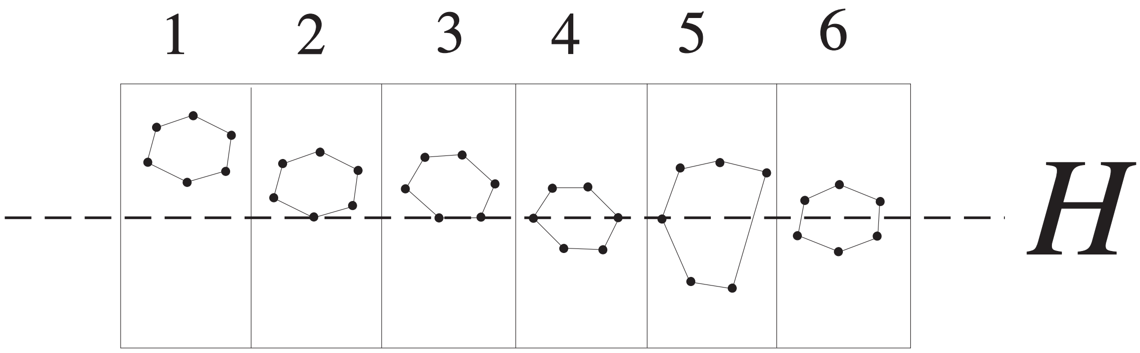

Abbildung 1.10: Betrachtung möglicher Polygonschnitte

PGN : *Knotenordnungen der Begrenzungspolygone von P, deren Schnitte, sowohl mit H_+° als auch mit H_-°, nicht leer sind.* (*Fälle* 4 *bis* 6 *in Abbildung* 1.10)

PGN_- : *analog zu* PGN_+

Sei $VO_Q \in PGN$. H_Q sei die Ebene, in welcher Q eingebettet ist. Da sowohl $(Q \cap H_+) \neq \emptyset$ als auch $(Q \cap H_-)$, existiert existiert die Gerade $g_Q := H_Q \cap H$. Damit teilt g_Q die Ebene H_Q entsprechend zur Normalen der orientierten Ebenen H in eine positive und eine negative Halbebene. Somit splittet die Gerade g_Q das Polygon Q in die beiden Polygone Q_+ und Q_- bzw. die Knotenordnung VO_Q nach Abschnitt 1.4.2 in VO_{Q_+} und VO_{Q_-}. Damit kann man die Menge PGN aufteilen in die beiden zueinander disjunkten Mengen: PGN_+^* und PGN_-^*

PGN_+^* :
Menge der durch Split der Knotenordnungen aus PGN neu entstandenen Knotenordnungen, für deren Polygone Q, welche sie repräsentieren, gilt: $Q \in H_+$

PGN_- : analog zu PGN_+

Da beim Polygonsplit die Orientierung der Polygone beibehalten wird, werden P bzw. B_P auch durch die modifizierte Menge der Knotenordnungen repräsentiert.

$$B_P : \underbrace{(PGN_+ \cup PGN_+^*)}_{+} \cup \underbrace{(PGN_-^* \cup PGN_-)}_{-}$$

Mit Satz 4 Seite 8 gilt:

$$B_{P_+} \;=\; (B_P \cap H_+^\circ) \cup \underbrace{f_H}_{H \cap P_+}$$

$$B_{P_-} \;=\; (B_P \cap H_-^\circ) \cup \underbrace{f_H}_{H \cap P_-}$$

Daraus folgt:

P_+ wird repräsentiert durch die Knotenordnungen der Menge

$$P_+ \;:\; \qquad PGN_+ \cup PGN_+^* \cup VO_{+_{f_H}}$$

P_- wird repräsentiert durch die Knotenordnungen der Menge

$$P_- \;:\; \qquad PGN_- \cup PGN_-^* \cup VO_{-_{f_H}}$$

Dabei muß nach Satz 7 auf Seite 14 $VO_{+_{f_H}}$ die zur Ebene H negativ orientierte Knotenordnung und $VO_{-_{f_H}}$ die zur Ebene H positiv orientierte Knotenordnung bzgl. des Polygones f_H sein.

Kapitel 2

Datenmodelle für das Polyeder

Wollen wir mit Polyedern auf dem Rechner arbeiten, so benötigen wir geeignete Datenmodelle. Wir unterscheiden bewußt zwischen 2 Arten von Datenmodellen — das **externe Datenmodell** und das **interne Datenmodell**. Das externe Datenmodell dient zur Speicherung der das Polyeder repräsentierenden Randbeschreibung auf einem externen Datenträger. Das interne Datenmodell dient zur Speicherung der das Polyeder repräsentierenden Randbeschreibung im Hauptspeicher des Rechners, um während des laufenden Programmes einen möglichst schnellen Zugriff auf die Daten des Polyeders zu haben. Dies erreicht man durch eine geschickte Verpointerung der Begrenzungsobjekte (Knoten, Kanten, Flächen) des Polyeders untereinander, womit — im Gegensatz zum externen Datenmodell — die Nachbarschaftsbeziehungen dieser Objekte direkt beschrieben werden. Wir werden weiter in Abschnitt 2.3 erfahren, wie wir aus dem externen Datenmodell das interne Datenmodell gewinnen können. Umgekehrt läßt sich das externe Datenmodell direkt aus dem internen Datenmodell in linearer Zeit ablesen. In Abschnitt 2.4 leiten wir uns ein Maß für die Komplexität des konvexen Polyeders her. Schließlich nutze ich dieses Kapitel noch dazu, um einige zwar bereits bekannte Algorithmen für Polgone und Polyeder zu beschreiben, welche mir jedoch im Zusammenhang meiner Arbeit als erwähnenswert erscheinen. So wird u.a. ein effizientes Verfahren zum Split eines Polygones und ein effizientes Verfahren zur Berechnung des Schnitts zweier Hyperpolyeder aufgezeigt.

2.1 Externes Datenmodell

Um so wenig Platz wie möglich zu verschwenden, sollte das das Polyeder repräsentierende externe Datenmodell möglichst kompakt und nicht redundant sein. Das Polyeder P werde repräsentiert durch die Knotenordnungen $VO_1, ..., VO_n$ seiner Begrenzungsflächen $f_1, ..., f_n$. Als geometrische Information merken wir uns für jeden Knoten v von P eine Identität $id(v)$ und seine eindeutigen kartesischen Koordinaten bezüglich des zugrundeliegenden Weltkoordinatensystems. Als topologische Information merken wir uns für jedes Begrenzungspolygon f_i, seine Knotenordnung $VO_i = (v_{i1}, ..., v_{im})$, indem wir uns für jeden Knoten dieser Ordnung nur seine Identität merken, also ($id(v_{i1})$, ..., $id(v_{im})$). Betrachten wir dazu das externe Datenmodell am Beispiel eines Würfels.

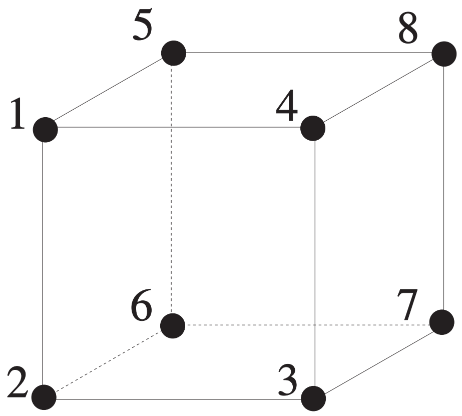

Abbildung 2.1: Würfel

geometrische Information
(Knoten des Würfels)

$$
\begin{aligned}
v_1 &= (x_1, y_1, z_1)\\
v_2 &= (x_2, y_2, z_2)\\
v_3 &= (x_3, y_3, z_3)\\
v_4 &= (x_4, y_4, z_4)\\
v_5 &= (x_5, y_5, z_5)\\
v_6 &= (x_6, y_6, z_6)\\
v_7 &= (x_7, y_7, z_6)\\
v_8 &= (x_8, y_8, z_8)
\end{aligned}
$$

topogische Information
(Begrenzungsflächen des Würfels)

$$
\begin{aligned}
f_1 &= (1, 2, 3, 4)\\
f_2 &= (4, 3, 7, 8)\\
f_3 &= (5, 8, 7, 6)\\
f_4 &= (1, 5, 6, 2)\\
f_5 &= (1, 4, 8, 5)\\
f_6 &= (2, 6, 7, 3)
\end{aligned}
$$

$x_i, y_i, z_i \in \mathbb{R}^3$
Koordinaten der Knoten

2.2 Internes Datenmodell

Um die Laufzeiten der von Polyederdaten abhängigen Algorithmen nicht unnötig zu erhöhen, sollte das interne Datenmodell des Polyeders schnellen Datenzugriff auf die das Polyeder repräsentierenden Daten ermöglichen. Die während der Laufzeit eines Programms benötigten Polyeder werden linear in einer Liste gespeichert. Knoten, Kanten und Flächen eines Polyeders werden ebenfalls je in einer linearen Liste gespeichert, auf welche das zugehörige Polyeder durch einen Adreßzeiger direkten Zugriff hat. Polyeder, Knoten, Kanten und Flächen werden eindeutig identifiziert und können bei Bedarf markiert werden. Jeder Knoten besitzt direkten Zugriff zu seinen Koordinaten, welche das Polyeder in seiner geometrischen Lage festlegen. Jede Kante zeigt auf ihren Anfangs- und ihren Endknoten, hat direkten Zugriff zu ihrer Vorgänger- und ihrer Nachfolgerkante und zeigt auf die Polygonfläche, welcher sie eindeutig angehört. Jede Polygonfläche f des Polyeders zeigt auf eine Kante e, welche dem Polygon f eindeutig zugeordnet werden kann. Folgt man sukzessive den Nachfolgern dieser Kante e , so erhält man die Kantenordung der Polygonfläche f, in Zeit Komplexität dieser Polygonfläche. Um einen möglichst schnellen Zugriff auf die Nachbarflächen einer Polgonfläche zu haben, hat jede Kante e einen Zeiger auf die ihr korrespondierende Kante $\overline{e}$. Jeder Knoten v

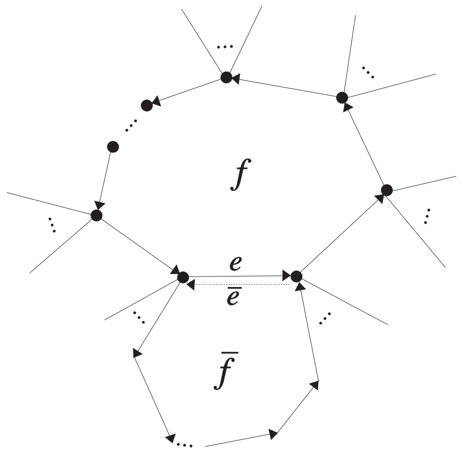

Abbildung 2.2: Zugriff zur Nachbarfläche

zeigt auf eine Kante e , welche ihn entweder als Anfangs- oder als Endknoten enthält. Mit Hilfe der Kantenzeiger auf Nachfolgekanten beziehungsweise auf korrespondierende Kanten kann man somit alle Eingangs- und Ausgangskanten des Knotens in einer Zeit proportional zum Grad des Knotens erreichen. Ebenso lassen sich in derselben Zeit alle Flächen bestimmen, welche den Knoten umgeben und diesen selbst als Knoten (Ecke) besitzen. Will man schließlich alle Objekte (Knoten, Kanten, Flächen) des Polyeders P besuchen, so gibt es hierfür grundsätzlich 2 Möglichkeiten:

1. Möglichkeit
   Durchlaufe–(linear)–Knoten, Kanten und Flächenliste von P.
   Laufzeit: $O(n_v + n_e + n_f) \leq O(n_e)$
   Dabei seien n_v, n_e, n_f die Anzahl der Knoten, Kanten und Flächen des Polyeders. (Siehe auch Abschnitt 2.4 *Komplexität eines konvexen Polyeders*.

2. Möglichkeit
   Durchlaufe (linear) die Flächenliste. Für jede Fläche f durchlaufe man alle Kan-

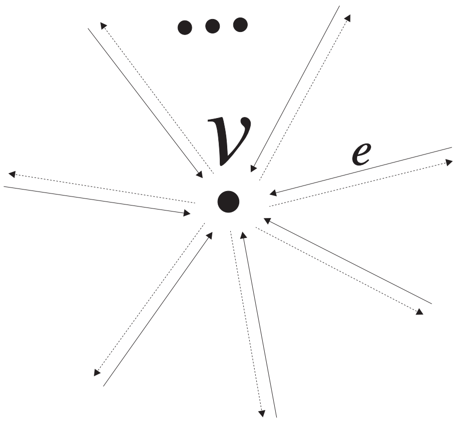

Abbildung 2.3: Ein- und Auskanten eines Knotens

ten von f und für jede Kante e auch ihre Knoten.
Laufzeit: $O(n_v + n_e + n_f) \leq O(n_e)$

Bezeichnung 11 (reguläre Boundary-Representation)
Wir wollen die Boundary-Representation eines konvexen Polyeders als regulär *bezeichnen, wenn folgende beiden Bedingungen erfüllt sind.*

- *Aufeinanderfolgende Kanten einer Fläche sind linear unabhängig.*

- *Zwei in einem Punkt oder in einer Kante benachbarte Flächen liegen in verschieden Ebenen.*

2.3 Algorithmus zum Einlesen des externen Datenmodells in das interne Datenmodell

Sei P ein konvexes Polyeder, gegeben in Form des externen Datenmodells. Wir lesen diese Daten in Zeit $O(n_e \cdot log(n_e))$ in den Hauptspeicher des Rechners als internes Datenmodell ein. Dabei sei n_v Anzahl der Knoten, n_e Anzahl der Kanten, n_f die Anzahl der Flächen, und es ist nach Abschnitt 2.4 zu beachten, daß gilt: $n_v \leq n_e$ und $n_f \leq n_e$. Wir gehen dabei in 3 Teilschritten vor.

1. Einlesen der Knoten
   Wir sammeln alle Knoten von P erstens in einer Knotenliste L_{vertex} in Zeit $O(n_v)$ und zweitens in einem balancierten Baum mit der Identität der Knoten als Schlüssel und der (Hauptspeicher-) Adresse der Knoten als Information. Letzeres kostet: $O(\sum_{i=1}^{n_v} log(n_v)) \leq O(n_v \cdot log(n_v))$

2. Einlesen von Flächen und Kanten
   Dabei sammeln wir die Flächen in einer Flächenliste L_{face}. Für jede einzelne Fläche f lese man die Knotenidentitäten dieser Fläche in Knotenordnung und mache folgendes:

   (a) Für paarweise aufeinanderfolgende Knotenordnungen $id(v_1)$, $id(v_2)$ speichere man die entsprechend zugehörige Kante $e = (v_1, v_2)$ und hänge diese in eine Kantenliste L_{edge}. Dabei bestimme man die Knoten v_1, v_2 aus dem balancierten Knotenbaum in Zeit $O(log(n_v))$.

   (b) Aufeinanderfolgende Kanten verpointere man miteinander.

   (c) Verpointere f mit einer beliebigen Kante aus f.

   (d) Verweise jeden Knoten auf eine Kante, welche diesen als Anfangs- oder Endknoten besitzt.

   (e) Verweise jede Kante e von f auf auf die Fläche f.

   (f) Für f merke man sich die Anzahl der Kanten von f.

   Für diesen Schritt verbrauchen wir also eine Laufzeit von $O(n_e \cdot log(n_e))$

3. Bestimmen der korrespondierenden Kanten
   Nun müssen wir noch jeder Kante e die ihr korrespondierende Kante $\overline{e}$ zuordnen. Dazu sortieren wir die Kanten in der Kantenliste L_{edge} nach Kanten, welche die gleiche Strecke repräsentieren.
   Laufzeit: $O(n_e \cdot log(n_e))$
   Danach liegen korrespondierende Kanten in L_{edge} nebeneinander. Erneut besuche man alle Kanten in L_{edge} und verpointere die korrespondierenden Kanten miteinander. Laufzeit: $O(n_e)$

2.4 Komplexität eines konvexen Polyeders

Maßgebend für die Komplexität eines konvexen Polyeders P ist die Anzahl seiner Knoten n_v, Kanten n_e und Flächen n_f. Dabei zählen wir mit n_e die Anzahl der gerichteten Kanten gemäß unserem Datenmodell für Polyeder und somit zu jeder Kante e auch die ihr korrespondierende Kante $\overline{e}$ mit. Demgemäß gehört jede Kante e eindeutig zu einer Fläche f. Da eine Fläche (konvexes Polygon) wenigstens 3 Kanten besitzt, gilt:

$$3 \cdot n_f \leq n_e$$

Weiterhin wissen wir, daß eine Fläche (konvexes Polygon) ebensoviele Knoten wie Kanten besitzt. Da jedoch ein Knoten, Ecke von wenigstens 3 Flächen ist gilt:

$$3 \cdot n_v \leq n_e$$

Damit haben wir in n_e ein Maß für die Komplexität des konvexen Polyeders. Letztlich wird diese Feststellung noch unterstützt durch den Eulerschen Polyedersatz der besagt:

$$e - f + 2 = v$$

wobei $e := \frac{n_e}{2}$ die Anzahl der ungerichteten Kanten des konvexen Polyeders und $v := n_v$ und $f := n_f$.

2.5 Algorithmen für konvexe Hyperpolyeder

2.5.1 Splitalgorithmus für konvexe Polygone

Sei $P \subseteq \mathbb{R}^3$ ein konvexes Polygon, repräsentiert durch die Knotenordnung $VO = (v_1, ..., v_n)$. Wir gehen dabei davon aus, daß aufeinanderfolgende Kanten des Polygones P, linear unabhängig sind. P liege in der orientierten Ebene H_P, zu der VO positiv orientiert ist. P soll durch die orientierte Ebene H — gegeben durch das Skalar c und der Normalen $\vec{n}$ — gesplittet werden. Dazu bieten sich zwei Verfahren an.

Erste Möglichkeit

Das erste Verfahren splittet P in O(n) wie folgt:
In Knotenordnung durchlaufen wir alle Knoten von P in Zeit O(n) und führen für jeden Knoten v folgende 3 Schritte in Zeit O(1) durch:

1. Bestimme für Knoten v seine Lage bzgl. H .

$$\begin{array}{lllll} v \in & H_+ & \Leftrightarrow & <\vec{n}, \vec{v}> & > & c \\ v \in & H & \Leftrightarrow & <\vec{n}, \vec{v}> & = & c \\ v \in & H_- & \Leftrightarrow & <\vec{n}, \vec{v}> & < & c \end{array}$$

2. Liegt ein Knoten in H_+°, so hängen wir diesen in eine Liste L_+.
   Liegt ein Knoten in H_-°, so hängen wir diesen in eine Liste L_-.

3. Liegen zwei aufeinanderfolgende Knoten v_1, v_2 auf verschiedenen Seiten der Ebene H, so berechnet sich der eindeutige Schnittpunkt $\vec{s}$ aus folgender Formel, deren Herleitung man auf Seite 20 nachlesen kann. Wir definieren dabei $\vec{v} := (\vec{v_2} - \vec{v_1})$.

$$\boxed{\vec{s} = \vec{v_1} + \frac{c- <\vec{n}, \vec{v_1}>}{<\vec{n}, \vec{v}>} \cdot \vec{v}}$$

Wäre $< \vec{n}, \vec{v} > \;=\; 0$, dann wäre $\vec{n}$ orthogonal zu $(\vec{v}_2 - \vec{v}_1)$ und die Strecke $(\vec{v_2} - \vec{v_1})$ läge parallel zur Ebene H, was der Ausgangssituation widerspricht. Weiterhin hängen wir $\vec{s}$ als neuen Knoten, sowohl in die Liste L_+ als auch in die Liste L_-.

Mit der folgenden Fallunterscheidung können wir das gewünschte Ergebnis in Zeit $O(1)$ bestimmen.

1. $P \subseteq H_+^\circ \Leftrightarrow \text{Size}(L_-) \leq 2$ (kein Split)

2. $P/H = \{P_+, P_-\} \Leftrightarrow (\ (\text{Size}(L_-) \geq 3) \wedge (\text{Size}(L_+) \geq 3)\ )$
   Dann repräsentiert L_+ die Knotenordnung von P_+ und L_- die Knotenordnung von P_-, welche beide positiv zur Ebene H_P orientiert sind. (s. Definition 1.8 S. 13)

3. $P \subseteq H_-^\circ \Leftrightarrow \text{Size}(L_+) \leq 2$ (kein Split)

Zweite Möglichkeit eines Polygonsplits

Auch dieser Algorithmus benötigt zur Berechnung des Splits durch H an P eine Laufzeit von O(n). Er ist entnommen aus [Meh84] Band 3 Seite 84. Hat man jedoch m Ebenen $H_1, ..., H_m$, und man interessiert sich lediglich für die Schnittpunkte dieser Ebenen mit P, so läßt sich diese Berechnung durchführen in Zeit :

$$O(n + m \cdot log(n))$$

Wir berechnen dazu zunächst die hierarchisch balancierte Darstellung $P_0, ..., P_k$ von P in Zeit O(n), welche als Grundlage des Verfahrens dient. Betrachte dazu Abbildung 2.4 Seite 32. Dabei sind $P_0, ..., P_k$ konvexe Polygone mit $P_k = P$ und P_{i-1} geht (grob gesehen) aus P_i dadurch hervor, daß aus P_i ein über den anderen Knoten gestrichen wird. $(1 \leq i \leq k)$. P_0 hat mindestens 3, höchstens jedoch 5 Knoten.
Da sich die Anzahl der Knoten von P_i nach P_{i-1} halbiert, gilt:

$$k \approx log(n)$$

und man benötigt zur Berechnung von $P_0, ..., P_k$ eine Laufzeit von:

$$\sum_{i=0}^{k} \frac{n}{2^i} \leq 2 \cdot n = O(n)$$

In gleicher Zeit läßt sich diesbezüglich ein 2-4 Baum konstruieren. Die Schnittpunkte einer Ebenen H mit P berechnen sich dann wie folgt. Wir bestimmen zunächst den Schnitt von H mit P_0.

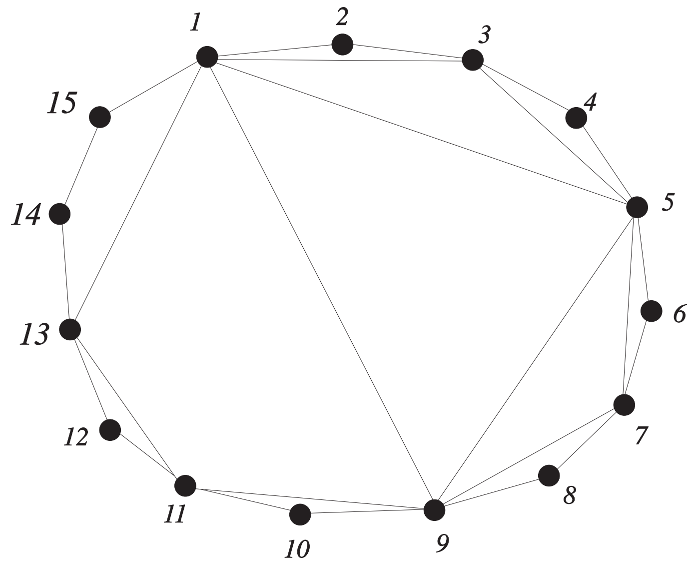

Abbildung 2.4: Hierarchisch balancierte Darstellung zu einem Polygon mit 15 Ecken

1.Fall H schneidet P_0 nicht.
Wir bestimmen die nächstgelegenen Ecken $d(P_0)$ zu H. (höchstens 2 Ecken).

1. $d(P_0) = \{\text{v}\}$
   Nur solche Kanten von P_1, welche die Kanten mit dem Knoten v aus P_0 ersetzen, können in P_1 durch die Ebene H geschnitten werden.

2. $d(P_0) = \{v_1, v_2\}$
   Nur solche Kanten von P, welche die Kante $(v_2 - v_1)$ aus P_0 ersetzen, können in P_1 durch die Ebene H geschnitten werden.

Betrachte also nur diese Kanten aus P_1 usw.

2.Fall H schneidet P_0.
Betrachte nur die Kanten in P_1, die geschnittene Kanten ersetzen. Wir machen das gleiche für $P_1, ..., P_k$.

Laufzeit bei gegebener hierarchischer Darstellung
In jedem Schritt werden nur konstant viele Kanten betrachtet. Maximal k solcher Schritte konstanter Laufzeit sind für die Berechnung nötig, und wir erhalten eine Laufzeit von:

$$O(log(n))$$

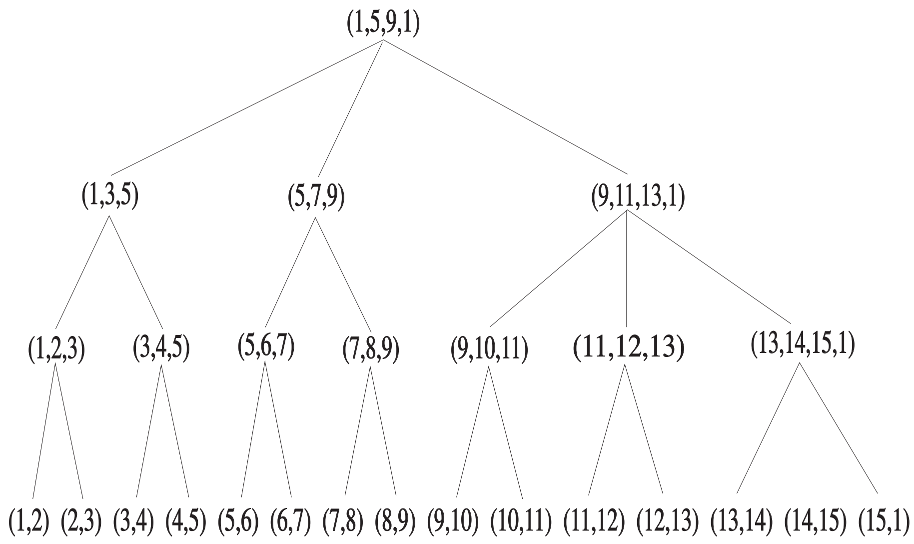

Abbildung 2.5: 2-4 Baum zur hierarchisch balancierten Darstellung eines Polygones mit 15 Ecken

2.5.2 Schnitt zweier konvexer Hyperpolyeder

Seien P, $Q \subseteq \mathbb{R}^d$ zwei konvexe Hyperpolyeder ($d \in \{2, 3\}$). Beide Hyperpolyeder sind in dem ihrer Dimension gemäßen Datenmodell gegeben. Dabei habe P die Komplexität n und Q die Komplexität m, wobei gilt $m \leq n$. Wir interessieren uns für:

$$R = P \cap Q$$

Es werden zwei Verfahren vorgestellt. Das erste ist ein für konvexe Hyperpolyeder allgemeingültiges Verfahren und berechnet den Schnitt in m aufeinanderfolgenden Splitberechnungen. Das zweite Verfahren ist speziell für Polygone vorgesehen und berechnet deren Schnitt in Zeit O(n).

Das allgemeine Verfahren zum Schnitt zweier konvexer Hyperpolyeder

Ist R wieder ein konvexes Hyperpolyeder der Dimension d , so soll das folgende Schnittverfahren $Cut(P_1, P_2)$ eine Repräsentation dieses Polyeders liefern, andernfalls NULL. Q sei konstruiert aus den orientierten Hyperebenen $\mathcal{H} = \{H_1, ..., H_m\}$. Diese Hyperebenen lassen sich gemäß dem Beweis von Satz 6 (S. 11) für Polygone bzw. dem Satz 7 (S. 14) für Polyeder in m Schritten konstanter Laufzeit herleiten. Der Schnitt zwischen P und Q berechnet sich dann aus dem Schnitt von P mit den geschlossenen positiven Halbräumen der Ebenen $\{H_1, ..., H_m\}$.

$$P \cap Q = P \cap \left(\bigcap_{i=1}^{m} H^{\circ}_{i_+} \right)$$

Bei dem Hyperpolyeder P beginnend, wird nun durch sukzessives Splitten durch die Ebenen $\{H_1, ..., H_m\}$ der Schnitt $R = P \cap Q$ wie folgt berechnet.

```
Cut(P, Q) {
    Berechne die Schnittebenen {H_1, ..., H_m}
    und speichere diese in einer Liste L_H;

    P_+ := P;                              // Splitpolygon

    forall(H in L_H) do {
        P_+ := Split(P_+,H);
        // d.h. belege P_+ mit (P_+ ∩ H°_+)
        // belege P_+ mit NULL, falls P_+ ∩ H_+ = ∅
        if(P_+ == NULL)     verlasse Schleife;
    }
    return P_+
}
```

Ist nach der Ausführung der Operation $Cut(P, Q)$ $P_+ = \text{NULL}$, dann gilt $P \cap Q = \emptyset$. Andernfalls entspricht P_+ einer Repräsentation des konvexen Hyperpolyeders $R = P \cap Q$. Der Splitalgorithmus für das konvexe Polyeder wird in Kapitel 3 ausführlich besprochen.

Schnitt zweier Polygone P und Q

Seien P, $Q \subseteq \mathbb{R}^2$ zwei konvexe Polygone, gegeben durch die Knotenordnungen $VO_P = (p_1, ..., p_n)$ bzw. $VO_Q = (q_1, ..., q_m)$ mit $m \leq n$, dann kann $P \cap Q$ in Zeit O(n) berechnet werden. Dieser Algorithmus ist entnommen aus [Meh84] Band 3 Seite 93. Sei p ein Punkt im Innern von P. Beispielsweise läßt sich p als Schwerpunkt der Knoten von P wie folgt in O(n) ermitteln.

$$\vec{p} = \frac{\sum_{i=1}^{n} \vec{p_i}}{n} \qquad (komponentenweise Addition)$$

Von p als Ursprung ausgehend, ziehen wir n Strahlen $Str_1, ..., Str_i, ..., Str_n$ derart, daß Strahl Str_i durch den i-ten Knoten p_i von P führt. $(1 \leq i \leq n)$ Dadurch wird der Raum (Ebene) $\mathbb{R}^2$ in n Sektoren $S_1, ..., S_n$ unterteilt. Für Knoten q_i von Q sei $S(q_i)$ derjenige Sektor von P, in welchen sich q_i befindet. Zunächst berechnen wir in Zeit O(n) denjenigen Sektor von P, in welchem sich Knoten q_1 befindet. Dazu testen wir

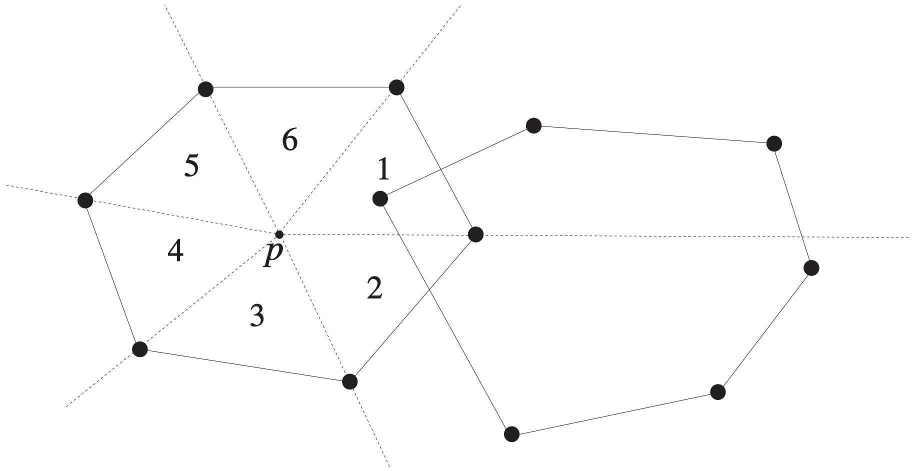

Abbildung 2.6: Schnitt zweier Polygone

jeden Sektor in Zeit O(1), ob sich q_i darin befindet. Als nächstes betrachten wir die Kante $e_1 = (q_1, q_2)$ von Q. Sei s_1 die Anzahl derjenigen Strahlen, welche diese Kante schneiden, dann können wir in Zeit $O(1 + s_1)$ alle Schnitte dieser Kante e_1 mit P berechnen. Dies folgt aus folgender Tatsache. Sei $e_{S_{q_1}}$ die Kante von P, welche durch den Sektor $S(q_1)$ verläuft. Wir können in Zeit $O(1)$ entscheiden, ob die Kante e_1 die Kante $e_{S_{q_1}}$ schneidet. Weiter können wir durch Schnitt der Kante e_1 mit den beiden den Sektor $S(q_1)$ bgrenzenden Strahlen entscheiden, ob die Kante e_1 den Sektor $S(q_1)$ verläßt und falls ja, in welchen der beiden angrenzenden Sektoren. Finden wir auf diese Weise heraus, daß ($e_1 \subseteq S(q_1)$) ist, so sind wir in *O(1)* fertig. Andernfalls führen wir unsere Analyse $e_1 \cap P$ im Nachbarsektor analog fort. In gleicher Weise berechnen wir alle Schnitte der Kante $e_2 = (q_2, q_3)$ mit den Kanten von P in Zeit $O(1 + s_2)$, wobei s_2 wiederum die Anzahl der Strahlen ist, welche die Kante e_2 schneiden. Fahren wir auf diese Weise mit unserer Berechnung fort, so können wir alle Schnittpunkte der Kanten von P in der Reihenfolge $q_1, ..., q_m$ in Zeit $O(m + \sum_{i=1}^{m} s_i)$ berechnen. Da ein Strahl höchstens 2 Kanten von Q schneiden kann, gilt $\sum_{i=1}^{m} s_i \leq 2 \cdot n$. Damit gilt:

$$O(m + \sum_{i=1}^{m} s_i) \leq O(n)$$

Nun müssen wir noch folgende Fälle unterscheiden:

1.Fall $P \cap Q$ ist konvexes Polygon und $P \nsubseteq Q$ und $Q \nsubseteq P$

2.Fall $Q \subseteq P$

3.Fall $P \subset Q$

4.Fall $P \cap Q = \emptyset$

Wir betrachten diese Fälle nun nacheinander.

1. Dieser erste Fall tritt genau dann ein, wenn es zwischen P und Q mindestens einen Schnittpunkt gibt. Um folgendes durchzuführen achten wir darauf, daß P und Q gleichorientiert sind. (s. Satz 6 bzw. Def. 1.8) In obigem Algorithmus durchlaufen wir die Knoten $(q_1, ..., q_m)$ von Q in ihrer Ordnung und erhalten in derselben Ordnung die Schnittpunkte $(s_1, ..., s_k)$ zwischen den beiden Polygonen P und Q. In Gedanken mischen wir diese Schnittknoten mit den Knoten von Q bezüglich der vorgegebenen Ordnung und erhalten eine Knotenordnung VO_Q^*, welche das Polygon Q ebenfalls repräsentiert. Wir bezeichnen $s \in \{s_1, ..., s_k\}$ als *Eingangsschnittknoten*, wenn es einen Knoten $q \in \{q_1, ..., q_m\}$ gibt , welcher bezüglich der Ordnung VO_Q^* Vorgängerknoten von s ist, und welcher nicht in P liegt. ($q = pred(s)$ und $q \notin S(q)$) Andernfalls bezeichnen wir s als Ausgangsknoten. Eingangs- und Ausgangsknoten alternieren auf $(s_1, ..., s_k)$ und sind bereits Knoten aus der gesuchten Knotenordnung VO_R von $R = Q \cap P$. Die Knotenordnung $(s_1, ..., s_k)$ muß möglicherweise noch aufgefüllt werden mit Knoten aus P und Q, damit sie das Polygon R vollständig repräsentiert. Dabei gilt, daß zwischen Eingangs- und Ausgangsknoten nur Knoten aus Q und zwischen Ausgangs- und Eingangsknoten nur Knoten von P in vorgegebener Ordnung liegen können. Somit erhalten wir die Knotenordnung VO_R von R, indem wir diese Zwischenlücken entsprechend ausfüllen. Dazu merken wir uns in obigem Algorithmus für jeden Schnittpunkt s Kanten von P und Q, auf welchen dieser Punkt s liegt. Nach Beendigung des obigen Algorithmus durchlaufen wir die Schnittknotenliste $(s_1, ..., s_k)$ und füllen, mit Hilfe der gemerkten Kanteninformation je Schnittknoten, die Zwischenlücken in Zeit Komplexität von R aus.

2. $Q \subseteq P \Leftrightarrow q_i \in (S(q_i) \cap P) \,\forall\, (1 \leq i \leq m)$

   Dies läßt sich leicht in Zeit O(m) testen.

3. $P \subset Q \Leftrightarrow p \in Q$ und $q_i \in (S(q_i) \setminus P) \,\forall\, (1 \leq i \leq m)$

   Dies läßt sich leicht in Zeit O(m) testen.

4. $P \cap Q = \emptyset$, falls nicht Fall1 und nicht Fall2 und nicht Fall3.

Betrachtet man Polygonschnitte im $\mathbb{R}^3$, so verwendet man anstelle der Strahlen Halbebenen, welche orthogonal zur Ebene sind, in welcher P liegt. Weiterhin kann dann $P \cap Q$ nur dann ein konvexes Polygon sein, wenn die Normalen der Polygone P und Q kollinear zueinander sind.

Kapitel 3

Splitalgorithmus

Wir kommen nun zu einem wesentlichen Teil meiner Arbeit — der Implementation der Basisoperation Split-Polyeder. Ein konvexes Polyeder P der Komplexität n (s.S. 29) sei im Hauptspeicher des Rechners in Form des internen Datenmodells gegeben. Weiter sei eine orientierte Ebene H durch die Normale $\vec{m}$ und das Skalar c wie folgt gegeben.

$$H := \{ \quad \vec{x} \in \mathbb{R}^d \quad : \quad < \vec{m}, \vec{x} >= c \quad \}$$

Mit dem nun folgenen Algorithmus sind wir in der Lage, den Split (s.S.22) des Polyeders P durch die orientierte Schnittebene H in Zeit $O(n)$ zu berechnen. Dabei erhalten wir als Ergebnis die beiden Polyeder $P_+ = (P \cap H_+^\circ)$ und $P_- = (P \cap H_-^\circ)$ in Form des internen Datenmodells. Ihr Rand wird beschrieben durch die beiden Formeln:

$$B_{P_+} = (B_P \cap H_+^\circ) \cup f_{H_+} \qquad B_{P_-} = (B_P \cap H_-^\circ) \cup f_{H_-}$$

f_{H_+}, f_{H_-} sind die beiden entgegengesetzt gerichteten Polygonflächen, in welchen P sich mit H schneidet. (s. Abschnitt 1.4.3 S. 22)

Die Einfachheit des nachfolgenden Split-Algorithmus soll nicht über die Schwierigkeit der Neuorganisation der komplexen Nachbarschaftsbeziehungen — der Knoten, Kanten und Flächen — in den Polyedern P_+ und P_- hinwegtäuschen. Er ist entstanden, sowohl aus theoretischen Überlegungen, als auch aus Überlegungen, welche sich erst während der Implementierung der Splitoperation ergeben haben. Schließlich kommt es bei einem solchen Algorithmus nicht nur auf Zeit- und Platzeffizienz an. Es ist ebenso wichtig, bei der Entwicklung eines Algorithmus darauf zu achten, daß er klar und übersichtlich ist, um so gravierende Fehler bei der Implementierung zu vermeiden. Wir gliedern den Splitalgorithmus in drei Teilverfahren.

Wie wir in den nachfolgenden Kapiteln noch erfahren werden, benötigen wir das Polyeder P auch noch nach Durchführung des Splitalgorithmus. Dies ist der Grund, warum wir uns zunächst eine Kopie der Repräsentation von P erzeugen. Dazu sind zwei Objektdurchläufe nötig. Im ersten wird neuer Speicher für die zu kopierenden Objekte erzeugt. Gleichzeitig wird für jedes Objekt (Knoten, Kante, Fläche) der alte mit dem neuen Speicherplätz verpointert. Im zweiten Objektdurchlauf können dann die Nachbarschaftsbeziehungen dieser Objekte einfach in die neue Repräsentation durchpropagiert werden. Desweiteren werden wir nach Beendigung dieses ersten Teilverfahrens dazu in der Lage sein, darüber zu entscheiden, ob das Polyeder P durch die Ebene H gesplittet wird.

Das zweite Teilverfahren dient dazu, die kopierte Repräsentation von P derart zu modifizieren, daß der Rand B_P von P nur noch von solchen Begrenzungsflächen beschrieben wird, welche entweder vollständig in H_+° oder vollständig in H_-° liegen. Damit schaffen wir uns eine günstigere Ausgangsposition für die Durchführung des Splits. Wir haben somit die Fallunterscheidungen, womit wir in der urspünglichen Repräsentation hätten rechnen müssen, auf ein Minimum reduziert. Denn betrachtet man sich nun noch einmal die Formel $B_{P_+} = (B_P \cap H_+^\circ) \cup f_{H_+}$, so erkennen wir, daß der erste Teil der Formel nun gerade durch die positiven Randflächen unserer neuen Repräsentation von P abgedeckt wird. Die Nachbarschaftsbeziehungen können wir nun problemlos aus dieser neuen Repräsentation von P für die Beschreibung des Teilrandes $(B_P \cap H_+^\circ)$ von P_+ verwenden.

Wir müssen uns nun nur noch um die Anbindung des Schnittpolygones f_{H_+} an diese Teilrepräsentation von P_+ kümmern, was wir im dritten Teilverfahren erledigen. Hierbei wird insbesondere die Nützlichkeit des Aufbaues des internen Datenmodells deutlich. Eine ähnliche Vorgehensweise — wie in diesem dritten Teilverfahren unseres Splitalgorithmus — beschreibt Mulmuley in [Mul94] in einem randomisierten Verfahren zur Berechnung des Schnitts von m positiven Halbräumen in Zeit $O(n \cdot log(n))$.

3.1 Kopie

3.1.1 Erzeugung

Wie oben bereits erwähnt, werden alle Knoten, Kanten und Flächen des Polyeders P durchlaufen. Dabei erzeugt man für jeden Knoten, jede Kante und jede Fläche Speicherplatz für einen neuen Knoten ,eine neue Kante bzw. eine neue Fläche. Analog zu den alten Knoten, Kanten und Flächen speichert man die Adressen der neuen Objekte je in einer neuen Knotenliste L_v, Kantenliste L_e und Flächenliste L_f. Zusätzlich erzeugt man für jede(n) alte(n) und jede(n) neue(n) Knoten, Kante und Fläche Speicher für eine Hilfs(split)infostruktur, worauf das zugehörige Objekt direkten Zugriff hat. Zunächst sollen mit Hilfe dieser Struktur die alten mit ihren neuen äquivalenten Objekten gegenseitig verbunden (verpointert) werden.

Sei v der alte Knoten von P, v^* der zugehörige neue Knoten, dann vereinbare man: $v = \text{old}(v^*)$; $v^* = \text{new(v)}$
Sei e die alte Kante von P, e^* die zugehörige neue Kante, dann vereinbare man: $e = \text{old}(e^*)$; $e^* = \text{new(e)}$
Sei f die alte Fläche von P, f^* die zugehörige neue Fläche, dann vereinbare man: $f = \text{old}(f^*)$; $f^* = \text{new(f)}$

Im noch gleichen Objektdurchlauf wird für jeden neuen Knoten v zusätzlich folgende Operation durchgeführt:

1. Kopiere die Koordinaten des alten Knotens old(v) in den neuen Knoten v.

2. Bestimme die Lage von v bzgl. der Schnittebene H.

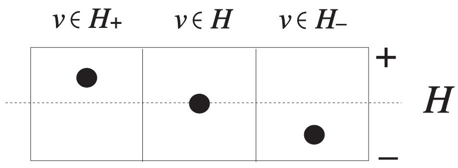

Abbildung 3.1: Lagebeziehung der Knoten bzgl. der Splitebene H

Lege die Lageinformation in der Variablen *Lage* der Hilfsstruktur von v ab. Die Information bestimmt man aus dem Standardskalarprodukt zwischen $\vec{v}$ (Ortsvektor des Knotens v) und dem Normalenvektor $\vec{m}$ der Schnittebene H in konstanter Zeit wie folgt.

$$\begin{array}{lrlclcll} Lage(v) = & + & 1 & \Leftrightarrow & v \in & H_+ & \Leftrightarrow & <\vec{m},\vec{v}> \;\; > \;\; c \\ Lage(v) = & & 0 & \Leftrightarrow & v \in & H & \Leftrightarrow & <\vec{m},\vec{v}> \;\; = \;\; c \\ Lage(v) = & - & 1 & \Leftrightarrow & v \in & H_- & \Leftrightarrow & <\vec{m},\vec{v}> \;\; < \;\; c \end{array}$$

Zusammenfassung

- In einem einmaligen Durchlauf durch alle Knoten, Kanten und Flächen von P (Laufzeit O(n)), werden diese, analog zur alten Datenbank von P, in der Knoten-, Kanten- und Flächenliste (L_v, L_e, L_f) abgelegt.

- Jedes alte Objekt hat zu seinem neuen Objekt direkten Zugriff und umgekehrt.

- Die Koordinaten der alten Knoten wurden in die neuen Knoten kopiert, und die Lage von jedem Knoten bzgl. H wurde bestimmt.

Splitentscheidung
Aufgrund der Lagebestimmung der Knoten läßt sich nach obigem Objektdurchlauf nach Satz 9, Seite 19 wie folgt unmittelbar entscheiden, ob P durch H gesplittet wird oder nicht.

- Falls gilt: $(\exists\, v_1 \in L_v \therefore \text{Lage}(v_1) = +1 \quad \wedge \quad \exists\, v_2 \in L_v \therefore \text{Lage}(v_2) = -1)$
  Dann wird P durch H in P_+ und P_- gesplittet. $(P/H = \{P_+, P_-\})$
- Andernfalls wird P durch H nicht gesplittet. Das Splitverfahren wird abgebrochen.

Wird P durch H gesplittet, so können wir wie folgt fortfahren. Zwar sind nun alle Objekte der alten Datenbank von P dupliziert und mit den Koordinaten der Knoten auch die **geometrische** Information kopiert. Aber die **topologische** Information d.h. die Verbindungen der Oberflächenobjekte untereinander, wodurch der Rand des Polyeders beschrieben wird, ist in der neuen Repräsentation von P noch nicht erzeugt. Dies geschieht in einem weiteren Objektdurchlauf.

3.1.2 Verbindung

Mit Hilfe des gegenseitigen Zugriffs der alten und neuen Objekte des Polyeders P lassen sich die in der neuen Datenbank von P noch fehlenden Verbindungen leicht wie folgt von der alten in die neue Datenbank durchpropagieren. Dabei werden (wie oben) alle Objekte (Knoten, Kanten und Flächen) von P erneut (in n Schritten) besucht.

Knoten

$\forall\, v$ der alten Knotenliste von P do {

$$new(v) \rightarrow edge = new(v \rightarrow edge);$$

}

Kanten

$\forall\, e$ der alten Kantenliste von P do {

$$new(e) \rightarrow v_1 = new(e \rightarrow v_1);$$
$$new(e) \rightarrow v_2 = new(e \rightarrow v_2);$$

$new(e) \rightarrow pred = new(e \rightarrow pred);$
$new(e) \rightarrow succ = new(e \rightarrow succ);$
$new(e) \rightarrow co = new(e \rightarrow co);$

$new(e) \rightarrow f = new(e \rightarrow f);$

}

Flächen

$\forall\, f$ der alten Flächenliste von P do {

$new(f) \rightarrow edge = new(f \rightarrow edge);$
$new(f) \rightarrow size = f \rightarrow size;$

}

3.1.3 Zusammenfassung

Damit ist die alte Datenbank von P komplett in die 3 neuen Objektlisten L_v, L_e, L_f kopiert. Dazu waren 2 Objektdurchläufe nötig – also 2mal n Schritte. Damit benötigt man für eine komplette Kopie eines Polyeders P eine Laufzeit von O(n). (n : Anzahl der Kanten in P)

3.2 Modifikation der Repräsentation von P

Nach Abschnitt 1.4.3 werde P repräsentiert durch die Knotenordnungen aus der Menge:

$$P: \quad PGN_+ \cup PGN \cup PGN_- \tag{3.1}$$

Nach Satz 10 (S. 22) kann P auch durch die Knotenordnungen der Menge:

$$P: \quad (\underbrace{PGN_+ \cup PGN_+^*}_{B_P \cap H_+^\circ}) \cup (PGN_-^* \cup PGN_-) \tag{3.2}$$

repräsentiert werden, wobei die Flächen, welche durch die Knotenordnungen der Mengen PGN_+^* und PGN_-^* repräsentiert werden, durch Split aus den Flächen hervorgehen, welche durch die Knotenordnungen der Menge PGN repräsentiert werden. Wir werden nun die vorliegende Repräsentation (3.1) von P in die Repräsentation (3.2) umwandeln.

Wir benötigen dazu zwei Flächendurchläufe wie folgt.

erster Flächendurchlauf

```
Für alle Flächen f ∈ L_f do {
    Für alle Kanten e ∈ f  (in Kantenreihenfolge)  do {
    OP_1(e, f);
    }
}
```

zweiter Flächendurchlauf

```
Für alle Flächen f ∈ L_f do {
    OP_2.1(f);
    OP_2.2(f);
}
```

Die Operationen OP_1, $OP_{2.1}$ und $OP_{2.2}$ werden im folgenden genauer betrachtet. Zunächst jedoch nehmen wir die Kanten von P näher unter die Lupe.

3.2.1 Kantenanalyse

Sei e eine Kante des Polyeders P, so ergeben sich für e folgende 4 mögliche Lagebeziehungen bzgl. der Schnittebene H.

1. **e und H schneiden sich nicht**
   - 1.1. $e \in H_+$
   - 1.2. $e \in H_-$
2. **e und H berühren sich in einem Knoten von** e
   - 2.1. $e \in H_+^\circ$
   - 2.2. $e \in H_-^\circ$

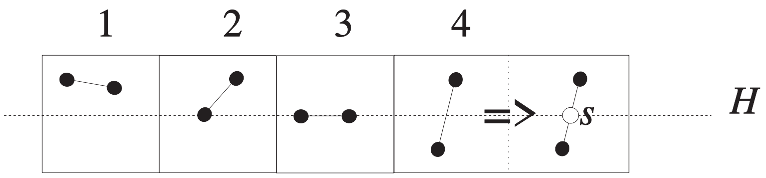

Abbildung 3.2: Lagebeziehung der Kanten bzgl. der Splitebene H

3. **e liegt in der Ebene** H

   3.1. e ist Kante eines Polygones, welches in H_+° liegt

   3.2. e ist Kante eines Polygones, welches in H_-° liegt

4. **e wird durch H gesplittet**
   in eine Kante e_+ und eine Kante e_-. Dabei schneiden sich e und H in einem Punkt $s \in H$, wobei s zunächst noch kein Knoten von P ist.

Da die Lage der Knoten von P bzgl. H bereits ermittelt ist, und jede Kante direkten Zugriff zu ihren Knoten hat, läßt sich auch die Lage von e leicht wie folgt ermitteln.

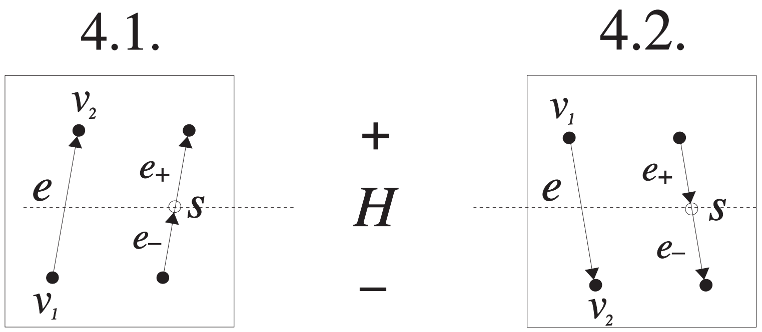

Abbildung 3.3: Split einer Kante

3.2.1.1 Kantensplit

Sei e eine Kante vom Kantenfall 4.1 mit $e \rightarrow v_1 \in H_-$ und $e \rightarrow v_2 \in H_+$. Dann schneidet die Kante e die Ebene H in genau einem Punkt $\vec{s}$ und e wird gesplittet in die Kanten e_+ und e_-.

$$e_+ = (s, v_2); \quad e_- = (v_1, s)$$

Lage(v_1)	**Lage**(v_2)	**Kantenfall(e)**	**Lage**(e)
+1	+1	1.1.	$(e \subseteq H_+)$
−1	−1	1.2.	$(e \subseteq H_-)$
1	0	2.1.1	$(e \subseteq H_+^\circ) \wedge (\lvert e \cap H \rvert = 1)$
0	1	2.1.2	$(e \subseteq H_+^\circ) \wedge (\lvert e \cap H \rvert = 1)$
−1	0	2.2.1	$(e \subseteq H_-^\circ) \wedge (\lvert e \cap H \rvert = 1)$
0	−1	2.2.2	$(e \subseteq H_-^\circ) \wedge (\lvert e \cap H \rvert = 1)$
0	0	3	$(e \subseteq H)$
−1	1	4.1	$(e \cap H_+ \neq \emptyset) \wedge (e \cap H_- \neq \emptyset)$
1	−1	4.2	

Tabelle 3.1:

Der Schnittpunkt $\vec{s} = e \cap H$ berechnet sich dann aus folgender Formel, deren Herleitung man auf Seite 20 nachlesen kann. Wir definieren dabei $\vec{v} := (\vec{v_2} - \vec{v_1})$.

$$\vec{s} \;=\; \vec{v}_1 + \frac{c- < \vec{m}, \vec{v}_1 >}{< \vec{m}, \vec{v} >} \cdot \vec{v}$$

Wäre $< \vec{m}, \vec{v} >= 0$, dann wäre $\vec{m}$ orthogonal zu $(\vec{v_2} - \vec{v_1})$ und e läge parallel zur Ebene H, was der Ausgangssituation widerspricht. Sei e eine Kante vom Kantenfall 4.2 so wird diese analog behandelt.

3.2.2 Auswirkungen eines Kantensplits auf das Datenmodell

Sei f ein Polygon aus der derzeit das Polyeder P repräsentierenden Randbeschreibung. f ist beschrieben durch die Kantenordnung $EO = (e_1, ..., e_m)$. $\overline{e}_1$ ist die zu e_1 korrespondierende Kante und ist Kante der zu f benachbarten Fläche $\overline{f}$. e_1 werde gesplittet in die beiden Teilkanten e_{1_+} und e_{1_-}. Nach Satz 7 (S. 13) wird f dann ebenfalls eindeutig beschrieben durch die modifizierte Kantenordnung $EO^* = (e_{1_+}, e_{1_-}, ...e_m)$. Da $\overline{e}_1$ nicht gleichzeitig zu den beiden Kanten e_{1_+} und e_{1_-} korrespondieren kann, wird diese Kante analog zu e_1 in $\overline{e}_{1_+}$ und $\overline{e}_{1_-}$ gesplittet. Entsprechend ändert sich damit auch die Kantenordnung von $\overline{f}$. Nun ist offensichtlich, daß e_{1_+} mit $\overline{e}_{1_+}$ korrespondiert und daß e_{1_-} mit $\overline{e}_{1_-}$ korrespondiert.(s.Abbildung 3.4 wobei $e = e_1$ ...) Damit repräsentiert die derart modifizierte Randbeschreibung von P ebenfalls das Polyeder.

3.2.3 Kantensplit durch Operation OP_1

Im ersten Flächendurchlauf wird für jede Fläche jede Kante (in Kantenreihenfolge) mit Operation OP_1 bearbeitet. Wir stellen nun eine vorläufige Version für OP_1 vor. Sei e

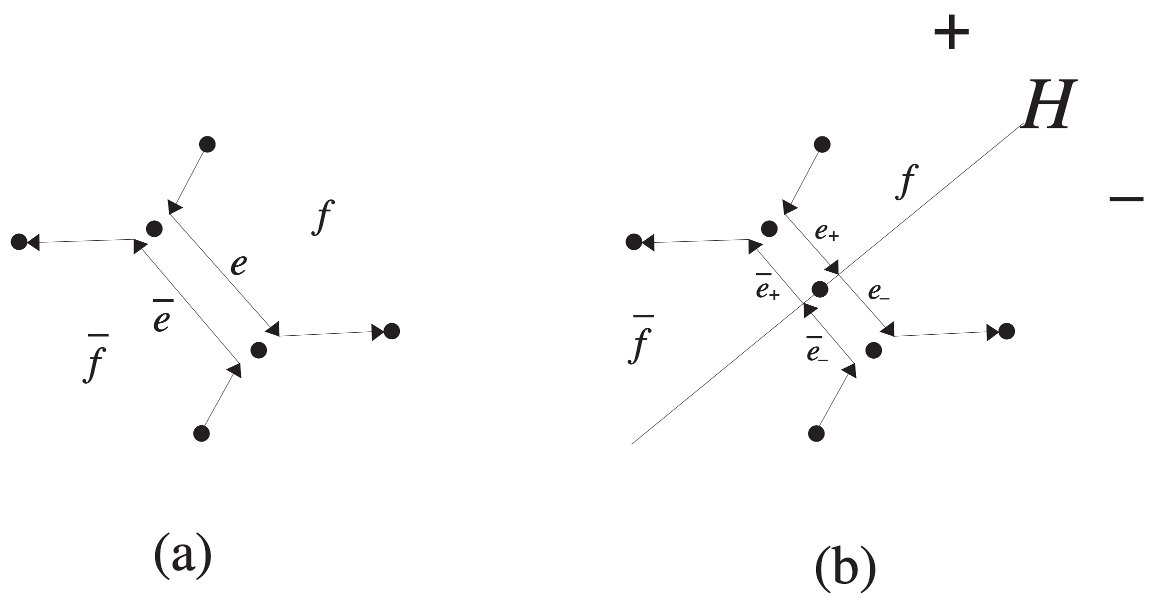

Abbildung 3.4: Auswirkungen eines Kantensplits auf das Datenmodell

die aktuell in Operation OP_1 zu bearbeitende Kante. Wird e durch H in e_+ und e_- gesplittet, so ersetzt man die Kante e durch die Kanten e_+ und e_- in der Kantenordnung der Fläche $e \rightarrow f$. Desweiteren baut man eine Vater - Sohn - Beziehung zwischen e und e_+, e_- auf. (s. Abb. 3.5)

$\mathbf{OP_1(e)}$ {

if(e eine Kante, welche durch H gesplittet wird) {

1. Berechne den Schnittpunkt, in welchem H die Kante e schneidet;
2. Erzeuge die beiden durch den Schnitt entstehenden Kanten e_+ und e_-
3. Erzeuge den Schnittknoten s und belege diesen mit den Schittkoordinaten.
4. Ersetze e durch e_+ und e_- in der Kantenliste L_e, ohne den Speicher der Kante e freizugeben und hänge s in die Knotenliste L_v.
5. if($e \rightarrow v_1 \in H_+$){ (siehe Abbildungen 3.3 und 3.4)
   (a) Verbinde e_+ mit der Vorgängerkante von e;
   (b) Verbinde e_+ mit e_- ;
   (c) Verbinde e_- mit der Nachfolgerkante von e;
   }else{ ($e \rightarrow v_1 \in H_-$)
   (analog)
6. //Vater - Sohn - Beziehung

$plus\_son(e) = e_+;$
$minus\_son(e) = e_-;$
$father(e_+) = e;$
$father(e_-) = e;$

(siehe Abbildung 3.5)

}

}

Da die Kantensplits nicht gleichzeitig für alle Kanten vom Kantenfall 4 durchgeführt werden, können die zu e_+ bzw. e_- korrespondierenden Kanten noch nicht miteinander verbunden (verpointert) werden. Indirekt jedoch existiert diese Verbindung über die Vater-Sohn-Beziehung wie folgt:
(siehe auch Abb. 3.5)

$$\mathrm{co}(e_+) := plus\_son(\overbrace{co(\overbrace{father(e_+)}^{e})}^{\bar{e}});$$
$$\mathrm{co}(e_-) := minus\_son(\underbrace{co(\underbrace{father(e_-)}_{e})}_{\bar{e}});$$

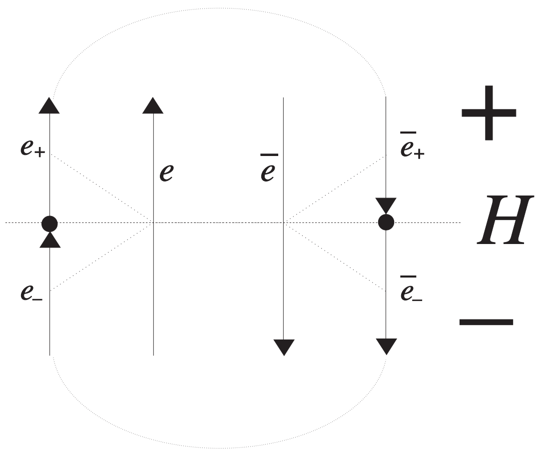

Abbildung 3.5: Verpointerung korrespondierender Kanten bei einem Kantensplit

Nach Abschluß des ersten Flächendurchlaufs können wir folgendes festhalten:

- Die Objektlisten L_v, L_e, L_f repräsentieren (indirekt) den Polyeder P gemäß dem internen Datenmodell.

- Die Kantenliste L_e enthält keine Kanten mehr, welche durch H gesplittet werden.
- Sei f eine Fläche aus der Flächenliste L_f, welche durch H gesplittet wird. Dann existieren 2 Knoten $x_1 \in H$ und $x_2 \in H$ von f, in welchen der Rand von f durch die Schnittebene H gesplittet wird. (D.h. es gibt nur noch Flächen $f \in L_f$ vom Flächenfall 4 der Abbildung 1.10 (S. 23) .)

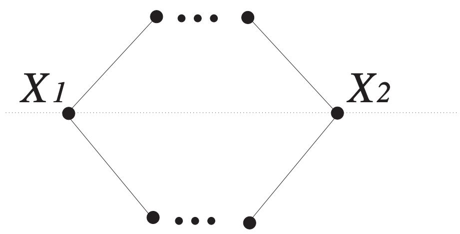

Abbildung 3.6: Vom Split betroffene Fläche nach dem ersten Flächendurchlauf

3.2.4 Auswirkungen eines speziellen Polygonsplits auf das Datenmodell

Sei $f \in L_f$ eine Fläche, welche durch H in den Knoten x_1 und x_2 gesplittet wird. Dann existieren für f 4 spezielle Kanten **in_plus**, **out_plus** , **in_minus**, **out_minus**, welche wie folgt eindeutig definiert sind.

Bezeichnung 12
Sei e eine Kante von f dann gilt:

$$\begin{array}{rclclcl}
\textbf{in_plus} & := & e & \Leftrightarrow & e \to v_1 \in H & \wedge & e \to v_2 \in H_+ & (i_+) \\
\textbf{out_plus} & := & e & \Leftrightarrow & e \to v_1 \in H_+ & \wedge & e \to v_2 \in H & (o_+) \\
\textbf{in_minus} & := & e & \Leftrightarrow & e \to v_1 \in H & \wedge & e \to v_2 \in H_- & (i_-) \\
\textbf{out_minus} & := & e & \Leftrightarrow & e \to v_1 \in H_- & \wedge & e \to v_2 \in H & (o_-)
\end{array}$$

Weiterhin definieren wir: $x_1 := out\_plus \to v_2 \quad x_2 := in\_plus \to v_1$

Nun splitten wir die Fläche f in die Flächen f_+ und f_- wie in Abbildung 3.7. Dabei entsteht für f_+ eine neue Kante $e_+ = (x_1, x_2)$ und für f_- eine neue Kante $e_- = (x_2, x_1)$. Dazu trennen wir die Verbindung zwischen den Kanten $\mathbf{o}_+$ und $\mathbf{i}_-$ auf. Dann verbinden wir $\mathbf{i}_-$ mit der neuen Kante e_- und wir verbinden $\mathbf{o}_+$ mit der neuen Kante e_+. Analog verarbeiten wir die Kanten $\mathbf{o}_-$ und $\mathbf{i}_-$. Schließlich verbinden wir die beiden Kanten e_+ und e_- korrespondierend miteinander. Nach Satz 10 (S. 22) kann die Fläche f nun durch die beiden Flächen f_+ und f_- in der Liste L_f ersetzt werden, und die neu gewonnene Repräsentation von P entspricht dem internen Datenmodell.

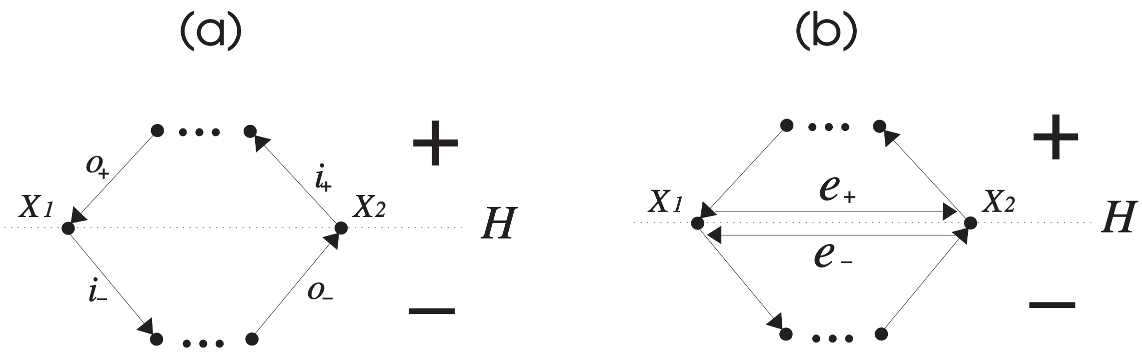

Abbildung 3.7: Split einer Polygonfläche

3.2.5 Polygonsplit durch Operation $OP_{2.1}$

Obiges Verfahren führen wir nun für alle Flächen $f \in L_f$, welche durch H gesplittet werden, in einem zweiten Flächendurchlauf mit der Operation $OP_{2.1}(f)$ durch. Damit die Operation $OP_{2.1}$ auch durchgeführt werden kann, wird die Operation $OP_1(e, f)$ im ersten Flächendurchlauf wie folgt ergänzt, wobei f die aktuelle Fläche ist und e die aktuelle Durchlaufkante der Fläche f. Die zur Fläche f zugehörigen Spezialkanten (i_+, o_+, i_-, o_-) sollen in der Hilfs(split)struktur von f vermerkt werden. Als Spezialkanten kommen dabei solche Kanten in Frage welche:

1. die Ebene H in einem Knoten berühren (Kantenfall 2 (s. S. 42 ff.))

2. aus dem Split einer Kante (vom Kantenfall 4 (s. S. 42 ff.)) hervorgehen und somit ebenfalls H in einem Knoten berühren.

Die Programmerweiterung zu OP_1 entnehme man Tabelle 3.2. Es ist offensichtlich, daß

Kantenfall(e)	**Lage(e)**	**Ergänzung in $OP_{1.1}$**
2.1.1	$(e \to v_1 \in H_+) \wedge (e \to v_2 \in H)$	$out\_plus(f) = e;$
2.1.2	$(e \to v_2 \in H_+) \wedge (e \to v_1 \in H)$	$in\_plus(f) = e;$
2.2.1	$(e \to v_1 \in H_-) \wedge (e \to v_2 \in H)$	$out\_minus(f) = e;$
2.2.2	$(e \to v_2 \in H_-) \wedge (e \to v_1 \in H)$	$in\_minus(f) = e;$
4.1	$(e \to v_1 \in H_-) \wedge (e \to v_2 \in H_+)$	$out\_minus(f) = e_-;$ $in\_plus(f) = e_+;$
4.2	$(e \to v_1 \in H_+) \wedge (e \to v_2 \in H_-)$	$out\_plus(f) = e_+;$ $in\_minus(f) = e_-$

Tabelle 3.2: Ergänzende Operationen zu OP_1

in den Flächen, welche durch H nicht gesplittet werden, höchstens 2 Spezialkanten vermerkt werden. (Siehe dazu auch Abbildung 3.8) In $OP_{2.1}$ kann man dann wie folgt verfahren.

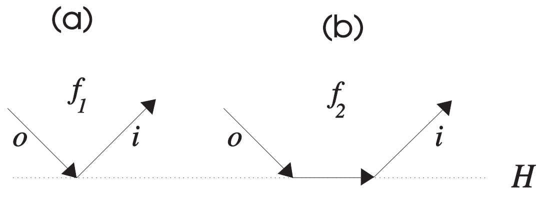

Abbildung 3.8: Flächen, welche nicht gesplittet werden, mit Spezialkanten

Sei f die aktuelle Fläche in $OP_{2.1}$. Sind alle 4 Kanten (o_+, i_-, o_-, i_+) in der Hilfs(split)struktur von f mit entsprechenden Adressen belegt (also keine Nullpointer), so handelt es sich bei f um eine gesplittete Fläche. f wird dann in f_+ und f_- gesplittet, wie oben beschrieben. f wird in der Flächenliste L_f durch f_+ und f_- ersetzt. Anschließend wird f gelöscht. Zusätzlich werden die speziellen Kanten, falls sie aus einer gesplitteten Kante heraus entstanden sind, mit ihren korrespondierenden Kanten (mit Hilfe der Vater-Sohn-Beziehung s. Abbildung 3.5 Seite 46) verbunden.

$OP_{2.1}(f)\{$

if(in_plus $\wedge$ in_minus $\wedge$ out_plus $\wedge$ out_minus) $\{$
// Flächensplitentscheidung

1. erzeuge e_+ und e_-;
2. hänge die Kanten e_+ und e_- je in die Kantenliste L_e;
3. // Knoten der neuen Kanten
   - $e_+ \rightarrow v_1 = out\_plus \rightarrow v_2$;
     $e_+ \rightarrow v_2 = in\_plus \rightarrow v_1$;
   - $e_- \rightarrow v_1 = out\_minus \rightarrow v_2$;
     $e_- \rightarrow v_2 = in\_minus \rightarrow v_1$;
4. // Verbindungsumlenkung
   - $out\_plus \rightarrow succ = e_+$;
     $e_+ \rightarrow pred = out\_plus$;
   - $in\_plus \rightarrow pred = e_+$;
     $e_+ \rightarrow succ = in\_plus$;
   - $out\_minus \rightarrow succ = e_-$;
     $e_- \rightarrow pred = out\_minus$;
   - $in\_minus \rightarrow pred = e_-$;
     $e_- \rightarrow succ = in\_minus$;

5. erzeuge f_+ und f_-;
6. ersetze in der Liste L_f die Fläche f durch die Flächen f_+ und f_-, ohne den Speicherplatz von f freizugeben;
7. // Kantenzuweisung $f_+ \to e = e_+$;
   $f_- \to e = e_-$;
8. //Flächenzuweisung
   $e_+ \to f = f_+$;
   $e_- \to f = f_-$;
9. //Lagezuweisung $f_+ \to mark = 1$; // d.h. $f_+ \subseteq H_+^\circ$
   $f_- \to mark = -1$; // d.h. $f_- \subseteq H_-^\circ$

- } $//end\_if$

} $//end\_OP_{2.1}$

Nach Abschluß der Operation $OP_{2.1}$ können wir folgendes festhalten:

- Der Polyeder P wird durch die modifizierten Listen L_v, L_e, L_f gemäß dem internen Datenmodell repräsentiert bis auf folgende Tatsache.

  Die Anzahl der Knoten der durch den Split neu entstandenen Flächen ist in diesen Flächen noch nicht eingetragen.

- Die Flächenliste L_f enthält nun keine Flächen mehr, welche durch H gesplittet werden.

3.2.6 Propagierung der Repräsentationsobjekte von P auf P_+ bzw. P_-

Die beiden Flächendurchläufe (s. S. 42) dienen *erstens* dazu, die Datenrepräsentation von P geeignet für einen Polyedersplit zu modifizieren, *zweitens* läßt dich für jedes Randobjekt (Knoten, Kante, Fläche) der neuen Repräsentation von P in konstanter Zeit entscheiden, ob es ein Randobjekt des Polyeders $P_+ = P \cap H_+^\circ$ oder des Polyeders $P_- = P \cap H_-^\circ$ ist, oder ob beides der Fall ist. Dazu stellen wir folgende Überlegungen an.

Da L_f keine Flächen mehr enthält, welche durch die Ebene H gesplittet werden, gilt folgendes Lemma.

Lemma 3
Sei $f \in L_f$ dann gilt:

$$\begin{array}{llllll} \textbf{entweder} & f \in H_+^\circ & \Rightarrow & f \subseteq (B_P \cap H_+^\circ) \subseteq B_{P_+} & \Rightarrow f \subseteq P_+ \\ \textbf{oder} & f \in H_-^\circ & \Rightarrow & f \subseteq (B_P \cap H_-^\circ) \subseteq B_{P_-} & \Rightarrow f \subseteq P_- \end{array}$$

Lemma 4
Sei $f \subseteq P_\pm$ dann gilt:

- *Sei e eine gerichtete Kante von f* $\Rightarrow \quad e \subseteq P_\pm$
- *Sei v ein Knoten von f* $\Rightarrow \quad v \subseteq P_\pm$

Feststellung 1

- *Nach Lemma 3 ist eine* **Fläche** *entweder dem Polyeder P_+ oder dem Polyeder P_- zuzuorden.*
- *Da eine* **Kante** *eindeutig einer Fläche zuzuordnen ist und diese eindeutig einem gesplitteten Polyeder, so ist auch die Kante entweder dem Polyeder P_+ oder dem Polyeder P_- zuzuordnen.*
- *Wie wir gleich in Lemma 5 sehen, gilt diese entweder-oder-Entscheidung nicht für* **Knoten***.*

Lemma 5
Sei v ein Knoten aus L_v dann gilt:

$$\begin{array}{llcl} 1.Fall: & v \in H_+ & \Rightarrow & v \in P_+ \wedge v \notin P_- \\ 2.Fall: & v \in H & \Rightarrow & v \in P_+ \wedge v \in P_- \\ 3.Fall: & v \in H_- & \Rightarrow & v \in P_+ \wedge v \notin P_+ \end{array}$$

Beweis zu Lemma 5

1.Fall: $v \in H_+$
Es gilt $v \in B_P \quad \Rightarrow \quad v \in (B_P \cap H_+^\circ) \subseteq P_+ \Rightarrow v \in P_+$
Angenommen $v \in (P_- \subseteq H_-^\circ)$

$\Rightarrow v \in H_-^\circ$
Dies jedoch **widerspricht** $v \in H_+$.

$\Rightarrow v \notin P_-$

2.Fall: $v \in H$
$\Rightarrow v \in H_+^\circ \wedge v \in H_-^\circ$
Weiterhin gilt: $v \in B_P$
$\Rightarrow \quad v \in (B_P \cap H_+^\circ) \subseteq P_+ \quad \wedge \quad v \in (B_P \cap H_-^\circ) \subseteq P_-$
$\Rightarrow v \in P_+ \wedge v \in P_-$
(v muß dann dupliziert werden in v_+ und v_- .)

3.Fall: $v \in H_-$
(Beweis analog zu 1.Fall)

∎

Lemma 6 (Knotenexistenz)
Sei $f \in L_f$ mit $f \subseteq P_\pm$, dann gilt: $\exists\, v \in f .\because. v \in H_\pm$

Beweis zu Lemma 6

Sei $f \in L_f$ mit $f \subseteq P_+ = (P \cap H_+^\circ)$. $\Rightarrow$ $f \subseteq H_+^\circ$

Annahme $f \subseteq H$
$\Rightarrow P \cap H = f \Rightarrow H$ ist redundant $\quad$ P wird also nicht gesplittet.

Wir haben einen **Widerspruch**.
$\Rightarrow \quad \exists\, x \in f \;\therefore\; x \in H_+ \quad (f \cap H_+ \neq \emptyset)$
$\Rightarrow \quad \exists\, v \in f$ *mit* $v \in L_v \;\therefore\; v \in H_+$ $\quad$ (da f konvexes Polygon)
Sei $f \in L_f$ mit $f \subseteq P_- = (P \cap H_-^\circ)$. $\quad$ (Beweis analog)

■

Lemma 7 (Flächenlageentscheidung)
Sei $v \in L_v$ mit $v \in H_\pm$, dann gilt $\forall f \in L_f$ mit $v \in f$: $f \in P_\pm$

Beweis zu Lemma 7

Sei $v \in H_+$ und f eine beliebige Fläche aus L_f mit $v \in f$.

Annahme $f \subseteq P_-$
$\Rightarrow \exists\, v^* \in L_v$ mit $v^* \in H_-$ $\quad$ (nach Lemma 6)

$\Rightarrow v^* \in P_-$ $\quad$ (nach Lemma 5)

Da f durch einen zusammenhängenden Graphen repräsentiert wird, existiert ein Pfad auf diesem Graphen von v nach v^*. Damit wäre f jedoch eine Fläche, welche durch H gesplittet wird — **Widerspruch**. Somit gilt: $\mathbf{f} \in \mathbf{P}_+$.

■

Mit diesen Informationen ergänzen wir den zweiten Flächendurchlauf mit der Operation $OP_{2.2}$. Zunächst sorgen wir dafür, daß eine leere Knotenliste L_{v_+}, eine leere Kantenliste L_{e_+} und eine leere Flächenliste L_{f_+} zur Repräsentation von P_+ zur Verfügung stehen. Analog halten wir L_{v_-}, L_{e_-} und L_{f_-} für P_- bereit. Wir befinden uns im zweiten Flächendurchlauf nach Abarbeitung der Operation $OP_{2.1}$(f). Demnach ist f die aktuelle Fläche, welche gerade besucht wird.

1. Ist f eine Fläche, welche durch H in die Flächen f_+ und f_- gesplittet wird, so verfahre man wie folgt. (Zur Entscheidung, ob f gesplittet wird oder nicht, betrachte man $OP_{2.1}$ Seite 48)

   - Hänge f_+ in die Liste L_{f_+} und
     Hänge f_- in die Liste L_{f_-}
   - Nun werden alle Kanten und damit auch alle Knoten von f_+ durchlaufen und entsprechend in die Listen L_{e_+} bzw. L_{v_+} gehängt.

     Gleichzeitig zählen wir die Knoten von f und belegen $f \rightarrow size$.
     Analog verfahren wir mit der Fläche f_-.

2. Ist f eine Fläche, welche nicht gesplittet wird, so wissen wir zunächst nicht, ob $f \in P_+$ oder $f \in P_-$.

   - Wir halten 2 spezielle leere Zusatzlisten, eine Knotenliste SL_v und eine Kantenliste SL_e bereit.
   - Analog wie oben bereits beschrieben, werden alle Kanten und damit auch alle Knoten von f in Reihenfolge durchlaufen. Dabei hängt man die Knoten nach SL_v und die Kanten nach SL_e. Bei diesem Durchlauf betrachtet man zusätzlich für jeden Knoten seine Lage bzgl. H. Mit Lemma 6 und 7 (S. 52 ff.) läßt sich spätestens nach Durchlauf aller Knoten entscheiden, ob $f \in P_+$ oder $f \in P_-$.
   - $f \in P+$ dann:
     $f_+ \rightarrow mark = 1;$ // d.h. $f_+ \subseteq P_+$
     hänge f nach L_{f_+};
     hänge SL_v an die Liste L_{v_+} ;
     hänge SL_e an die Liste L_{e_+} ;
     (Dies ist in konstanter Zeit möglich – siehe LEDA Manual)
   - $f \in P-$ dann:
     $f_- \rightarrow mark = -1;$ // d.h. $f_- \subseteq P_-$
     $\vdots$

Damit ist Operation $OP_{2.2}$ beendet.

3.2.7 Zusammenfassung

Nach Ablauf aller Flächen im zweiten Flächendurchlauf gilt mit der zusätzlichen Durchführung von $OP_{2.2}$:

- Die Listen L_v, L_e und L_f repräsentieren das Polyeder P gemäß dem Datenmodell eines Polyeders. L_f enthält keine Fläche, welche gesplittet wird.

- Die Listen L_{v+}, L_{e+} und L_{f+} repräsentieren den Teilrand $B_P \cap H_+^\circ \subseteq B_{P_+}$ von P_+ gemäß dem Datenmodell eines Polyeders. (analog L_{v-}, L_{e-} und L_{f-} bzgl. P_-)
- Sei $v \in L_v$ ein Knoten von P mit $v \in H$, so gilt:

$$v \in L_{v_+} \quad \wedge \quad v \in L_{v_-}$$

Zur Laufzeit dieses zweiten Teilverfahrens des Split-Algorithmus sei noch folgendes anzumerken. Die Objekte (Knoten, Kanten, Flächen) des Polyeders P werden zweimal durchlaufen. Wir benötigen somit eine Laufzeit von $O(n)$, wobei man sich leicht selbst davon überzeugen kann, daß die Laufzeitkonstante vernachlässigbar klein ist.

3.3 Konstruktion von P_+ (P_-)

Wir bechränken uns auf die Konstruktion des Polyeders P_+. (Die Konstruktion von P_- erfolgt analog.)
Die Listen L_v, L_e, L_f beschreiben den Rand B_P von P und es gilt:

$$B_P = \underbrace{(B_P \cap H_+^\circ)}_{B^*_{P_+}} \cup \underbrace{(B_P \cap H_-^\circ)}_{B^*_{P_-}}$$

Der positive Teilrand $B^*_{P_+}$ von P wird durch die Objekte der Listen L_{v_+}, L_{e_+}, L_{f_+} repräsentiert. Der negative Teilrand $B^*_{P_-}$ von P wird durch die Objekte der Listen L_{v_-}, L_{e_-}, L_{f_-} repräsentiert. Sei G der Graph, welcher durch die Listen L_v, L_e, L_f beschrieben wird.

1. Nach Abschnitt 1.3 wissen wir, daß G zusammenhängend ist.
2. Weiterhin gilt: $L_{f_+} \cup L_{f_-} = L_f$

Daraus entnehmen wir, daß $B^*_{P_+}$ mit $B^*_{P_-}$ durch korrespondierende Kanten in G miteinander verbunden sind. Siehe dazu Abbildung 3.9.

Für den Rand B_{P_+} von P gilt: $B_{P_+} = \overbrace{(B_P \cap H_+^\circ)}^{B^*_{P_+}} \cup f_H$
Dabei ist $f_H = P \cap H$ das Polygon, welches durch den Split von P mit H zwischen P_+ und P_- entsteht (s. Abschnitt 1.4.3). Da in G der Teilrand $B^*_{P_+}$ noch mit dem Teilrand $B^*_{P_-}$ verbunden ist, ist es nun unser Ziel, den Graphen so zu verarbeiten, daß $B^*_{P_+}$ mit dem Polygon f_H anstelle von $B^*_{P_-}$ verbunden ist. Nach Satz 7 (S.14) muß das Polygon f_H, welches P_+ begrenzt, negativ zu H orientiert sein. Bzgl. P_+ bezeichnen wir das Polygon f_H nun fortan mit f_{H_+}.

f_{H_+} werde repräsentiert durch die Kantenordnung $EO = (e_1, ..., e_m)$ bzw. durch die Knotenordnung $VO = (v_1, ..., v_m)$ Da $L_{f_+} \cup f_{H_+}$ alle Flächen sind, welche den Rand

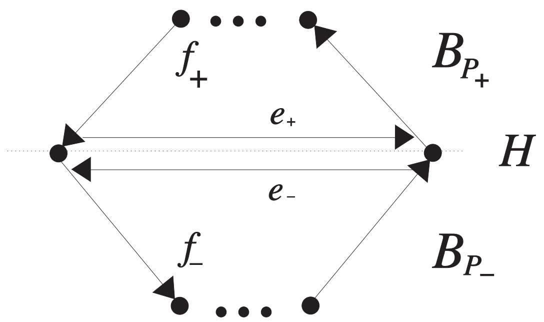

Abbildung 3.9: Flächenberührung durch korrespondierende Kanten an der Splitebene H

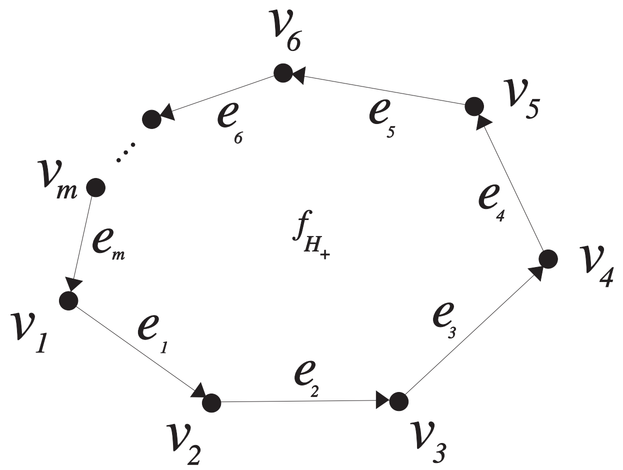

Abbildung 3.10: Splitpolygonfläche f_{H_+}

B_{P_+} des Polyeders P_+ beschreiben und da f_{H_+} durch m Kanten begrenzt ist, existieren in L_{f_+} m Flächen $\overline{f_1}, ..., \overline{f_m}$, welche f_{H_+} in den Kanten $\overline{e_1}, ..., \overline{e_m}$ berühren. Dabei gilt $\overline{e_1} = co(e_1), ..., \overline{e_m} = co(e_m)$. Mit den Kanten $\overline{e_1}, ..., \overline{e_m}$ sind auch die Knoten $v_1, ..., v_m$ von f_{H_+} erzeugt, wohingegen die Kanten $e_1, ...e_m$ noch nicht erzeugt sind. Damit die Listen L_{v_+}, L_{e_+} und L_{f_+} das Polyeder P_+ gemäß dem internen Datenmodell beschreiben verfahren wir wie folgt.

1. Erzeuge Speicher für das Polygon f_{H_+} und hänge die Speicheradresse in die Liste L_{f_+}.

2. (a) Besuche alle Kanten $\overline{e}_1$, ..., $\overline{e}_m$ in dieser Reihenfolge.

   Dazu benötigt man eine Laufzeit von $O(n)$, wobei n die Anzahl der gerichteten Kanten des Polyeders ist.

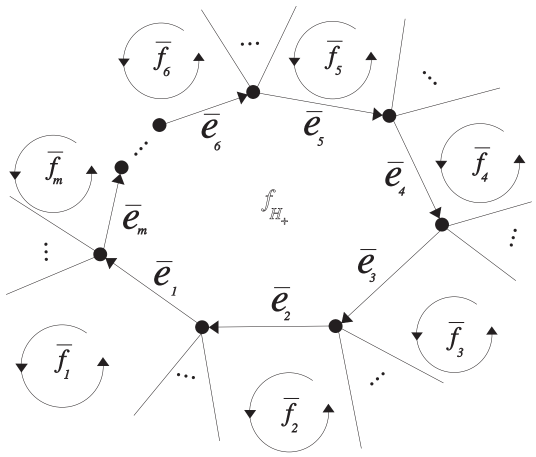

Abbildung 3.11: Nachbarflächen der Splitpolygonfläche f_{H_+}

(b) Für jede Kante $\overline{e}_i$ $(1 \leq i \leq m)$, welche besucht wird, führe man die Operation $\mathbf{OP}(\overline{\mathbf{e}}_\mathbf{i})$ in konstanter Zeit durch.

Zunächst beschäftigen wir uns mit 2(a), wodurch die Kantenordnung von f_{H_+} bestimmt wird. Anschließend wird mit 2(b) die Operation **OP** beschrieben.

3.3.1 Kantenordnung von f_{H_+}

Um die Kanten $\overline{e_1}, ..., \overline{e_m}$ in Reihenfolge zu besuchen, verfahren wir wie folgt:

1. Bestimme einen beliebigen Begrenzungsknoten v von f_{H_+}.

   Existiert ein solcher Knoten $v \in H$?

   Da P gesplittet wird:
   $\Rightarrow \quad \exists\, v_+ \in P \;\wedge\; \exists v_- \in P \;\therefore\; v_+ \in H_+ \;\wedge\; v_- \in H_-$
   Da der Graph G zusammenhängend ist, existiert ein Pfad von v_+ nach v_- (über die Kanten von G).
   $\Rightarrow \quad \exists\, e^* \in P \quad \therefore \quad e^* \cap H \;\neq\; \emptyset$

   Weiterhin wissen wir, daß die Kantenliste L_e keine Kanten (mehr) enthält, welche durch H gesplittet werden. Damit muß e eine Kante sein, welche H in einem Knoten $v^* \in H$ berührt. Mit v^* existiert also ein Knoten von P, welcher in H liegt.

   Bestimmung von $v \in H$

Im zweiten Flächendurchlauf des vorigen Abschnittes werden keine (zusätzlichen) Knoten mehr erzeugt. Es werden jedoch alle Knoten noch einmal durchlaufen. Dabei merkt man sich einen Knoten $v \in H$.

2. Sei o.B.d.A. $v_1 := v$, dann bestimme $\overline{e_1}$. (bzgl. Abbildung 3.11

   Die Kante $\overline{e_1}$ ist von allen bisher erzeugten In- und Outputkanten des Knotens v_1 wie folgt eindeutig identifizierbar.

$$\overline{e_1} \to v_2 = v_1 \; und \; \overline{e_1} \to v_1 \in H \; und \; \overline{e_1} \in P_+ \quad (\Leftrightarrow \quad e \to f \subseteq P_+) \\ (s.S.50 Abschnitt 3.2.6)$$

   Alle In- und Outputkanten von v_1 und somit auch die Kante e_1 können in grad(v_1) konstanten Schritten bestimmt werden. (Abschnitt 2.2 S. 26 ff.)

3. Bestimme den Knoten $v_2 = \overline{e_1} \to v_1$

4. Durch sukzessive Anwendung der Schritte 2 und 3 bestimme man in Reihenfolge die Kanten $\overline{e_2}, ..., \overline{e_m}$ bzw. die Knoten $v_3, ..., v_m$.

3.3.2 Verbindung von $\mathbf{B^*_{P_+}}$ mit $\mathbf{f_{H_+}}$

Die Kanten $\overline{e}_1, ..., \overline{e}_m$, werden in dieser Reihenfolge von der Operation OP verarbeitet. Dabei werden die Kanten $e_1, ..., e_m$ von f_{H_+} in dieser Reihenfolge erzeugt und entsprechend zyklisch (gemäß der Kantenordnung von f_{H_+}) untereinander verbunden. Jede erzeugte Kante e_i wird dabei auch mit ihrer Fläche f_{H_+} verbunden und in die Liste L_{e_+} gehängt. Schließlich werden die Kanten $e_1, ..., e_m$ mit den Kanten $\overline{e}_1, ..., \overline{e}_m$ in dieser Reihenfolge korrespondierend miteinander verbunden, so daß – wie die Überschrift bereits andeutet – der positive Teilrand $B^*_{P_+}$ des Polyeders P mit dem Polygon f_{H_+} verbunden wird. Nach Abarbeitung aller m Durchläufe der Operation OP, wird demnach das Polyeder P_+ gemäß dem internen Datenmodell durch die Listen L_{v_+}, L_{e_+} und L_{f_+} repräsentiert.
Die Operation OP wird wie folgt durchgeführt.

$OP(\overline{e}_i)\{$

1. (a) Erzeuge Speicherplatz für die Kante e_i;
   Hänge die Speicheradresse von e_i in die Kantenliste L_{e_+};
   (b) //Belege die Kante e_i mit ihren Knoten wie folgt:
   $e_i \to v_1 = \overline{e}_i \to v_2$;
   $e_i \to v_2 = \overline{e}_i \to v_1$;
   (c) Verbinde die Kante e_i mit ihrer Fläche f_{H_+};

2. //Die Operation OP werde zum i-ten mal durchlaufen. $(1 \leq i \leq m)$

   //Verbinde die Kante e_i mit ihrer zyklischen Vorgängerkante
   //(bzgl. der Kantenordnung von H) — falls diese bereits erzeugt ist.

   (a) $\mathbf{i = 1}$

      //Somit gilt: $e_i = e_1$
      //Die Vorgängerkante e_m von e_1 ist noch nicht erzeugt.
      //Damit wird e_1 mit e_m vorläufig noch nicht verbunden.
      //Wir merken uns e_1 in der Variablen *first_edge*.
      //Zusätzlich merken wir uns e_1 in der Variablen *pred_edge*.

      $first\_edge = e_i;$
      $pred\_edge = e_i;$

   (b) $(\mathbf{1 < i \leq m})$

      // $e_{i-1} = pred\_edge$

      Verbinde die Kante e_i mit ihrer zyklischen Vorgängerkante e_{i-1};
      $pred\_edge = e_i;$

   (c) $(\mathbf{i = m})$

      // $e_1 = first\_edge$ und $e_i = e_m$

      Verbinde die Kante e_m mit ihrer zyklischen Nachfolgerkante e_1;

3. Verbinde die Kante e_i mit $\overline{e}_i$ korrespondierend;

}

3.4 Konstruktion von P_+ und P_- (gleichzeitig)

Das Polyeder P_- läßt sich analog zu P_+ erzeugen. Will man P_+ und P_- (gleichzeitig) erzeugen, so verdoppelt sich der Aufwand und damit auch die Laufzeit. Dabei ist zu beachten, daß ein Knoten $v \in H$ sowohl ein Knoten von P_+ als auch ein Knoten von P_- ist. Er kommt jedoch nur einfach vor und muß infolgedessen dupliziert werden. Diesbezüglich erweitern wir den Splitalgorithmus wie folgt.

1. Im Splitalgorithmus befinden wir uns im zweiten Teilverfahren nach Abarbeitung des ersten Flächendurchlaufs unmittelbar vor dem zweiten Flächendurchlauf (siehe Abschnitt 3.2 Seite 42), dann gilt:
   Alle Knoten von P_+ bzw. P_- sind erzeugt — mit obig geschilderter Ausnahme.

Wir erweitern den vorhandenen Algorithmus wie folgt. Für jeden Knoten in diesem zweiten Flächendurchlauf
do{

- Erzeuge die Knoten v_+ und v_-; Hänge v_+ nach L_+; Hänge v_- nach L_-;
- Weise diesen Knoten die Koordinaten von v zu .
- Konstruiere eine Vater-Sohn-Verbindung zwischen den Kindern v_+,v_- und dem Vaterknoten v.

}

2. Wir befinden uns im dritten Teilverfahren des Splitalgorithmus, in der Operation, mit welcher die Knotenordnung von f_{H_+} bestimmt wird (Abschnitt 3.3.1 S. 56). Für jeden Knoten $v \in H$ besuchen wir nun nicht nur einen Teil, sondern alle In- bzw. Outputkanten von v. (Dies ändert nichts an der bisherigen Laufzeitabschätzung). Für alle positiven Kanten e ($e \subseteq P_+$) ersetzen wir v durch v_+. Analog ersetzen wir für alle negativen Kanten e ($e \subseteq P_-$) v durch v_-. Zur Entscheidung, ob $e \subseteq P_+$ oder $e \subseteq P_-$ siehe Abschnitt 3.2.6 Seite 50. Gleichzeitig wird v_+ eine dieser positiven und v_- eine dieser negativen Kanten zugewiesen. Der Speicher des Knotens v kann danach freigegeben werden.

3. Analog wie im Abschnitt 3.3.2 Seite 57 konstruiert man zu f_{H_+} nun auch f_{H_-}. Die Operation OP verarbeitet die Kanten $\overline{e}_1, ..., \overline{e}_m$ in Reihenfolge. So, wie in der Operation OP($\overline{e}_i$) die Kanten $\overline{e}_i$ verarbeitet werden, werden jetzt zusätzlich in dieser Operation die Kanten co($\overline{e}_i$) in analoger Weise bearbeitet. Dabei berücksichtige man, daß f_{H_-} entgegengesetzt zu f_{H_+} gerichtet ist. Danach wird das Polyeder P_+ durch die Listen L_{v_+}, L_{e_+}, L_{f_+} und das Polyeder P_- durch die Listen L_{v_-}, L_{e_-}, L_{f_-} gemäß des internen Datenmodells repräsentiert. Die Listen L_v, L_e, L_f können geleert werden.

Kapitel 4

Reguläres Objekt und BSP-Tree

Mit Hilfe der Operation Split-Polyeder können, beispielsweise aus einem Würfel, beliebige konvexe Polyeder herausmodelliert werden. Wir interessieren uns nun aber nicht nur für konvexe Objekte, sondern für beliebige, räumlich begrenzte, zusammenhängende Objekte. Um ein solches Objekte zu präzisieren, definieren wir uns den Begriff *reguläres Objekt*. Wir werden erkennen, daß dieses sich durch ein Polyeder, welches aus einer Vereinigung konvexer Polyeder zusammengesetzt ist, approximieren läßt. Dieses wiederum läßt sich durch einen sogenannten *BSP-Tree* repräsentieren — *BSP-Representation*. Dies ist ein Binärbaum, dessen Knoten konvexe Polyeder repräsentieren, und dessen Blätter in ihrer Vereinigung eine Partitionierung der Wurzel des Baumes bilden. Mit einer Vereinigung einer bestimmten Auswahl von Blättern, läßt sich ein bestimmtes Polyeder repräsentieren. Es folgt eine Analyse, in welcher wir wichtige Eigenschaften des Baumes in Form von Sätzen festhalten. Abschließend zu diesem Kapitel werden noch einige grundlegende Algorithmen bezüglich des BSP-Tree's besprochen.

4.1 Reguläre Objekte

Definition 4.1 (reguläres Objekt)

Sei $V \subseteq \mathbb{R}^d$; $(\ d \in \{1,2,3\}\ )$
V *ist ein* **reguläres Objekt** *wenn gilt:*

1. *V ist kompakt.*
2. *V ist wegzusammenhängend.*
3. *Die Menge der inneren Punkte von V ist nicht leer.*
4. *Die Menge V entsprecht dem Abschluß ihrer inneren Punkte.*

Konvexe reguläre Objekte lassen sich durch konvexe Polyeder beliebig nah approximieren. Nicht konvexe reguläre Objekte lassen sich partitionieren in konvexe reguläre Objekte, welche wiederum durch konvexe Polyeder approximiert werden können. Eine Vereinigung konvexer Polyeder in diesem Sinne entspricht aber gerade einem Polyeder, welches wir uns in Definition 4.2 definieren.

Lemma 8
P_1, $P_2 \subseteq \mathbb{R}^d$ ($d \in \{1,2,3\}$) seien zwei konvexe Hyperpolyeder für die gilt:

$$B_{P_1} \cap B_{P_2} = P_1 \cap P_2$$

Da P_1, P_2 konvex sind, und diese sich nur in ihren Rändern berühren, trifft genau einer der folgenden 4 Fälle ein.

1. $P_1 \cap P_2 = \emptyset$ $\qquad d \in \{1,2,3\}$
2. $P_1 \cap P_2 = \{S\} \subseteq \mathbb{R}^d$ $\qquad d \in \{1,2,3\}$
3. $\exists! g \subseteq \mathbb{R}^d \quad \because \quad P_1 \cap P_2 \subseteq g$ *(Gerade)* $\qquad d \in \{2,3\}$
4. $\exists! H \subseteq \mathbb{R}^d \quad \because \quad P_1 \cap P_2 \subseteq H$ *(Ebene)* $\qquad d \in \{3\}$
   Indeisem Fall existieren zwei Begrenzungspolygone
   $f_1 \subseteq B_{P_1}$ und $f_2 \subseteq B_{P_2}$ wobei $f_1 := P_1 \cap H$ und $f_2 := P_2 \cap H$, dann gilt:
   $$f_1 \cap f_2 = P_1 \cap P_2 \quad \textit{(ist ein konvexes Polygon)}$$

Bezeichnung 13
Bezüglich der Fälle des vorigen Satzes bezeichnen wir P_1, P_2 als:

1. **unvereinbar**
2. **punktvereinbar**
3. **streckenvereinbar**
4. **flächenvereinbar**

In den Fällen 2 bis 4 bezeichnen wir P_1, P_2 auch als **vereinbar**

Definition 4.2 (Hyperpolyeder)

Sei $M = \{P_1, ..., P_m\}$ eine Menge konvexer Hyperpolyeder aus $\mathbb{R}^d$ ($d \in \{1,2,3\}$). $P = \bigcup_{i=1}^m P_i$ ist ein **Hyperpolyeder** *wenn:*

1. $P_i \cap P_j = B_{P_i} \cap B_{P_j}$ $\qquad i,j \in \{1,...,m\}$ *und* $i \neq j$
2. *$G = (V,E)$ ist ein zusammenhängender Graph mit*
   Knotenmenge $V = \{1,...,m\}$ und Kantenmenge E
   dabei gilt für $i,j \in V$ mit $i \neq j$
   $$e = \{i,j\} \in E \Leftrightarrow P_i, P_j \textit{ sind vereinbar}$$

Bezeichnung 14
Bezüglich Satz 4.2 vereinbaren wir:
Im Falle $d = 1$ ist P eine **Strecke**.
Im Falle $d = 2$ ist P ein **Polygon**.
Im Falle $d = 3$ ist P ein **Polyeder**.

Definition 4.3 (regulärer-Polyeder)

Sei $M = \{P_1, ..., P_m\}$ eine Menge konvexer Polyeder aus $I\!R^3$ und sei $P = \bigcup_{i=1}^{m} P_i$ ein Polyeder. P ist ein reguläres *Polyeder, wenn die konvexen Polyeder aus M paarweise weder punktvereinbar noch streckenvereinbar sind.*

Taucht in dem nun folgenden Teil der Diplomarbeit der Begriff Polyeder auf, so wollen wir von nun an darunter reguläre Polyeder verstehen. Dies ist zwar eine Einschränkung, dennoch kann man leicht einsehen, daß Polyeder beliebig genau durch reguläre Polyeder approximiert werden können. Mit dieser Einschränkung können wir uns im Paste-Polyeder Verfahren auf wesentliche Teile des Algorithmus konzentrieren. Hat man einmal die prinzipielle Vorgehensweise dieses Algorithmus verstanden, so kann man sich das Verfahren leicht derart erweitern, daß es für Polyeder im Sinne der Definition 4.2 gültig ist. In Abbildung 4.1 Seite 62 sehen wir zunächst ein abstraktes Beispiel für ein Hyperpolyeder P der Dimension $d = 2$, welches sich aus den konvexen Teilpolyedern $P_1, P_2, P_3, P_4, P_5, P_6, P_7$ zusammensetzt.

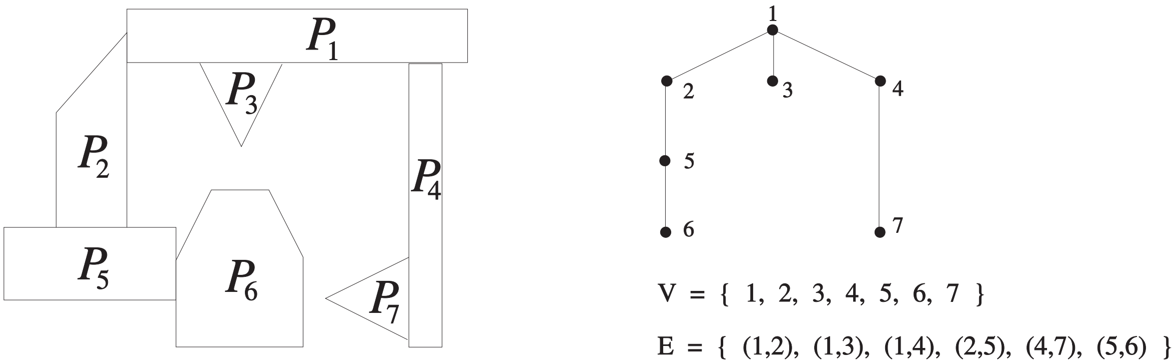

Abbildung 4.1: Hyperpolyeder $P = P_1 \cup ... \cup P_7$ mit zugehörigem Graph $G = (V, E)$

Wie läßt sich nun ein solches Polyeder konstruieren bzw. im Rechner repräsentieren? Betrachten wir dazu als Beispiel das Polyeder $P = (P_1 \cup P_2 \cap P_3 \cap P_4)$ in Abbildung 4.2 Seite 63. Wir bilden die Box, welche das Polyeder minimal umschließt und partitionieren diese durch sukzessive ebene Schnitte entlang der Begrenzungsflächen der

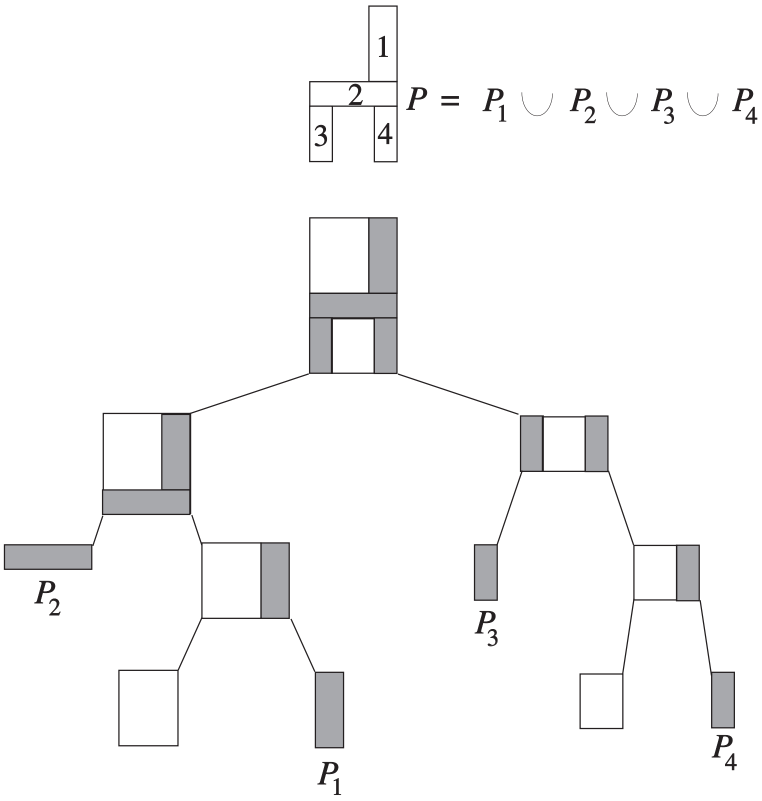

Abbildung 4.2: BSP-Representation eines Polyeders P

konvexen Polyeder $\{P_1, ..., P_4\}$ durch die Operation Split-Polyeder. Wir erhalten als Ergebnis einen Binärbaum — Abbildung 4.2 Seite 63 — bestehend aus konvexen Polyedern. Die Blätter dieses Baumes bilden eine Partitionierung der Box. Vereinigt man hingegen nur die markierten Blätter $\{P_1, ..., P_4\}$, so erhalten wir damit das Polyeder P. Den Baum bezeichnen wir als Binary-Space-Partitioning-Tree — kurz *BSP-Tree*. Die Art dieser Repräsentation für konvexe Polyeder wollen wir als *BSP-Representation* bezeichnen. In den folgenden drei Abschnitten erfahren wir genaueres über den BSP-Tree.

4.2 BSP-Tree

Definition 4.4 (Binärbaum)
Sei $G = (V, E)$ ein gerichteter Graph mit $V = \{1, ..., |V|\}$ (Menge der Knoten des Graphen) und $E = V\mathrm{x}V$ (Menge der Kanten des Graphen).

- $\exists\ !\ w_\circ \in V \ \therefore\ \nexists\ v \in V$ *mit* $(v, w_\circ) \in E \qquad (\Leftrightarrow pred(w_\circ) = \emptyset)$
  Wir bezeichnen $w_\circ$ als **Wurzel** *des Graphen G.*

- *Sei* $v \in V$ *dann gilt:*

  - entweder*:*
    $\exists\,!\; v_1, v_2 \in V \quad \therefore \quad (v, v_1) \in E \wedge (v, v_2) \in E$
    Dann bezeichne man v *als* **Vaterknoten** *von* v_1*,* v_2*.*
    v_1 v_2 *bezeichne man als* **Söhne** *von* v *und untereinander bezeichne man sie als* **Brüder***.*

  - oder*:*
    $\nexists\; v^* \in V \;\; \therefore \;\; (v, v^*) \in E$
    Dann bezeichne man v *als* **Blatt** *in* G*.*

Ein Graph G *mit diesen Eigenschaften bezeichne man als einen* **gerichteten Binärbaum***.*

Definition 4.5 (BSP-Tree)
Sei $G = (V, E)$ *ein gerichteter Binärbaum mit* $V = \{1, ..., |V|\}$ *(Menge der Knoten des Graphen) und* $E = V\mathrm{x}V$ *(Menge der Kanten des Graphen).*

- $\mathcal{P} := \{P_1, ..., P_{|V|}\}$ *seien eine Menge von* $|V|$ *konvexen Hyperpolyedern* $P_i \subseteq I\!R^d$ *mit* $d = \{1,2,3\}$

- $\alpha : V \rightarrow \mathcal{P}$ *ist eine bijektive Funktion.*

- $\mathcal{H} := \{H_1, ..., H_m\}$ *ist eine Menge orientierter Hyperebenen.*

  $\beta : \mathcal{P} \rightarrow \mathcal{H}$ *ist eine partielle Funktion.*

- $w_\circ \in V$ *sei die Wurzel des Baumes* G*.*
  $P_\circ := \alpha(w_\circ)$ *bezeichne man als* **Wurzel(hyper)polyeder** *des Baumes* G*. (*$P_\circ \in \mathcal{P}$*)*

Sei $P \in \mathcal{P}$

1. *(*entweder*)* $\exists\; H \in \mathcal{H} \;\; \therefore \;\; \beta(P) = H$
   Dann und nur dann existieren zwei konvexe Hyperpolyeder P_+ *und* $P_- \quad \therefore$ $P/H = \{P_+, P_-\} \subseteq \mathcal{P}$ *wobei* $P_+ \subseteq H_+^\circ$ *und* $P_- \subseteq H_-^\circ$*.*

   weiterhin gilt dann:

$$\begin{aligned} \left(\alpha^{-1}(P), \alpha^{-1}(P_+)\right) &\in E \\ \left(\alpha^{-1}(P), \alpha^{-1}(P_-)\right) &\in E \end{aligned}$$

Bezeichnung:

$\mathbf{P}$	:	**Vater** *von* P_+ *und* P_-
$\{\mathbf{P}_+, \mathbf{P}_-\}$	:	**Brüder**
P_+	:	**positiver (linker) Sohn**
P_-	:	**negativer (rechter) Sohn**

2. (oder) $\nexists\ H \in \mathcal{H}\ \therefore\ \beta(P) = H$
dann und nur dann $\nexists\ v^* \in V\ \therefore\ (\alpha^{-1}(P), v^*) \in E$
In diesem Fall bezeichne man P *als* **Blatt(hyper)polyeder** *von* G*.*

Wir bezeichnen $\mathbf{T_{BSP}} = (\mathbf{V}, \mathbf{E}, \mathcal{P}, \mathcal{H}, \alpha, \beta, \mathbf{P}_\circ)$ *als* **Binary-Space-Partioning-Tree** *von konvexen Hyperpolyedern.*

4.2.1 Datenmodelle eines BSP-Trees

Auch hier haben wir in Analogie zu den Datenmodellen des konvexen Polyeders ein *externes* und ein *internes* Datenmodell.

4.2.1.1 Externes Datenmodell

1. Daten des Wurzelpolyeders

2. Rekursive Auflistung der Schnittebenen mit Zusatzinformationen

4.2.1.2 Internes Datenmodell

Sei $P \in \mathbb{R}^3$ ein konvexes Polyeder.
P sei repräsentiert in Form des internen Polyederdatenmodells. P werde durch eine orientierte Ebene $H \subseteq \mathbb{R}^3$ durch den in Kapitel 3 beschriebenen Algorithmus in die Polyeder P_+ und P_- gesplittet. P_+ und P_- liegen damit ebenfalls in Form des internen Polyederdatenmodells vor.

- Dann verknüpfe man P_+ und P_- mit ihrem Vaterpolyeder P wie folgt (Vater-Sohn-Beziehung):

  Plus_Sohn(P) = P_+;
  Minus_Sohn(P) = P_-;
  Vater(P_+) = P;
  Vater(P_-) = P;

- Weiterhin belege man die Datenstruktur von P mit der Ebene H und die Datenstrukturen von P_+ bzw. P_- mit den Splitpolygonflächen $f_{H_+} = P_+ \cap H$ bzw. $f_{H_-} = P_- \cap H$.

Split_Ebene(P) = H;
Split_Fläche(P_+) = f_{H_+};
Split_Fläche(P_-) = f_{H_-};

- Solange die Strukturvariable *Split_Ebene* eines Polyeders mit *NULL* belegt ist, handelt es sich bei diesem Polyeder um ein **Blatt**.
- Ist die Strukturvariable *Vater* eines Polyeders mit *NULL* belegt, so handelt es sich bei diesem Polyeder um das **Wurzelpolyeder**.
- Handelt es sich bei P um das Wurzelpolyeder, dann:
  1. Belege P mit der Knotenidentität 1. $id(P) = 1;$
  2. Belege die Wurzel von P mit P. $Wurzel(P) = P;$
  3. Ist P auch Blattpolyeder, so ist sein Knotenzähler mit 1 belegt.
     zähler(P) = 1;
  4. P befindet sich in der Tiefe 0 des Baumes. Tiefe(P) = 0;
- Die beiden neu erzeugten Polyeder P_+ und P_- belege man wie folgt:
  1. Belege P_+ bzw. P_- mit ihrem Wurzelpolyeder.

     Wurzel(P_+) = Wurzel(Vater(P_+));
     Wurzel(P_-) = Wurzel(Vater(P_-));

     Damit hat jedes Polyeder des Polyederbaumes Zugriff auf das Wurzelpolyeder.
  2. Belege P_+ bzw. P_- mit ihren Knotenidentitäten

     zähler(Wurzel(P_+))++; id(P_+) = zähler(Wurzel(P_+));
     zähler(Wurzel(P_-))++; id(P_-) = zähler(Wurzel(P_-));
     Damit sind die Knoten des Polyederbaumes topologisch sortiert.
  3. Belege P_+ bzw. P_- mit ihrer Tiefe
     Tiefe(P_+) = Tiefe(P)+1;
     Tiefe(P_-) = Tiefe(P)+1;

4.2.2 BSP-Tree Analyse

Satz 10

Sei $T_{BSP} = (V, E, \mathcal{P}, \mathcal{H}, \alpha, \beta, P_\circ)$ ein BSP-Tree. Seien $P_1, P_2 \in \mathcal{P}$ zwei konvexe Hyperpolyeder des Baumes. Dann gilt:

$$\text{Es existiert ein Pfad } v_0, ..., v_k \text{ in } T_{BSP} \text{ mit } P_1 = \alpha(v_0) \text{ und } P_2 = \alpha(v_k).$$
$$\Leftrightarrow$$
$$P_2 \subseteq P_1$$

Weiterhin bezeichnen wir P_1 als **Vorfahre** *von P_2 und P_2 als* **Nachkomme** *von P_1.*

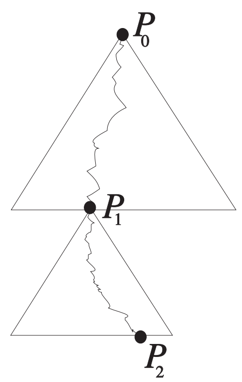

Abbildung 4.3: Polyederpfade im BSP-Tree

Beweis : (unmittelbar aus Definition 4.5 ersichtlich) (siehe auch Abildung 4.5 bzw. 4.2 (S. 63))

Satz 11

Sei $T_{BSP} = (V, E, \mathcal{P}, \mathcal{H}, \alpha, \beta, P_\circ)$ ein BSP-Tree. $\{P_1, ..., P_m\} \subseteq \mathcal{P}$ sei die Menge der Blatt(hyper)polyeder von T_{BSP}, dann gilt:

$$\bigcup_{i=1}^{m} P_i = P_\circ$$

Beweis: Induktion über $|\mathcal{P}|$

Induktionsanfang
Für $|\mathcal{P}| = 1$ ist die Behauptung trivial, da in diesem Falle $P_\circ$ einziger Blattpolyeder

des Baumes ist.

Induktionsannahme

Die Behauptung sei erfüllt für $|\mathcal{P}| \leq n_\circ \in \mathbb{N}$. Sei $T_{BSP} = (V, E, \mathcal{P}, \mathcal{H}, \alpha, \beta, P_\circ)$ ein BSP-Tree mit $|\mathcal{P}| = n_\circ$.

Sei $\{P_1, ..., P_m\} \subseteq \mathcal{P}$ die Menge der Blätter von T_{BSP}, dann gilt: $\bigcup_{i=1}^{m} P_i = P_\circ$

Induktionsschluß

Wir erweitern den Baum mit einen Splitvorgang durch die Ebene H an dem Blattpolyeder P_m (o.B.d.A.) derart, daß gilt: $P_m/H = \{P_{m+}, P_{m-}\}$.

Damit ist $\{P_1, ..., P_m\} \cup \{P_{m+}, P_{m-}\} \subseteq \mathcal{P}$ die erweiterte Menge der Blätter des derart modifizierten Baumes. Weiterhin wissen wir, daß: $P_{m+} \cup P_{m-} = P_m$. Demnach gilt:

$$P_\circ = \bigcup_{i=1}^{m} P_i \;=\; \bigcup_{i=1}^{m-1} P_i \cup P_m \;=\; \bigcup_{i=1}^{m-1} P_i \cup (P_{m+} \cup P_{m-})$$

■

Bezeichnung 15

Sei T_{BSP} ein BSP-Tree für konvexe Hyperpolyeder und $P_\circ$ die Wurzel des Baumes. Die Menge der Blätter des Baumes bezeichnen wir auch als **Partitionierung** *von $P_\circ$ durch T_{BSP}.*

Satz 12

Sei $T_{BSP} = (V, E, \mathcal{P}, \mathcal{H}, \alpha, \beta, P_\circ)$ ein BSP-Tree von Hyperpolyedern. Seien P_1, P_2 aus $\mathcal{P}$. Dann gilt:

$$\begin{array}{ccc} P_1 \not\subseteq P_2 & \textit{und} & P_2 \not\subseteq P_1 \\ & \Leftrightarrow & \\ B_{P_1} \cap B_{P_2} & = & P_1 \cap P_2 \end{array}$$

D.h. Sind diese Aussagen wahr, *so berühren sich P_1, P_2 höchstens in ihren Rändern.*

Beweis: Induktion über $|\mathcal{P}|$

$\Leftarrow$ (trivial)

$\Rightarrow$ durch Induktion

Sei $T_{BSP} = (V, E, \mathcal{P}, \mathcal{H}, \alpha, \beta, P_\circ)$ ein BSP-Tree.

Induktionsanfang

Sei $|\mathcal{P}| = 1$ dann: $\nexists P_1, P_2 \in \mathcal{P} \quad .\dot{}\,. \quad P_1 \not\subseteq P_2$ und $P_2 \not\subseteq P_1$

Induktionsannahme
Die Behauptung sei erfüllt für $|\mathcal{P}| \leq n_\circ \in \mathbb{N}$. Sei $T_{BSP} = (V, E, \mathcal{P}, \mathcal{H}, \alpha, \beta, P_\circ)$ ein BSP-Tree mit $|\mathcal{P}| = n_\circ$.

Induktionsbehauptung:
Sei $\{P_1, ..., P_m\} \subseteq \mathcal{P}$ die Menge der Blätter von T_{BSP}, und sei P_{Blatt} ein Blattpolyeder aus dieser Menge. Wir erweitern den Baum mit einem Splitvorgang durch die Ebene H an dem Polyeder P_{Blatt} derart, daß gilt: $P_{Blatt}/H = \{P^+_{Blatt}, P^-_{Blatt}\}$. Sei $P \in \mathcal{P}$, dann ist zu zeigen:

$$B_P \cap B_{P^+_{Blatt}} = P \cap P^+_{Blatt} \quad falls \quad P \not\subseteq P^+_{Blatt} \;\; und \;\; P^+_{Blatt} \not\subseteq P$$

Induktionsschluß:
Wir unterscheiden dazu folgende Fälle:

1. $P_{Blatt} \subseteq P \Rightarrow P^+_{Blatt} \subseteq P$ (Voraussetzung nicht erfüllt)
2. $P \subseteq P_{Blatt}$
   Dann wäre P_{Blatt} nach Satz 10 Vorfahre von P. *Widerspruch* zu P_{Blatt} war Blattpolyeder vor dem Splitvorgang. (Voraussetzung nicht erfüllt)
3. $P \not\subseteq P_{Blatt}$ und $P_{Blatt} \not\subseteq P$

   Demnach berühren sich P und P_{Blatt} nach Induktionsannnahme (höchstens) in ihren Rändern. Somit berühren sich die Polyeder P^+_{Blatt} und P auch allenfalls in ihren Rändern und es gilt:

$$B_P \cap B_{P^+_{Blatt}} = P \cap P^+_{Blatt}$$

■

4.3 Algorithmen zum BSP-Tree

4.3.1 Baumdurchlauf

Sei T_{BSP} ein im internen Datenmodell gegebener BSP-Tree. $P_\circ$ sei die Wurzel dieses Baumes. Wir wollen alle Knoten P des Baumes besuchen und bei jedem Besuch die Operation $visit\_node(P)$ durchführen. Dazu gibt es 3 effiziente Algorithmen Pre_Order, $Post\_Order$, In_Order mit einer Laufzeit von je $O(\sum_{i=1}^{n} t_i(visit\_node))$. (Man findet diese Algorithmen auch in [Meh84] Band 1.) Dabei sei $t_i(visit\_node)$ die Laufzeit zur Verarbeitung eines Knotens P_i mit der Operation $visit\_node(P_i)$ und n die Anzahl der Knoten des Baumes. Alle 3 Algorithmen durchlaufen den Baum rekursiv. Die rekursiven Algorithmen bekommen als Ausgangsargument die Wurzel $P_\circ$ des Baumes T_{BSP}.

Aufruf: $Pre\_Order(P_\circ)$; $Post\_Order(P_\circ)$; $In\_Order(P_\circ)$;

Pre-Order

Wir besuchen bei jedem Knoten P zuerst den Knoten P, dann den linken und anschließend den rechten Teilbaum.

```
Pre_Order(P){
    visit_node(P);
    if(Left(P)) Pre_Order(Left(P));
    if(Right(P)) Pre_Order(Right(P));
}
```

Post-Order

Wir besuchen bei jedem Knoten P zuerst den linken, anschließend den rechten Teilbaum, anschließend den Knoten P selbst. (Algorithmus analog wie oben)

In-Order

Wir besuchen bei jedem Knoten P zuerst den linken Teilbaum, dann den Knoten P und anschließend den rechten Teilbaum. (Algorithmus analog wie oben)

4.3.2 Partitionieralgorithmus

Zur Erinnerung:
Die Operation $Cut(P_1, P_2)$ (S. 33) berechnet, durch iterative Anwendung von Splitvorgängen, für 2 konvexe Hyperpolyeder P_1, P_2 den Schnitt:

$$P_{cut} = P_1 \cap P_2$$

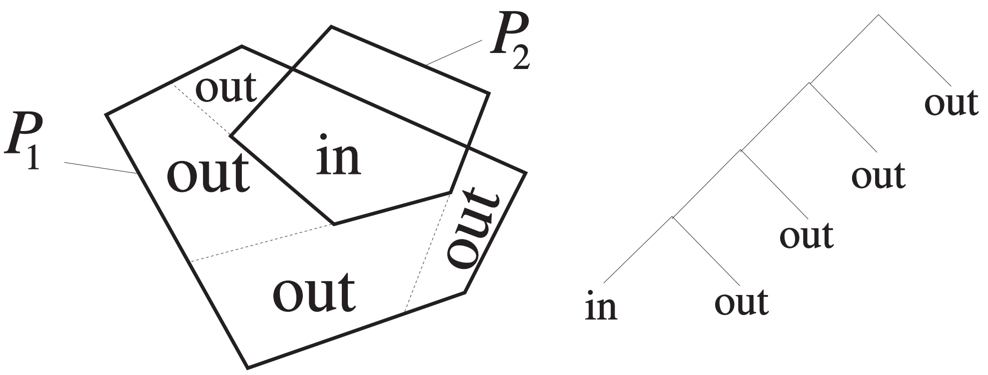

Abbildung 4.4: Partitionierung von P_1 durch P_2

Dadurch entsteht ein BSP-Tree T_{BSP} mit P_1 (o.B.d.A.) als Wurzel. Man verfolge den Pfad, welcher von der Wurzel ausgeht und stets in den nächst möglichen positiven Unterbaum absteigt. Am Ende dieses Pfades findet man P_{cut} als Blattpolyeder. Diesbezüglich erweitern wir nun die Operation *Cut* zur Operation *Partitionierung*. In ihr wird zusätzlich der obig angesprochene BSP-Tree T_{BSP} mit P_1 als Wurzel kreiert. Weiterhin markieren wir P_{cut} mit **IN**, die übrigen Blätter mit **OUT**. Wir sagen:

„ P_1 wird durch P_2 partitioniert“.

Außerdem gilt: $P_1 \setminus P_2$ = Vereinigung aller OUT-Blätter

4.3.3 Seep-Algorithmus

Definition 4.6 (sickern)

1. *Sei* $T_{BSP} = (V, E, \mathcal{P}, \mathcal{H}, \alpha, \beta, P_\circ)$ *ein BSP-Tree mit* $P_\circ$ *als Wurzel.*

2. *Sei* P^* *ein Polyeder mit* $P^* \subseteq P_\circ$.

3. *Sei* $P \in \mathcal{P}$ *ein beliebiges Polyeder des Baumes* T_{BSP}

4. *new(P) ist eine zusätzliche Strukturvariable des Polyeders P. Wir können sie mit einem Verweis auf ein beliebiges Polyeder belegen. Ist new(P) = NULL, dann ist der Verweis leer.*

5. *Sei P kein Blatt, dann ist* $H(P) := \beta(P) \in \mathcal{H}$ *die zu P zugehörige Schnittebene derart, daß* $P/H(P) = \{left\_son(P), right\_son(P)\}$.

„ *Wir lassen* $\mathbf{P}^*$ *mit der Operation* **OP** *in den Baum* $\mathbf{T_{BSP}}$ **sickern** “.

Darunter verstehen wir folgenden algorithmischen Vorgang, den wir als **Seep - Algorithmus** *(seep engl. sickern) bezeichnen. Er ist eine Modifikation des Pre-Order Algorithmus (S. 70). (Eine Anmerkung zur Operation OP finden wir auf Seite 74.)*

```
Seep(P*){
    new(P∘) := P*;          // Initialisierung
    Pre_Order*(P∘);
}
```

```
Pre_Order*(P){
    visit_node(P);

    if(left_son(P)){
        if(new(left_son(P))) Pre_Order*(left_son(P));
    }

    if(right_son(P){
        if(new(right_son(P))) Pre_Order*(right_son(P));
    }

}
```

```
visit_node(P){

    if(P ist kein Blatt){
        H := H(P);
        Splitte das Polyeder new(P) durch die Ebene H;
        Zusatzanweisungen();
    }

    else // (P ist Blatt){
        OP(P, new(P));
    }

}
```

Zusätzlich wird folgende Fallunterscheidung anstelle des Befehls *Zusatzanweisungen()* durchgeführt.

Fall	**result. Vorgang**	**zusätzliche Anweisungen**
$new(P) \subseteq H_+^\circ$	kein Split	$new(left\_son(P)) = new(P)$; $new(right\_son(P)) =$ NULL ;
$new(P)/H = \{new(P)_+, new(P)_-\}$	Split	$new(left\_son(P)) = new(P)_+$; $new(right\_son(P)) = new(P)_-$;
$new(P) \subseteq H_-^\circ$	kein Split	$new(left\_son(P)) =$ NULL ; $new(right\_son(P)) = new(P)$;

Tabelle 4.1: Zusatzanweisungen zur Operation **visit_node**

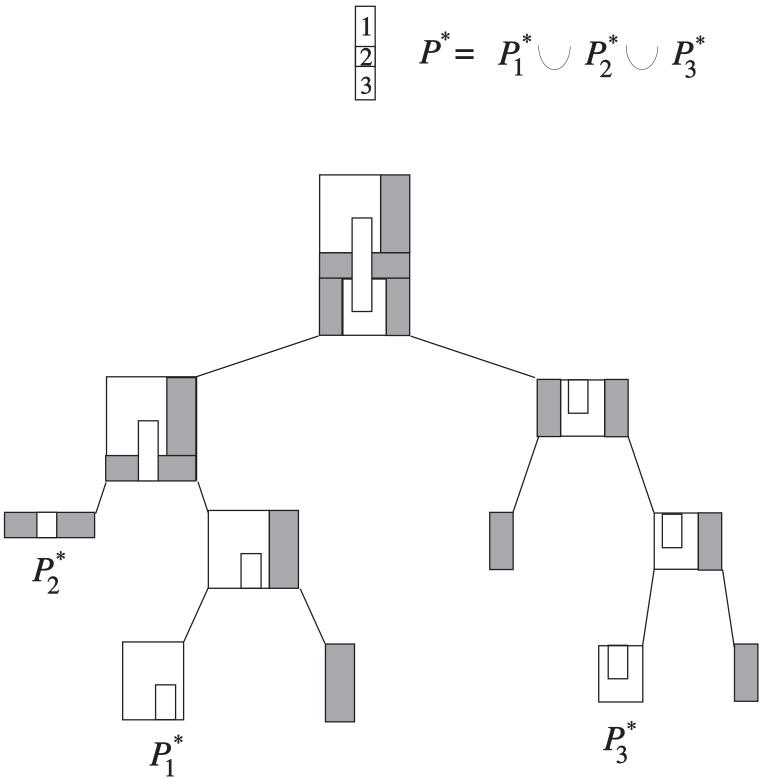

Abbildung 4.5: P^* *sickert* in den BSP-Tree

In Abbildung 4.5 sehen wir als Beispiel einen Sickervorgang eines Polyeders P^* in den Baum, der uns bereits aus Abbildung 4.2 Seite 63 bekannt ist.

Satz 13

Sei T_{BSP} ein BSP-Tree mit $P_\circ$ als Wurzel. Wir lassen $P^* \subseteq P_\circ$ in den Baum sickern . Seien $\{P_1, ..., P_m\}$ die Blätter von T_{BSP} nach dem Sickervorgang, für die $new(P_i) \neq NULL$ gilt $(1 \leq i \leq m)$, dann gilt:

1. $P^* = \bigcup_{i=1}^{m} new(P_i)$ *(Partitionierung von P^*)*

2. $\forall\ P \in \mathcal{P}$ *mit* $new(P) \neq NULL$ *gilt:* $new(P) \subseteq P$

Anmerkung zur Operation *OP*

Die Operation *OP* steht stellvertretend für eine beliebige Operation mit 2 Polyedern P_1, P_2 als Argumente, wie in Operation *visit_node* ersichtlich ist. Anstelle der Operation *OP*(P_1,P_2) können wir beispielsweise die Operation *Partitionierung*(P_1,P_2) (s.S. 70) einsetzen. In unserem Beispiel in Abbildung 4.5 , würden dadurch die Blätter, in welche die Polyeder P_1^*, P_2^*, P_3^* *gesickert* sind, durch die Polyeder P_1^*, P_2^*, P_3^* partitioniert werden, wodurch der Baum erweitert wird. Anwendung findet der Seep-Algorithmus in Abschnitt 5.2.1(S. 86) und in Abschnitt 6.1 (S. 106), insbesondere unter Anwendung der Operation *Partitionierung*.

Kapitel 5

Paste Algorithmus

Sei $T_{BSP} = (V, E, \mathcal{P}, \mathcal{H}, \alpha, \beta, P_\circ)$ ein BSP-Tree bestehend aus konvexen Polyedern. $\{P_1, ..., P_m\}$ sei eine Auswahl von Blattpolyedern. Wir interessieren uns dafür, ob $P = \bigcup_{i=1}^{m} P_i$ ein Polyeder ist. Falls dem so ist, interessieren wir uns weiter für eine Boundary - Representation von P in gleicher Weise, wie wir diese bereits für konvexe Polyeder kennengelernt haben. Diese Umrechnung wird mit dem Verfahren *Paste-Polyeder* durchgeführt, welches ein weiterer Schwerpunkt dieser Arbeit darstellt und welches in diesem Kapitel vorwiegend behandelt wird. Das Verfahren, mit welchem wir die Boundary-Repr in die BSP-Repr umrechnen können, ist vergleichsweise einfacher. Es wird mit Hilfe der Operation Split-Polyeder durchgeführt. Eine genaue Beschreibung dieses Verfahrens finden wir im letzten Abschnitt dieses Kapitels.

Das Verfahren *Paste-Polyeder* gliedert sich in drei Teilverfahren.

1. **Polyederentscheidung**
   Mit diesem Verfahren wird darüber entschieden, ob $P = \bigcup_{i=1}^{m} P_i$ ein Polyeder ist.
   (Laufzeit: $O(n_e \cdot m^2)$)

2. **Begrenzungsflächen**
   In diesem Teilverfahren werden die Bgrenzungsflächen des Polyeders P für die Boundary-Repr bestimmt.
   (Laufzeit: $O(m \cdot x^4 \cdot q^4)$)

3. **Korrespondierende Kanten**
   Zwar hat man im zweiten Teilverfahren mit der Bestimmung der Begrenzungsflächen bereits eine Boundary-Repr für P. Diese entspricht jedoch noch nicht unserer gewünschten Boundary-Repr, in welcher es zu jeder Kante e auch eine ihr entgegengesetzt gerichtete *korrespondierende* Kante $\overline{e}$ existiert. Diesbezüglich wird im dritten Teilverfahren die Boundary-Repr modifiziert.
   (Laufzeit: $O(n_e \cdot log(n_e))$)

Dabei entspricht n_e der Komplexität des Polyeders P, auf welche wir Seite 91 näher eingehen. Der Wert der Variablen x entspricht der Anzahl, wie oft die Polyeder $\{P_1, ..., P_m\}$

miteinander vereinbar sind. Im günstigsten Fall ist jeder dieser Polyeder nur mit konstant vielen anderen Polyedern aus dieser Menge vereinbar. Dann hat die Variable x nur einen konstanten Wert c. Im ungünstigsten Fall ist jedes Polyeder mit jedem anderen vereinbar. Wir erhalten für x also: $(c \leq x \leq m-1)$. Der Wert der Variablen q, soll die durchschnittliche Komplexität eines Begenzungspolygones repräsentieren.

5.1 Polyederentscheidung

Ist $P = \bigcup_{i=1}^{m} P_i$ ein Polyeder?

Wir wollen P nur als Polyeder zulassen, wenn P regulär ist. Betrachten wir dazu nochmal Definition 4.2 für Polyeder bzw. Definition 4.3 für reguläre Polyeder (S. 61). Durch die BSP-Representation ist bereits gewährleistet, daß $\{P_1, ..., P_m\}$ konvexe Polyeder sind. Nach Satz 10 (S. 67) und Satz 12 (S. 68) berühren sich diese Polyeder höchstens in ihren Rändern. Weiterhin ist zu beachten, daß wir mit Definition 4.3 paarweise Strecken- bzw. Punktvereinbarkeit nicht zulassen dürfen. Tritt dies dennoch auf, so wird dies durch die Operation *Vereinbarkeit* (s.S. 78) gemeldet. Schließlich muß der Graph $G = (V, E)$ gemäß Definition 4.2 Seite 61 zusammenhängend sein.

1. $V = \{1, ..., m\}$
2. P_i, P_j *sind flächenvereinbar* $\Leftrightarrow e = \{i, j\} \in E \qquad (1 \leq i, j \leq m)$

Um die Kantenmenge aufzubauen, muß für jedes Polyederpaar $\{P_i, P_j\}$ $(1 \leq i, j \leq m)$ entschieden werden, ob es flächenvereinbar (s.S. 61) ist. Wir gehen dabei nun wie folgt vor.

Den Graphen $G = (V, E)$ repräsentieren wir in m adjazenten (Integer) Listen L_1, ... , L_m. Jede dieser Listen L_i $(1 \leq i \leq m)$ speichere die Indizes derjenigen Polyeder, mit denen der Polyeder P_i (, welches unmittelbaren Zugriff auf L_i haben soll,) flächenvereinbar ist.

Im Rahmen der Operation *Polyederentscheidung* führen wir zunächst für jedes Polyederpaar $P_i \neq P_j$ $(i, j \in V)$ die Operationen *Vereinbarkeit* (siehe Abschnitt 5.1.1) und *Flächenpartitionierung* (siehe Abschnitt 5.2.1) durch. Sind P_i, P_j flächenvereinbar, so liefert die Operation *Vereinbarkeit* die eindeutigen konvexen Polygone f, f_i, f_j. Dabei ist f_i ein Begrenzungspolygon von P_i und f_j ein Begrenzungspolygon von P_j, wobei gilt:

$$f = f_i \cap f_j$$

In diesem Falle wird in die Liste L_i die Nummer j und in die Liste L_j die Nummer i gehängt.

Interessiert man sich lediglich für die Entscheidung, ob P ein Polyeder ist, so kann hier auf die Operation *Flächenpartitionierung* verzichtet werden. Im Rahmen des Gesamtverfahrens *Paste-Polyeder* jedoch nutzt die Operation *Flächenpartitionierung* an dieser Stelle aktuelle Informationen zur Weiterverarbeitung aus. Wir werden in Abschnitt 5.2 auf diese Operation näher eingehen.

Sind alle Paare auf diese Weise bearbeitet, so ist auch der Graph G in den adjazenten Listen L_1, ... , L_m vollständig repräsentiert. Mit dem Algorithmus DFS(G) (*Depth First Search* nach [Meh84] Bd. 2) kann in Zeit $O(m + e)$ entschieden werden, ob G zusammenhängend ist. Dabei ist mit $e \leq m^2$ die Anzahl der Kanten (= Anzahl der flächenvereinbaren Polyeder) von G gemeint.

```
Polyederentscheidung(P_1, ..., P_m) {
    forall {P_i, P_j} do {                                        (1 ≤ i, j ≤ m )
        if(Vereinbarkeit(P_i, P_j)){  // falls P_i, P_j flächenvereinbar
            hänge j nach L_i;         // füllen der adjazenten Listen,
            hänge i nach L_j;         // welche den Graphen G = (V, E) repräsentieren
        }
        Flächenpartitionierung(P_i, P_j);
    }
    DFS(G); // testet den Graphen G auf Zusammenhang
}
```

Für das Beispiel aus Abbildung 4.1 Seite 62 ergibt sich folgender Graph adjazenter Listen.

$$
\begin{array}{lcl}
P_1 \longrightarrow L_1 & : & 2 \rightarrow 3 \rightarrow 4 \dashv \\
P_2 \longrightarrow L_2 & : & 1 \rightarrow 5 \dashv \\
P_3 \longrightarrow L_3 & : & 1 \dashv \\
P_4 \longrightarrow L_4 & : & 1 \rightarrow 7 \dashv \\
P_5 \longrightarrow L_5 & : & 2 \rightarrow 6 \dashv \\
P_6 \longrightarrow L_6 & : & 5 \dashv \\
P_7 \longrightarrow L_7 & : & 4 \dashv
\end{array}
$$

5.1.1 Vereinbarkeit konvexer Polyeder

Seien P_1, P_2 zwei Blätter aus $T_{BSP} = (V, E, \mathcal{P}, \mathcal{H}, \alpha, \beta, P_\circ)$. *Vereinbarkeit(...)* ist die Operation, welche für P_1, P_2 entscheidet, ob ihr Schnitt leer ist, ob sie punktvereinbar, streckenvereinbar oder flächenvereinbar sind. Im letzteren Fall existieren zwei Begrenzungsflächen f_1, f_2 von P_1 bzw. P_2 mit $f = (f_1 \cap f_2)$ ist ein konvexes Polygon. Diese Flächen werden ebenfalls als Ergebnis geliefert. Wir teilen die Operation *Vereinbarkeit* wie folgt in 4 Teilverfahren auf.

$Vereinbarkeit(P_1, P_2)\{$
$\quad Tiefenangleichung(P_1, P_2);$
$\quad Ursprungssuche(L_1, L_2);$
$\quad Objektsuche(L_1, L_2);$
$\quad Cut(O_1, O_2);$
$\}$

Wir wissen, daß P_1, P_2 sich - als Blätter - höchstens in ihren Rändern berühren und konvexe Polyeder sind.

$$P_1 \cap P_2 = B_{P_1} \cap B_{P_2}$$

5.1.1.1 Tiefenangleichung

Lemma 9 (Tiefe)

Sei $Q \neq P_\circ$ ein Polyeder des Baumes T_{BSP} dann gilt:

1. $0 \leq Tiefe(Q) \leq |\mathcal{P}|$
2. *Sei $m := Tiefe(Q)$. Für alle i mit $(0 \leq i \leq Tiefe(Q))$ existieren $m+1$ Polyeder aus T_{BSP}: $Q_0, ..., Q_i, Q_{i+1} ..., Q_m$, welche in dieser Reihenfolge einen Pfad von der Wurzel $P_\circ = Q_0$ bis zum Polyeder $P = P_m$ bilden, und es gilt:*
   (i = Tiefe(Q_i) = Tiefe(Q_{i+1}) + 1)

Sei Tiefe(P_2) $\geq$ Tiefe(P_1), dann existiert nach Lemma 9 ein P_2^* derart, daß $Tiefe(P_2^*) = Tiefe(P_1)$, und es gilt $P_2 \subseteq P_2^*$. Durch sukzessives Bestimmen des Vaterpolyeders kann man von Polyeder P_2 bis zur Tiefe(P_1) zurück aufsteigen. Die dabei durchlaufenen Polyeder und (damit den Pfad) merke man sich in einer Polyederliste L_2. Dann ist P_2^* der zuletzt in dieser Liste angehängte Polyeder mit der Tiefe(P_2^*) = Tiefe(P_1). Analog würden wir operieren, falls gilt Tiefe(P_1) $>$ Tiefe(P_2). Damit sind für die Operation *Tiefenangleichung* also höchstens max{Tiefe(P_1), Tiefe(P_2)} konstante Grundoperationen nötig.

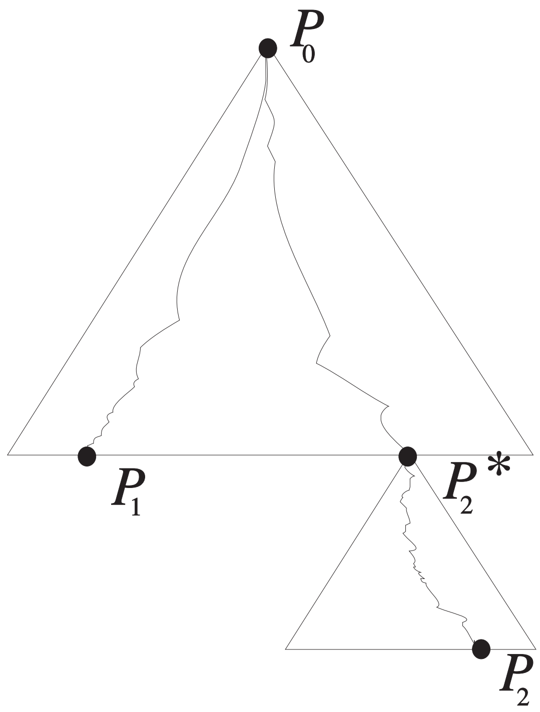

Abbildung 5.1: $P_2 \not\subset P_1$

```
Tiefenangleichung(P1, P2){

  // L1, L2 zwei leere Polyederlisten
  hänge P1 an die Liste L1;
  hänge P2 an die Liste L2;

  if(Tiefe(P2) > Tiefe(P1)){
    //Aufsteigen
    while(Tiefe(tail(L2)) > Tiefe(P1))
            hänge Vater(tail(L2)) nach L2;

  }else{ // (Tiefe(P1) > Tiefe(P2))
    (analoge Bearbeitung wie oben);
  }

  Es gilt nun: Tiefe(tail(L1)) = Tiefe(tail(L2))
}
```

(tail(L) entspricht dem zuletzt an die Liste L angehängten Glied.)

5.1.1.2 Ursprungssuche

Lemma 10

> *Seien T_{BSP}, P_1, $P_2 \in \mathcal{P}$ wie oben, dann existiert ein Polyeder $P \in \mathcal{P}$ mit einer Ebene $H \in \mathcal{H}$ derart, daß $P/H = \{P_+, P_-\}$, und es gilt:*
> **entweder** $P_1 \subseteq P_+$ *und* $P_2 \not\subseteq P_+$ *und* $P_2 \subseteq P_-$ *und* $P_1 \not\subseteq P_-$
> **oder** $P_2 \subseteq P_+$ *und* $P_1 \not\subseteq P_+$ *und* $P_1 \subseteq P_-$ *und* $P_2 \not\subseteq P_-$
>
> *Wir bezeichnen das Polyeder P als* gemeinsamen Vorfahren *von P_1 und P_2.*

Beweis durch Induktion

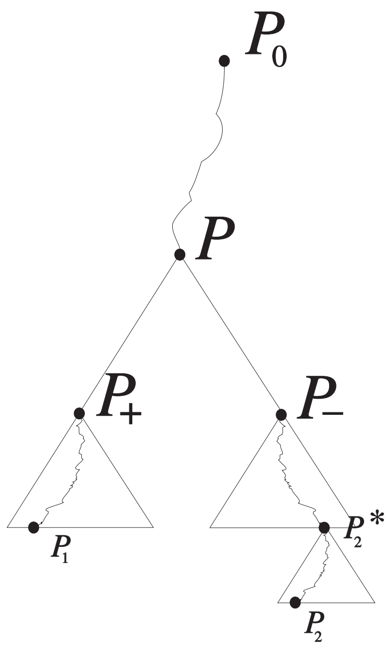

Abbildung 5.2:

Sei o.B.d.A $P_1 \subseteq P_+$ und $P_2 \subseteq P_-$ bzgl.Lemma 10. Wir fahren in unserem Verfahren fort mit der Operation $Ursprungssuche(P_1, P_2)$. Nach Abarbeitung der Operation $Tiefenangleichung(P_1, P_2)$ können wir davon ausgehen, daß Tiefe(tail(L_1)) = Tiefe(tail(L_2)). Man bestimme (analog zur Operation $Tiefenangleichung$) sukzessive und gleichzeitig die Vorfahren von tail(L_1) und tail(L_2) und vergleiche diese miteinander. Dieser Vorgang wird solange durchgeführt, bis zwei Vorfahren gleicher Tiefe einander identisch sind. Dieser Vorfahre entspricht dann eindeutig dem gemeinsamen Vorfahren P. Mit den Operationen $Tiefenangleichung$ und $Ursprungssuche$ wird also die Zeitkomplexität O(max$\{Tiefe(P_1), Tiefe(P_2)\}$) nicht überschritten.

```
Ursprungssuche(L_1, L_2){
    while(tail(L_1) ≠ tail(L_2)){
        hänge Vater(tail(L_1)) nach L_1;
        hänge Vater(tail(L_2)) nach L_2;
    }
    // Es gilt nun: P := tail(L_1) = tail(L_2)
}
```

5.1.1.3 Objektsuche

Falls nun P_1, P_2 flächenvereinbar, dann existieren zwei Begrenzungsflächen f_1 aus P_1 und f_2 aus P_2 derart, daß $f_1 \cap f_2 = P_1 \cap P_2$ ein konvexes Polygon ist. Aus Lemma 10 folgt dann unmittelbar, daß $f_1 \subseteq H$ und $f_2 \subseteq H$. Sei $f_{H_+} = P_+ \cap H$ die Schnittpolygonfläche von P_+ und $f_{H_-} = P_- \cap H$ die Schnittpolygonfläche von P_-. Dann können wir weiter folgern, daß:

$$f_1 \subseteq f_{H_+} \quad (da\ f_1 \subseteq P_1 \cap H \subseteq \underbrace{P_+ \cap H}_{f_{H_+}}) \quad und \quad f_2 \subseteq f_{H_-}$$

Bevor wir mit unserem Verfahren fortfahren, erweitern wir den Splitalgorithmus für konvexe Polyeder wie folgt. Eine Anwendung dieser algorithmischen Erweiterung erfolgt unmittelbar danach.

> Sei P^* ein konvexes Polyeder, welches durch eine Ebene H^* in die Polyeder P_+^* und P_-^* gesplittet wird. Jede Polygonbegrenzungsfläche f von P^* wird wie folgt bearbeitet (Flächen-Vater-Sohn-Beziehung) :
>
> 1. $f \subseteq P_+^*$
> f_+ sei die zu f kongruente gleichorientierte Polygonbegrenzungsfläche von P_+^*. Diese Flächen werden dann zusätzlich wie folgt miteinander verpointert:
>   Plus_Sohn(f) = f_+; Minus_Sohn(f) = NULL;
>   Vater(f_+) = f;
> 2. $f \cap H_+ \neq \emptyset$ und $f \cap H_- \neq \emptyset$ $\qquad (f/H = \{f_+, f_-\})$
>   Dann existieren 2 Polygonbegrenzungsflächen $f_+ \subseteq P_+^*$ und $f_- \subseteq P_-^*$. Diese Flächen werden dann zusätzlich wie folgt miteinander verpointert:
>   Plus_Sohn(f) = f_+; Minus_Sohn(f) = f_-;
>   Vater(f_+) = f; Vater(f_-) = f;
> 3. $f \subseteq P_-^*$
>   (Bearbeitung erfolgt analog zu Fall 1)
>
> Analog verfahren wir mit Knoten und Kanten.

Lemma 11

Sei T_{BSP} der BSP-Tree wie oben. Seien $P_2 \subseteq P_1$ zwei Polyeder des Baumes. f_1 sei Begrenzungspolygon von P_1. f_2 sei Begrenzungspolygon von P_2.
Weiterhin gilt: $f_2 \subseteq f_1$. Dann gilt:

1. $\exists\, !\, H \quad .^{\cdot}. \quad f_2 \subseteq f_1 \subseteq H$ *und* $f_1 = P_1 \cap H$ *und* $f_2 = P_2 \cap H$

2. *Nach Satz 10 (S. 67) existiert ein Pfad in T_{BSP} von P_1 nach P_2.*
   Seien P_i P_{i+1} 2 Polyeder dieses Pfades, wobei gilt: P_i = Vater(P_{i+1}).
   Dann existieren 2 Flächen $f_2 \subseteq f_i \subseteq f_{i+1} \subseteq f_1$ für die gilt: Vater(f_{i+1}) = f_i.
   Dabei ist f_i Begrenzungsfläche von P_i und f_{i+1} ist Begrenzungsfläche von P_{i+1}.

Demnach ist dieser Pfad von f_2 beginnend durch sukzessives Bestimmen des Flächen-Vater-Polygones bis zur Fläche f_1 eindeutig rekonstruierbar. Wir bezeichnen diesen Pfad auch als **Flächenpfad** *von f_1 nach f_2.*

Nach obiger Erweiterung des Splitalgorithmus, ist jede Begrenzungsfläche f die Wurzel eines BSP-Tree's für Polygone, welcher sich lediglich dadurch von der Def. des BSP-Tree's unterscheidet, daß eine Fläche keinen, **einen** oder 2 Söhne haben kann.

Analoges bzgl. Lemma 11 gilt auch für Knoten und Kanten. Wir fahren mit unserem Verfahren fort. In L_1 haben wir den Polyeder-Pfad von P_+ nach P_1 gespeichert. Somit haben wir auch unmittelbaren Zugriff auf die Fläche f_{H_+} (s. S. 65). Unbekannt ist noch die Fläche $f_1 \subseteq f_{H_+}$. Existiert diese Fläche f_1, so muß nach Lemma 11 ein zum Pfad L_1 identischer Flächenpfad von f_{H_+} nach f_1 existieren. Dazu steigen wir den Pfad L_1 von P_+ beginnend in Richtung P_1 ab und kontrollieren dabei die Existenz des Flächenpfades mit Hilfe der Flächen - Vater - Sohn - Beziehung. Bricht die Flächenverbindung während des Pfadabstieges ab, so sind P_1, P_2 nicht flächenvereinbar. Jedoch können diese noch punkt- bzw. streckenvereinbar sein. Denn P_1 kann P_2 noch mit einer Kante oder mit einem Knoten berühren. Ihre Existenz läßt sich analog zur Suche nach der Fläche f_1 feststellen. Finden wir kein solches Objekt (Knoten, Kante, Fläche), so sind P_1 P_2 nicht vereinbar, und wir können das Verfahren an dieser Stelle abbrechen. Andernfalls merken wir uns das gefundene Objekt (Knoten, Kante, Fläche) in der Variablen O_1. Analog verfahren wir mit dem Polyeder P_2 und merken uns, falls wir auch hier nicht das Verfahren vorzeitig abbrechen, das Objekt O_2. In diesem Fall liefert die Operation *Objektsuche* die beiden Objekte O_1 und O_2, und wir fahren mit der Operation *Cut* fort.

5.1.1.4 Cut

Wir schneiden die beiden Objekte O_1, O_2 miteinander und erhalten (s. dazu auch ab S. 20 bzw. ab S. 30):

1.Fall $O_1 \cap O_2 = \emptyset \quad \Rightarrow \quad P_1 \cap P_2 = \emptyset$
$\Rightarrow$ P_1,P_2 sind unvereinbar

2.Fall $O_1 \cap O_2 = \{S\} \quad \Rightarrow \quad P_1 \cap P_2 = \{S\}$
$\Rightarrow$ P_1,P_2 sind punktvereinbar

3.Fall $O_1 \cap O_2 = e$ eine Strecke $\quad \Rightarrow \quad P_1 \cap P_2 = e$
$\Rightarrow$ P_1,P_2 sind streckenvereinbar

4.Fall $O_1 \cap O_2 = f$ eine konvexe Polygonfläche $\quad \Rightarrow \quad P_1 \cap P_2 = f$
$\Rightarrow$ P_1,P_2 sind flächenvereinbar

Laufzeitanalyse

Die Operation *Vereinbarkeit*, angewandt auf die beiden Polyeder P_1, P_2, überschreitet, in Bezug auf die Operation *Polyederentscheidung* (S. 76), nicht eine Zeitkomplexität von:

$$O(n_e)$$

Dabei ist n_e die Komplexität von P (S. 76), gemessen in der Anzahl seiner Kanten (s. S. 91). Wir wollen davon ausgehen, daß der Baum $T_{BSP} = (V, E, \mathcal{P}, \mathcal{H}, \alpha, \beta, P_\circ)$ effizient in Bezug auf das Polyeder P konstruiert ist. Dies erreicht man dadurch, indem möglichst nur solche Schnittebenen H verwendet werden, für die eine Begrenzungsfläche f des Polyeders P existiert derart, daß gilt: $f \subseteq (P \cap H)$. Damit wäre die Zahl der Schnittebenen in $\mathcal{H}$ auf $O(n_f)$ begrenzt, wobei n_f die Anzahl der Begrenzungsflächen des Polyeders P ist.

> In Abschnitt 5.4 (S. 100) werden wir ein Verfahren kennenlernen, welches diesbezüglich automatisch einen BSP-Tree zur Repräsentation eines Polyeders P konstruiert. Gleichzeitig beweist dieses Verfahren, daß wir dazu mit n_f Schnittebenen auskommen.

Die Komplexität der Länge eines Pfades in T_{BSP}, kann somit $O(m)$ nicht überschreiten. Da $(n_e > n_f)$ (s. S.91) haben wir schließlich in $O(n_e)$ eine obere Komplexitätsschranke für die maximale Tiefe des Baumes. Da die Operationen *Tiefenangleichung(...)*, *Ursprungssuche(...)* und *Objektsuche(...)* eine Laufzeit von $O(\max\{\text{Tiefe}(P_1), \text{Tiefe}(P_2)\}$ benötigen, überschreiten diese ebenfalls nicht die Komplexität von $O(n_e)$. Seien n_1, n_2

die Komplexitäten der Polyeder P_1, P_2 mit $n_1 \geq n_2$ (o.B.d.A.), so überschreitet obige Operation *Cut* nicht die Zeitkomplexität $O(n_1)$ und somit auch nicht die Zeitkomplexität $O(n_e)$.

5.2 Begrenzungsflächen

Haben wir festgestellt, daß $P = \bigcup_{i=1}^{m} P_i$ ein Polyeder ist, so ist unser nächstes Ziel, die Boundary-Representation für P zu berechnen — gemäß der Boundary-Repr für konvexe Polyeder. Mit Hilfe der nachfolgenden Definition wollen wir den Rand B_P von P analysieren.

Definition 5.1 (Berührflächen)

1. $I\!F$:
   Menge aller Begrenzungsflächen aus der regulären Boundary-Repr (s.S. 28) der konvexen Polyeder $\{P_1, ..., P_m\}$. Wir teilen $I\!F$ in die beiden zueinander komplementären Mengen $I\!F^*$ und $\overline{I\!F^*}$.

   $$I\!F := I\!F^* \cup \overline{I\!F^*}$$

2. $I\!F^*$:
   Menge aller Begrenzungsflächen, in welchen Polyeder aus $\{P_1, ..., P_m\}$ flächenvereinbar sind.

   $$\begin{array}{llll} I\!F^* := \{ & f \in I\!F & : & \exists\, f' \in I\!F \; derart, daß \\ & & & \bullet \;\; sei\; P_f \; der \; zu \; f \; zugehörige \; Polyeder \\ & & & \bullet \;\; sei\; P_{f'} \; der \; zu \; f' \; zugehörige \; Polyeder \\ & & & \\ & & & dann \; soll \; gelten: \\ & & & P_f \; und \; P_{f'} \; sind \; in \; den \; Flächen \; f \; und \; f' \; flächenvereinbar. \\ & & & ((f \cap f') \; ist \; ein \; konvexes \; Polygon) \\ & & & \\ & & & Wir \; sagen: \; „f \; wird \; durch \; f' \; berührt“ \\ \} & & & \end{array}$$

3. $\overline{I\!F^*}$:
   Menge aller Begrenzungsflächen, in welchen Polyeder aus $\{P_1, ..., P_m\}$ nicht flächenvereinbar sind.

4. *Sei $f \in I\!F^*$*

(a) $\mathbf{M_f}$*:*
Menge der Begrenzungsflächen aus $I\!F^*$*, durch welche die Fläche* $f \in I\!F^*$ *berührt wird. Weiterhin gilt, daß die Flächen aus* M_f *untereinander, bis auf ihre Ränder, disjunkt sind. (s. Abschnitt 5.1.1 S. 78)*

(b) $\mathbf{T_f}$*:*
Menge aller konvexer Polygone, in welchen sich die Polyeder aus $\{P_1, ..., P_m\}$ *und somit die Flächen aus* M_f *in der Fläche* f *schneiden.*

$$T_f := \{f' \subseteq f \ : \ \exists\, f_i \in M_f \ \therefore \ f' = (f_i \cap f)\}$$

(c) $\mathbf{T_f^*}$*:*
Menge aller Flächen aus T_f *— ohne ihre Ränder.*

$$T_f^* := \{\ f^* \subseteq I\!R^3 \ : \ \exists\, f' \in T_f \ mit\ f^* = f' \setminus B_{f'}\}$$

Nach dieser Definition setzt sich der Rand B_P des Polyeders P wie folgt zusammen.

$$\begin{aligned} B_P &= \left(\bigcup_{f \in \overline{\mathbb{F}^*}} f\right) \cup \left(\bigcup_{f \in \mathbb{F}^*} f\right) \setminus C \quad \subseteq \mathbb{R}^3 \\ C &:= \bigcup_{f \in D} f \\ D &:= \left\{\left(\bigcap_{f^* \in T_f^*} f^*\right) \ : \ f \in \mathbb{F}^*\right\} \end{aligned}$$

Wir wollen B_P durch konvexe Polygone beschreiben. Die Flächen aus $\overline{f} \in \overline{\mathbb{F}^*}$ sind konvexe Polygone, die uns bereits in Form von Knotenordnungen vorliegen. Wie aber beschreiben wir die Teilfläche $\left(f \setminus (\bigcup_{f^* \in T_f^*} f^*)\right)$ einer Fläche $f \in \mathbb{F}^*$? Dazu partitionieren wir f in eine Menge konvexer Polygone, welche wir mit $Part_f$ bezeichnen.

$$Part_f = Sub_{T_f} \cup \overline{Sub_{T_f}}$$

Dabei sei Sub_{T_f} die Menge derjenigen konvexen Polygone, deren Vereinigung genau der Vereinigung der konvexen Polygone aus T_f entspricht. (Vereinigung von Punktmengen !)

$$\bigcup_{f^* \in Sub_{T_f}} f^* = \bigcup_{f' \in T_f} f'$$

Im folgenden Unterabschnitt *Flächenpartitionierung* wird gezeigt, wie wir für jede Fläche $f \in \mathbb{F}^*$ eine solche Partitionierung erhalten.

Insgesamt gehen wir nun wie folgt vor:

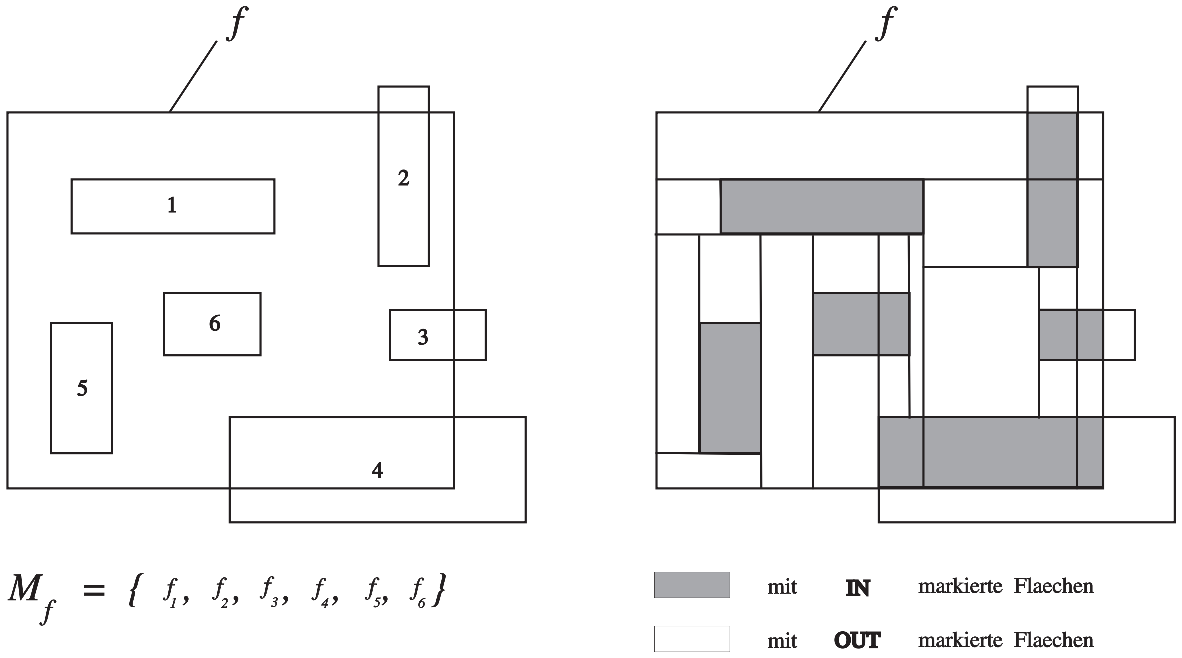

Abbildung 5.3: Partitionierung der Fläche f

1. Markiere alle Flächen $f \in \mathbb{F}$ mit OUT.

2. In der Operation *Polyederentscheidung* werden alle Polyederpaare $\{P_i, P_j\}$ aus $\{P_1, ..., P_m\}$ durchlaufen und mit der Operation *Vereinbarkeit* verarbeitet. Sind P_i, P_j in den Flächen f_i, f_j flächenvereinbar, so sollen diese nun mit der Operation *Flächenpartitionierung* weiterverarbeitet werden, durch welche die Flächen f_i, f_j partitioniert beziehungsweise weiter partitioniert werden. Dazu mehr im nächsten Unterabschnitt Flächenpartitionierung. Sind alle Polyederpaare auf diese Weise verarbeitet, so sind auch alle Flächen aus $f \in \mathbb{F}^*$ partitioniert wie oben gefordert. Dabei sind die Flächen aus Sub_{T_f} mit **IN** und die Flächen $\overline{Sub_{T_f}}$ mit **OUT** gekennzeichnet.

3. Der Rand B_P wird dann durch die Vereinigung aller mit OUT markierten Polygonflächen repräsentiert. Dabei sind die, die Polygone repräsentierenden Knotenordnungen derart orientiert, daß ihre Normalen aus dem Polyeder P herauszeigen. (vgl. Satz 7 Seite 14) Wir fassen diese konvexen Polygonflächen in der Liste L_{face} zusammen.

5.2.1 Flächenpartitionierung

Sei $f \in \mathbb{F}^*$ eine Fläche mit der Menge ihrer Berührungsflächen $M_f = \{f_1, ..., f_k\}$. Alle Flächenpaare $f_i, f_j \in \mathbb{F}^*$ deren Polyeder $P_i, P_j \in \{P_1, ..., P_m\}$ in diesen beiden Flächen flächenvereinbar sind, durchlaufen die Operation $Flächenpartitionierung(f_i, f_j, f_{ij})$ genau einmal. Dabei sind die Flächen f_i, f_j und das Schnittpolygon $f_{ij} = f_i \cap f_j$ bereits in der Operation *Flächenvereinbarkeit* berechnet worden. Demnach durchlaufen auch alle Flächenpaare $\{f, f_1\}, ..., \{f, f_k\}$ die Operation *Flächenpartitionierung* in dieser Reihenfolge (o.B.d.A.) genau einmal.

Sei $T_{BSP}(f)$ der BSP-Tree (konvexer Polygone), welcher f als Wurzel hat. Zunächst sei f einziges Polygon dieses Baumes. In diesen Baum lassen wir nun sukzessive die Polygone $f_1, ..., f_k$ mit der Operation *Partitionierung sickern*. (s. dazu auch S. 71 ff.)

Danach gilt:

$Part_f$:
Menge der Blätter des Baumes $T_{BSP}(f)$. Damit stellt $Part_f$ eine Partitionierung der Fläche f in konvexe Teilpolygone dar.

$$Part_t = \underbrace{\{\textbf{IN}\ markierte\ Bl\ddot{a}tter\}}_{Sub_{T_f}} \cup \underbrace{\{\textbf{OUT}\ markierte\ Bl\ddot{a}tter\}}_{\overline{Sub_{T_f}}}$$

5.2.2 Laufzeitanalyse zur Flächenpartitionierung

Jede Fläche $f \in \mathbb{F}^*$ wird im Verfahren der Flächenpartitionierung durch die sie berührenden Flächen $M_f = f_1, ..., f_k$ in konvexe Teilflächen partitioniert. Wir tun dies durch Konstruktion eines BSP-Tree's $T_{BSP}(f)$ mit f als Wurzel, indem wir nach und nach eine Fläche aus M_f in den (anfangs aus f selbst bestehenden) $T_{BSP}(f)$ mit der Operation *Partitionierung einsickern* lassen. (s. S. 71 ff.)

5.2.2.1 Komplexität eines $T_{BSP}(f)$

In [Ed87] finden wir folgenden Satz.

Satz 14

Seien n Geraden $g_1, ..., g_n \subseteq \mathbb{R}^2$ gegeben. Seien n_s die Anzahl der Partitionen, in welche diese Geraden den Raum $\mathbb{R}^2$ partitionieren, dann gilt:

$$n_s \leq n^2$$

$f_1^*, ..., f_u^*$ seien die konvexen Polygone, welche eine Partitionierung von f bilden und gleichzeitig die Blätter des $T_{BSP}(f)$ darstellen. Wir verlängern nun in Gedanken alle Kanten dieser Blattpolygone zu Geraden und zählen diese in der Variablen n_g.

Aufgrund dessen, daß $f_1^*, ..., f_u^*$ durch Sicker- und Partitioniervorgänge der Polygone $f_1, ..., f_k$ als Blätter in $T_{BSP}(f)$ entstanden sind gilt:

$$n_g \leq \text{Anzahl der Kanten der Polygone}\{f, f_1, ..., f_k\}$$

Mit Satz 14 gilt dann:

$$u \leq {n_g}^2 \leq (\text{Anzahl der Kanten der Polygone}\{f, f_1, ..., f_k\})^2$$

Schätzen wir die Komplexität eines Polygones durchschnittlich mit q ab, so erhalten wir:

$$u \leq {n_g}^2 \leq (q \cdot (k+1))^2$$

Demnach erhalten wir für die Anzahl der Polygone in $T_{BSP}(f)$ und damit für die Komplexität von $T_{BSP}(f)$:

$$O((q \cdot k)^2)$$

5.2.2.2 Konstruktionszeit von $T_{BSP}(f)$

Die Laufzeit zur Konstruktion des $T_{BSP}(f)$ setzt sich zusammen aus der Summe der Laufzeiten der Sicker- und Partitionieroperationen, welche die Polygone $f_1, ..., f_k$ verursachen.
Sei $f_i \in \{f_1, ..., f_k\}$.
Sei $T^*_{BSP}(f)$ der BSP-Tree von f, (unmittelbar) bevor wir f_i in den Baum *sickern* haben lassen.

1. Wir lassen f_i in den Baum T^*_{BSP} sickern. f_i kann durch den Sickervorgang in $O((q \cdot k)^2)$ viele Polygone zerlegt werden, da $T^*_{BSP}(f)$ aus $O((q \cdot k)^2)$ vielen Polygonen (Baumknoten) besteht. Eine einfache Zerlegung (Polygonsplit durch eine Gerade) kostet $O($ Durchschnittskomplexität eines Polygones $)$ = *O(q)*. (Man bedenke! Entstehen zwei neue Teilpolygone, so haben diese zusammen die Komplexität ihres Vaterpolygones). Wir erhalten somit eine Sickerlaufzeit von:

$$O(q \cdot (q \cdot k)^2)$$

2. Sind Partitionen von f_i, durch den Baum $T^*_{BSP}(f)$ in dessen Blätter gesickert, so werden die Blätter durch diese Teilpartitionen von f_i weiter partitioniert, so daß $T^*_{BSP}(f)$ erweitert wird. Dies sind, wie oben bereits erwähnt, $O((q \cdot k)^2)$ Partitionen von f_i. Sei $f^* \subseteq f_i$ eine solche Teilpartition und sei f^* in das Blattpolygon f_{Blatt} gesickert. Für beide (f^*, f_{Blatt}) berechnen wir je eine Durchschnittskomplexität von q. Dann erhalten wir für den Partitioniervorgang (s. S.4.3.2) eine Laufzeit von: q^2
Für alle Partitioniervorgänge bzgl. f_i im Baum $T^*_{BSP}(f)$ brauchen wir dann ein Laufzeit von:

$$O(q^2 \cdot (q \cdot k)^2)$$

Beziehen wir schließlich alle Flächen $M_f = \{f_1, ..., f_k\}$ zur Partitionierung der Fläche f mit ein, so erhalten wir zur Konstruktion des $T_{BSP}(f)$ eine Zeitkomplexität von:

$$O(\underbrace{k}_{(f_1,...,f_k)} \cdot q^2 \cdot (q \cdot k)^2)$$

5.2.2.3 Zeitkomplexität der gesamten Flächenpartitionierung

Sei $t := |\mathbb{F}^*|$ (Anzahl der Polyederverbindungsstellen), so entstehen im Verfahren der Flächenpartitionierung t viele T_{BSP} Bäume. Wir erhalten somit für die Gesamtlaufzeit dieses Verfahrens eine Zeitkomplexität von :

$$O(t \cdot (k^3 \cdot q^4))$$

Wir wollen dieses Ergebnis noch etwas präzisieren.

1. Günstigenfalls sind die Polyeder $\mathcal{P}^* = \{P_1, ..., P_m\}$ derart miteinander flächenvereinbar, daß für alle Polyeder $P_i \in \mathcal{P}^*$ gilt:
   P_i ist mit höchstens c (konstant) vielen Polyedern aus $\mathcal{P}^*$ flächenvereinbar, wobei:
   $$c << m$$
2. Ungünstigenfalls – was sehr unwahrscheinlich ist – ist in $\mathcal{P}^*$ jedes Polyeder mit jedem flächenvereinbar.

Damit erhalten wir folgende obere und untere Schranken für k und t:

$$\begin{array}{rcl} c \leq & k & \leq m-1 = O(m) \\ m \cdot c \leq & t & \leq \frac{1}{2} \cdot (m^2 - m) = O(m^2) \end{array}$$

Wir erhalten somit eine feiner analysierte Gesamtkomplexität von:

$$\Omega(m \cdot c^4 \cdot q^4) \leq t \cdot (k^3 \cdot q^4) \leq O(m^5 \cdot q^4)$$

5.3 Korrespondierende Kanten

Im vorigen Teilabschnitt unseres Algorithmus zur Vereinigung der Polyeder $\{P_1, ..., P_m\}$ zu dem Polyeder P haben wir in der Liste L_{face} die Menge der das Polyeder P begrenzenden Flächen (konvexe Polygone) zusammengefaßt. Diese repräsentieren zwar den Rand B_P von P, entsprechen jedoch noch nicht ganz dem Datenmodell für konvexe Polyeder. Denn nicht jede Kante e besitzt eindeutig eine ihr korrespondierende Kante $\overline{e}$ Demgemäß muß die vorliegende Boundary-Repr modifiziert werden.

Dazu betrachten wir uns folgendes motivierende Beispiel. Seien $f, f_1, f_2, f_3 \in L_{face}$ benachbarte Flächen wie in Abbildung 5.4. Sie werden repräsentiert durch folgende Knotenordnungen:

$$f_1 = (1,2,3,4) \quad f_2 = (2,4,5,6) \quad f_3 = (5,7,3,6)$$

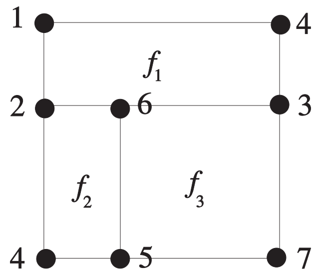

Abbildung 5.4:

Hierbei kann den Kanten e_2 =(6,2) von f_2 und e_3 = (3,6) von f_3 nur indirekt die Kante $e_1 = (2,3)$ von f_1 als korrespondierende Kante zugewiesen werden. Die Kante e_1 müßte also in die zu e_1 und e_2 korrespondierenden Kanten $\overline{e_2}$ = (2,6) und $\overline{e_3}$ = (6,3) partitioniert werden.

Um diesbezüglich eine Repräsentation für das Polyeder P aus ihren es begrenzenden Flächen L_{face} zu gewinnen, gehen wir wie folgt in 4 Teilschritten vor.

Im **ersten** Teilschritt erzeugen wir die Knotenliste L_{vertex} des Polyeders P und modifizieren diesbezüglich die Flächenliste L_{face}.
(Laufzeit: $O(n_e \cdot log(n_e))$)

Im **zweiten** Teilschritt werden alle Kanten der Flächenliste L_{face} zunächst in der Liste L_{edge} gesammelt. Anschließend werden alle Kanten nach Geraden sortiert, welche durch die Kanten eindeutig repräsentiert werden. Die Geraden werden in einem Feld F_g gesammelt, welche je ein Feld von allen Knoten $v \in L_{vertex}$ zugeteilt bekommen, die auf der Geraden liegen.
(Laufzeit: $O(n_e \cdot log(n_e))$)

Im **dritten** Teilschritt werden die Knoten der Knotenfelder aller Geraden nach der *Richtung* der Geraden sortiert.
(Laufzeit: $O(n_e \cdot log(n_e))$)

Im **vierten** Teilschritt werden mit Hilfe der sortierten Knotenfelder der Geraden die noch fehlenden Knoten auf die Kanten propagiert.
(Laufzeit: $O(n_e \cdot log(n_e))$)

Insgesamt verbraucht man also für das Teilverfahren des Paste-Algorithmus zur Propagierung der korrespondierenden Kanten, eine Laufzeit von.

$$O(n_e \cdot log(n_e))$$

Anmerkung zur Komplexität des Polyeders P

Gemäß Abschnitt 2.4 Seite 29 kann man sich leicht klarmachen, daß für die Anzahl der Knoten (n_v), der Kanten (n_e^*) und der Flächen n_f bzgl. der Flächenliste L_{face} gilt:

$$(n_v < n_e^*) \wedge (n_f < n_e^*)$$

Da die Flächen der Liste L_{face} nur noch um korrespondierende Kanten erweitert werden sollen, bleibt die Anzahl der Knoten n_v und der Flächen n_f konstant. Mit n_e – der Anzahl der Kanten der noch zu gewinnenden Datenrepräsentation von P – gilt dann :

$$n_e^* \leq n_e$$

Deshalb schätzen wir die Laufzeit des Algorithmus zur Knotenpropagierung mit $\mathbf{n_e}$ (Anzahl der Kanten Begrenzungsrepräsentation von P) ab.

5.3.1 Flächenliste und Knotenliste von P

Da alle Begrenzungsflächen von P in L_{face} vorhanden sind, sind auch alle Knoten des Randes von P bereits kreiert. Sei L_{vertex} eine zunächst leere Knotenliste. Wir besuchen alle Flächen der Liste L_{face} und für jede dieser Flächen alle ihre Knoten. Diese Knoten sammeln wir in der Liste L_{vertex}. Damit enthält L_{vertex} alle Knoten der Randbeschreibung von P. (Laufzeit O(n_e)) Weil ein Knoten mindestens von 3 Flächen umgeben ist , die sich diesen Knoten teilen, kann man davon ausgehen, daß in L_{vertex} Knoten, welche denselben geometrischen Punkt repräsentieren, mehrfach vorkommen. Bevor wir die Liste L_{vertex} weiter diesbezüglich modifizieren, soll folgendes sowohl für

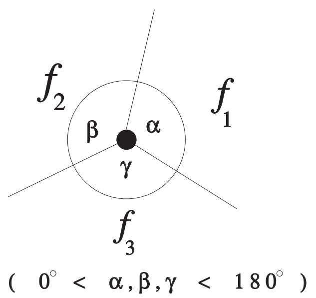

Abbildung 5.5: Ein Knoten ist mindestens durch 3 Flächen begrenzt.

Knoten der Polyeder des T_{BSP} (s. S. 75) als auch für die Knoten der Polygone der BSP-Tree's der Berührungsflächen (s. S. 86 Abschnitt 5.2.1) gelten:

> Knoten gleicher geometrischer Lage sollen gleiche Globalidentität (bei der Konstruktion der BSP-Trees) zugeordnet bekommen.
>
> Man führt dazu einen globalen Knotenzähler *glob_count* ein, welcher vor Entstehung von T_{BSP} und seiner Polyeder auf 0 gesetzt wird. Sei weiterhin *glob_id* eine Strukturvariable eines Knotens v
>
> 1. Für jeden neu entstehenden Knoten v führen wir folgende beiden Anweisungen durch:
> $v \rightarrow glob\_id = glob\_count + 1;$
> $glob\_count = glob\_count + 1;$
> 2. Wird beim Split ein Knoten $v \in H_+$ zu einem Knoten v_+ durchpropagiert, so führen wir folgende Anweisung durch.
> $v_+ \rightarrow glob\_id = v \rightarrow glob\_id;$
>   (analog $v \in H_-$)

Um das Mehrfachvorkommen Knoten gleicher Globalidentität auszuschließen, sortieren wir die Knoten der Liste L_{vertex} nach ihrer Globalidentität in $O(n_e \cdot log(n_e))$. Für jede Gruppe *Glob_Id* von Knoten gleicher Globalidentität machen wir folgendes. Jedem Knoten v, der Gruppe *Glob_Id* wird derselbe willkürlich aus dieser Gruppe stammender Knoten $v_\circ$ zugeordnet. Anschließend werden alle Knoten dieser Gruppe, außer dem auserwählten Knoten $v_\circ$, aus der Liste L_{vertex} gestrichen, ohne ihren Speicher freizugeben. Damit kommen in L_{vertex} keine Knoten mehr mehrfach vor. (Laufzeit: O(n_e)) Schließlich müssen wir dieses Mehrfachvorkommen von Knoten gleicher Globalidentität auch in den Flächen von L_{face} beseitigen. Wir besuchen dazu jeden Knoten v jeder Fläche f der Liste L_{face} und tauschen den Knoten v gegen den auserwählten Knoten $v_\circ$ der jeweiligen Gruppe und geben den Speicher von v frei. (Laufzeit: O(n_e))

5.3.2 Kantenliste und Geradenliste von P

Zunächst werden, indem alle Flächen der Flächenliste durchlaufen werden, alle Kanten der Flächen erzeugt. Pro Fläche f werden die zur Fläche f zugehörigen Kanten in Kantenordnung in einer lokalen Kantenliste gesammelt, worauf f direkten Zugriff hat. Zusätzlich wird eine globale Kantenliste L_{edge} der Kanten aller Flächen aus L_{face} erzeugt. (Laufzeit: O(n_e)) Wir wollen nun die Kanten der Liste L_{edge} derart sortieren, daß Kanten, welche die gleiche Gerade repräsentieren, nebeneinander liegen. (Laufzeit: *O($n_e log(n_e)$))*) Dazu benötigen wir eine Ordnung auf den Geraden $g \in \mathbb{R}^3$ und darauf basierend eine Vergleichsoperation *comp_edge(e_1, e_2)*, welche für 2 Kanten e_1, e_2 der Liste L_{edge} in konstanter Zeit folgende Fallunterscheidung berechnet:

$$1.Fall \quad g_{e_1} \;<\; g_{e_2}$$

$$\begin{array}{llll} 2.Fall & g_{e_1} & = & g_{e_2} \\ 3.Fall & g_{e_1} & > & g_{e_2} \end{array}$$

Dabei sind g_{e_1}, g_{e_2} die Geraden, welche durch die Kanten e_1, e_2 eindeutig repräsentiert werden.

Anschließend wird für jedes Kantenbündel der sortierten Liste L_{edge}, bestehend aus Kanten, welche die gleiche Gerade g repräsentieren, die Gerade g erzeugt. Gleichzeitig wird dafür gesorgt, daß alle Kanten eines Bündels auf ihre neu erzeugte Gerade einen Verweis bekommen. Zusätzlich wird allen Geraden g ein Knotenfeld zugewiesen. Dieses Feld enthält die Knoten der Kanten des Kantenbündels, woraus g entstanden ist. Die Größe des Feldes entspricht der Anzahl der Knoten des zugehörigen Kantenbündels. Die Geraden selbst speichern wir in einem Geradenfeld F_g, wobei sich jede Gerade ihre spezifische Feldnummer merkt. (s. Abbildung 5.6 (S. 94))
(Laufzeit: $O(n_e)$)

> **Bemerkung**
> Die Benutzung der Knotenfelder bzw. des Geradenfeldes ermöglicht in einem späteren Teil des Verfahrens einen Zugriff auf die Knoten in konstanter Zeit.
>
> Sei g eine Gerade mit dem Feldindex i, so wollen wir g mit g_i bezeichnen. Weiterhin bezeichnen wir mit $E_{g_i} \subseteq L_{edge}$ die Menge der Kanten, welche die Gerade g_i repräsentieren und auf welche die Kanten aus E_{g_i} direkten Zugriff haben.
>
> Das Knotenfeld der Geraden g_i wollen wir mit $F_{v_{g_i}}$ im weiteren Verlauf unseres Verfahrens bezeichnen.

5.3.2.1 Geradenordnung und Vergleichsoperation

Wir wollen eine Ordnung für alle Geraden $g \subseteq \mathbb{R}^3$ festlegen.
Dies geschieht mit Hilfe von 3 Ebenenpaaren
$\{ \{H_1, H_2\}, \{H_3, H_4\}, \{H_5, H_6\} \}$, welche wie folgt definiert sind:

1. $H_1 : (\vec{e}_y;\ 0)$ und $H_2 : (\vec{e}_y;\ 1)$
2. $H_3 : (\vec{e}_x;\ 0)$ und $H_4 : (\vec{e}_x;\ 1)$
3. $H_5 : (\vec{e}_z;\ 0)$ und $H_6 : (\vec{e}_z;\ 1)$

Die Vektoren $\vec{e}_x = (1,0,0)$, $\vec{e}_y = (0,1,0)$, $\vec{e}_z = (0,0,1)$ sind die Einheitsvektoren des zugrundeliegenden Weltsystems (Rechtssystem) und sind hier Normalenvektoren zu

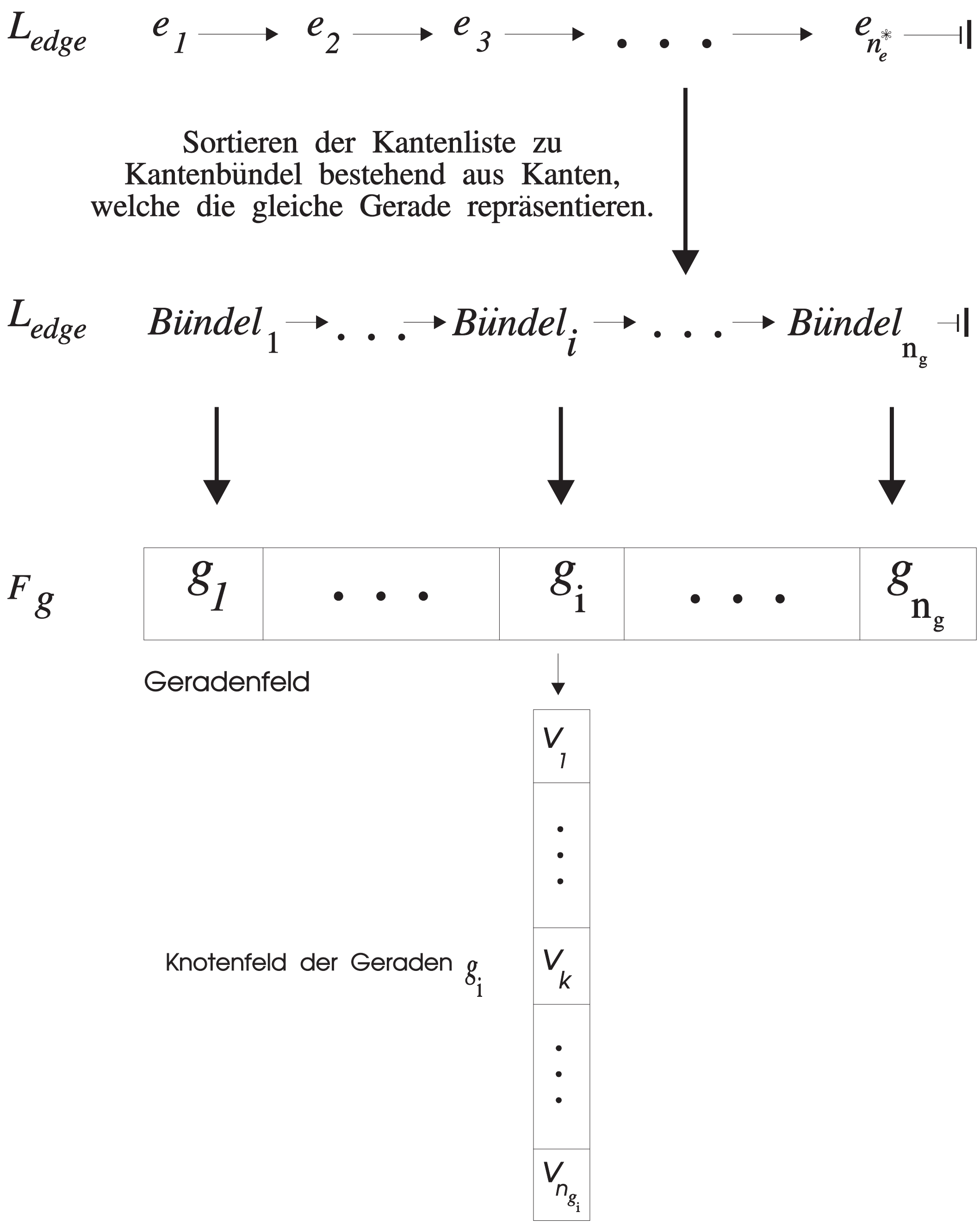

Abbildung 5.6: Datenstruktur zur Knotenpropagierung

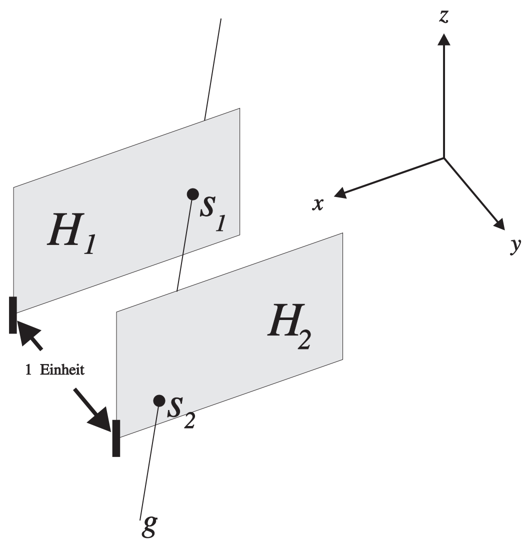

Abbildung 5.7:

obigen Ebenen. Mit den Abstandsskalaren 0 bzw. 1 haben die Ebenen pro Paar einen Abstand von einer Einheit zueinander. Dabei liegt das erste Ebenenpaar parallel zur x-z-Ebene, das zweite Ebenenpaar parallel zur z-y-Ebene und das dritte Ebenenpaar parallel zur x-y-Ebene. (s.Abbildung 5.7 (S.95)) Für jede Kante e der Kantenliste L_{edge} führen wir dann folgende Operation *geraden_id* durch.

$geraden\_id(e)\{$

1. Schneidet die Gerade g_e, welche die Kante $e = (v_1, v_2)$ repräsentiert, die Ebene H_1 eindeutig in dem Schnittpunkt $s_1 = (x_1, 0, z_1)$, so schneidet g_e auch die Ebene H_2 eindeutig in einem von s_1 verschiedenen Schnittpunkt $s_2 = (x_2, 1, z_2)$. Für e merken wir uns dann den Zahlenvektor (1, x_1, z_1, x_2, z_2) und verlassen die Funktion.

2. Falls $g_e \parallel H_1$ und schneidet g_e das Ebenenpaar (H_3, H_4) in $s_1 = (0, y_1, z_1)$ bzw. in $s_2 = (1, y_2, z_2)$, so merken wir uns den Zahlenvektor (2, y_1, z_1, y_2, z_2) für e und verlassen die Funktion.

3. Falls $g_e \parallel H_1$ und falls $g_e \parallel H_3$, dann schneidet g_e das Ebenenpaar (H_5, H_6) in $s_1 = (x_1, y_1, 0)$ bzw. in $s_2 = (x_2, y_2, 1)$, und wir merken uns den Zahlenvektor (3, x_1, y_1, x_2, y_2) für e und verlassen die Funktion.

$\}$

Damit läßt sich jeder Geraden $g \subseteq \mathbb{R}^3$ eindeutig ein Zahlenvektor $\vec{x} \in \mathbb{R}^5$ zuordnen. In $\mathbb{R}^5$ läßt sich eine Ordnung aufbauen und demnach auch auf der Menge der Geraden. Wir fahren nun weiter fort wie folgt. Erneut durchlaufen wir alle Kanten und ordnen jeder Kante e, mit Hilfe der Funktion *geraden_id(e)*, ihren eindeutigen Geradenvektor $(a_1, ..., a_5) \in \mathbb{R}^5$ zu. (Laufzeit: $O(n_e)$) Anschließend sortiere man die Liste der Kanten L_{edge} derart, daß Kanten gleicher Geraden nebeneinander liegen mit Hilfe der Vergleichsfunktion *comp_edge*. (Laufzeit:$O(n_e log(n_e))$).

```
comp_edge(e1, e2){
    double F1[5], F2[5];     // 2 Felder zur Speicherung von je 5 reellen Zahlen
    Übertrage das Zahlentupel der Kante e1 : (a11, ..., a15) auf das Feld F1
    Übertrage das Zahlentupel der Kante e2 : (a21, ..., a25) auf das Feld F2

    for i = 1  to  5 do {
        if( F1[i] < F2[i] ) return -1;                       // return( g_e1 < g_e2 )
        if( F1[i] > F2[i] ) return 1;                        // return( g_e1 > g_e2 )
    }    // F1[i] = F2[i]     (invariant)

    return 0;                                                // return( g_e1 = g_e2 )
}
```

Schnitt zwischen einer Geraden und einer Ebene

Sei $g = \{ \vec{x} \in \mathbb{R}^3 \ : \ \exists! \, \lambda \in \mathbb{R} \ \therefore \ \vec{x} = \vec{v_1} + \lambda \cdot (\vec{v_2} - \vec{v_1}) \ \}$ eine Gerade. Dabei sind $v_1, v_2 \in \mathbb{R}^3$ fest gewählt, mit $v_1 \neq v_2$.
Sei $H = \{\vec{x} \in \mathbb{R}^3 \ : < \vec{n}, \vec{x} >= c \ , \ c \in \mathbb{R}\}$ eine Ebene mit der Normalen $\vec{n}$ und dem Skalar c, dann gilt:

$$\boxed{g \not\parallel H \quad \Leftrightarrow \quad < \vec{n}, \vec{v_2} - \vec{v_1} > \neq 0}$$

Gilt $g \not\parallel H$, so berechnet sich der eindeutige Schnittpunkt $\vec{s} = g \cap H$ wie folgt (Herleitung Abschnitt 1.4.1 S. 20):

$$\boxed{\vec{s} = \vec{v}_1 + \frac{c - < \vec{n}, \vec{v}_1 >}{< \vec{n}, \vec{v}_2 - \vec{v}_1 >} \cdot (\vec{v}_2 - \vec{v}_1)}$$

5.3.3 Sortieren der Knotenfelder

Sei g eine Gerade des Geradenfeldes F_g. Dann repräsentieren wir diese, indem wir die Gerade g auf eine Kante (des zugehörigen Kantenbündels) $e = (v_1, v_2) \in L_{edge}$ verweisen. Durch den Vektor $(v_2 - v_1)$ bekommt die Gerade g eine Richtung (gerichtete Gerade). Nun können wir die Knoten des Knotenfeldes dieser Geraden g nach der Richtung von g sortieren, indem wir eine Knotenordnung auf g wie folgt festlegen:

Sei $g = \{ \vec{x} \in \mathbb{R}^3 \ : \ \exists! \, \lambda \in \mathbb{R} \ \therefore \ \vec{x} = \vec{v}_1 + \lambda \cdot (\vec{v}_2 - \vec{v}_1)\}$ $\quad (v_1, v_2 \in \mathbb{R}^3$ fest gewählt, mit $v_1 \neq v_2)$ Damit kann jedem $x \in g$ genau ein $\lambda_x \in \mathbb{R}$ zugeordnet werden, das sich wie folgt aus den Punkten x, v_1, v_2 berechnen läßt:

$$\begin{array}{rrcl}
 & \vec{x} & = & \vec{v}_1 + \lambda_x \cdot \overbrace{(\vec{v}_2 - \vec{v}_1)}^{=\vec{v}_{21} \neq \vec{0}} \\
 & \overbrace{(\vec{x} - \vec{v}_1)}^{\vec{p}} & = & \lambda_x \cdot \vec{v}_{21} \\
\Leftrightarrow & < \vec{p}, \vec{v}_{21} > & = & \lambda_x < \vec{v}_{21}, \vec{v}_{21} > \\
\Leftrightarrow & \lambda_x & = & \frac{<\vec{p}, \vec{v}_{21}>}{<\vec{v}_{21}, \vec{v}_{21}>}
\end{array}$$

Seien $x_1, x_2 \in g$ dann legen wir folgende Ordnung fest:

$$\begin{array}{rcl}
\vec{x}_1 < \vec{x}_2 & \Leftrightarrow & \lambda_{x_1} < \lambda_{x_2} \\
\vec{x}_1 = \vec{x}_2 & \Leftrightarrow & \lambda_{x_1} = \lambda_{x_2} \\
\vec{x}_1 > \vec{x}_2 & \Leftrightarrow & \lambda_{x_1} > \lambda_{x_2}
\end{array}$$

Damit ist unter den Punkten, der Geraden g eindeutig eine Ordnung definiert. Zunächst berechnen wir für jeden Knoten v der Knotenfelder aller Geraden $g \in F_g$, das eindeutig zu v bzgl. seiner Geraden gehörige λ_v. (Laufzeit: $O(n_e)$) Danach sortieren wir die Knoten jedes Knotenfeldes bzgl. der obig festgelegten Knotenordnung mit Hilfe des bereits für jeden Knoten v vorberechneten λ_v. (Laufzeit: $O(n_e \cdot log(n_e))$

5.3.4 Knotenpropagierung

Für jede Gerade g des Geradenfeldes F_g sind die Knoten des zugehörigen Knotenfeldes bzgl. der Richtung der Geraden sortiert. (Sei $v \in L_{vertex}$ ein Knoten, so ist v in all denjenigen Knotenfeldern zu finden, auf deren zugehöriger Gerade g_i der Knoten v liegt.) Erneut besuchen wir alle Kanten der Kantenliste L_{edge}. Für jede Kante $e = (v_1, v_2)$ führe man folgendes aus:

Die Kante $e_{g_i} := e$ repräsentiere die Gerade g_i. (e hat direkten Zugriff auf g_i)

Fall 1
Die Kante e_{g_i} habe die gleiche Richtung wie die Gerade g_i, welche durch die Knoten (v_1^*, v_2^*) repräsentiert wird.

$$\boxed{< v_2 - v_1, v_2^* - v_1^* > \; > 0 \Leftrightarrow e_{g_i} \text{ und } g_i \text{ sind gleichgerichtet}}$$

Man bestimme die Feldnummer j_1 des Knotens $v_1 = F_{v_{g_i}}[j_1]$ bzgl. des Knotenfeldes $F_{v_{g_i}}$ und entsprechend die Feldnummer j_2 des Knotens $v_2 = F_{v_{g_i}}[j_2]$. Wir finden die Feldnummern j_1 bzw. j_2 in logarithmischer Zeit durch Binärsuche, da die Knotenfelder der Geraden bereits sortiert sind. (s. Abschnitt 5.3.3)

Nun bestimmen wir sukzessive alle Knoten:

$$F_{v_{g_i}}[j_1 + 1], ..., F_{v_{g_i}}[j_2 - 1]$$

und tragen diese Knoten in dieser Reihenfolge in eine eigens für die Kante e_{g_i} neu anzulegende Knotenliste und bezeichnen diese als *Zwischenknotenliste* dieser Kante. Diese Liste bleibt leer, falls kein solcher (Zwischen-)Knoten existiert.

Fall 2
Die Kante e_{g_i} ist der Geraden g_i entgegengerichtet. Dann erfolgt die Behandlung umgekehrt analog.

Bleibt die Knotenliste leer, so ist die Kante e_{g_i} auch Kante der neu zu gewinnenden Randbeschreibung von P. Ist hingegen die Liste nicht leer, so liegen die Knoten der Zwischenknotenliste von e_{g_i}, in dieser Reihenfolge, zwischen den Knoten v_1 und v_2 auf der Kante e_{g_i}. Damit müssen diese Knoten noch auf die Kante e_{g_i} propagiert werden (nächster Abschnitt), wodurch die Kante in Unterkanten zerlegt wird, welche dann die Kante e_{g_i} in der neuen Randbeschreibung von P ersetzen. Wir besuchten alle Kanten

e der Liste L_{edge} und zusätzlich mit den zu propagierenden Knoten auch alle (Unter-)Kanten von e. Damit haben wir auch alle Kanten der neuen Randbeschreibung B_P des Polyeders P besucht. Inklusive der Binärsuche der Knoten brauchen für diesen Schritt eine Laufzeit von $O(n_e \cdot log(n_e))$.

Schließlich besucht man nochmals alle Kanten $e \in L_{edge}$. Dies tun wir, indem wir für jede Fläche $f \in L_{face}$ alle Kanten dieser Fläche in Reihenfolge besuchen. Wir leeren dazu zunächst die Knotenliste der Fläche und hängen sukzessive für jede Kante $e = (v_1, v_2)$ zunächst den Knoten v_1 in die Knotenliste. Hieran hängen wir die Zwischenknotenliste der Kante e (Knotenpropagierung). Anschließend hänge man noch v_2 in die Liste.
(Laufzeit: $O(n_e)$)

Damit repräsentieren die Listen L_{face} und L_{vertex} den Rand B_P von P gemäß dem externen Datenmodell wie es für das konvexe Polyeder bereits auf Seite 25 beschrieben wurde. Will man daraus noch das interne Datenmodell für das Polyeder P gewinnen, so verfahre man wie in Kapitel 2 Abschnitt 2.3 Seite 28 beschrieben.
(Laufzeit: $O(n_e \cdot log(n_e))$)

5.4 Konvertierung Boundary-Repr nach BSP-Repr

Sei $\hat{P}$ ein in der Boundary – Representation gegebenes Polyeder. Wir interessieren uns für die BSP – Representation von $\hat{P}$. Seien $\{f_1, ..., f_m\}$ die Begrenzungspolygone von $\hat{P}$. Entgegen unserer Boundary-Repr, gehen wir davon aus, daß die Repräsenationen dieser Flächen derart gerichtet sind, daß ihre Normalen in das Innere des Polyeders $\hat{P}$ zeigen und nicht aus ihm herauszeigen.(vgl. Satz 7 Seite 14)

Dabei gehen wir wie folgt vor. Wir konstruieren die das Polyeder $\hat{P}$ (minimal) umschließende Box. Die Box splitten wir mit einer Ebene H_f, für die in der Menge der Begrenzungsflächen $\hat{L} = \{f_1, ..., f_m\}$ eine zur Ebene H_f gleichorientierte Fläche f existiert und erhalten die Polyeder P_+ und P_-. Wir propagieren all diejenigen Flächen $f \in \hat{L}$ nach P_+ für die gilt:

1. $f \in H_+^\circ$

2. Falls $f \subseteq H$, dann sind f und H gleichorientiert.

und merken uns diese Flächen für P_+ in der Flächenmenge L_+. Die restlichen Flächen aus $\hat{L}$ merken wir uns für das Polyeder P_- in einer Flächenmenge L_-. Analog wie wir die Box mit der Liste L verarbeiten, so verarbeiten wir P_+ mit L_+ und P_- mit L_-. Wir erhalten auf diese Weise, insbesondere mit Hilfe der Operation *Split-Polyeder*, einen Bsp-Tree T_{BSP} mit der Box als Wurzel. Ein Polyeder P aus T_{BSP} ist genau dann ein Blatt, wenn in dessen zugehöriger Flächenmenge L_P keine Fläche f existiert, mit welcher wir das Polyeder P weiter splitten können. $\hat{P}$ ergibt sich dann aus der Vereinigung all derjenigen Blätter P_{Blatt}, deren zugehörige Flächenmenge L_{Blatt} nicht leer ist. Alle anderen Blattpolyeder schneiden sich mit $\hat{P}$ höchstens in ihren Rändern. Damit wir für einen Polyeder P nicht alle Flächen aus seiner zugehörigen Flächenmenge L_P auf Split mit P untersuchen müssen, teilen wir L_P in 2 zueinander komlementäre Flächenmengen L und L^* auf für die gilt:

- $L_P = L \dot{\cup} L^*$

- L^* ist die Menge aller Flächen $f \in L_P$ deren zugehörige Ebene $H_f \supseteq f$ bereits für den Split eines Vorfahren von P verwendet wurde.

Eine ähnliche Vorgehensweise finden wir in [FKN80] zur Berechnung des Hidden-Surface-Problems für n Polygone im Raum. Die Funktion, welche die Konvertierung von der Boundary-Repr zur BSP-Repr berechnet bezeichnen wir mit *B_BSP_Konvertierung()*, und sie funktioniert wie folgt.

```
B_BSP_Konvertierung(){

    - Konstruiere P_Box, das Polyeder, welches das Polyeder P̂ als Box
      (minimal) umschließt;

    - Die Flächen f_1, ..., f_m speichern wir in einer Flächenliste L_∘.
      L*_∘ sei eine zusätzliche, aber leere Flächenliste;

    - führe die Operation make_BSP(P_Box, L_∘, L*_∘) durch;

}
```

```
make_BSP(P, L, L*){

   if(L = L* = ∅) markiere P mit OUT;

   else{
      if( ∃ f ∈ L mit P / H_f = {P_+, P_-} ){
         // Dabei ist H_f ⊆ IR^3 eine Ebene mit:
         // f ⊆ H_f, wobei f und H_f gleichorientiert sind.

         L_+, L*_+, L_-, L*_-  seien 4 neue Flächenlisten;

         forall(f' aus (L ∪ L*)) do {
            Fallunterscheidung( H_f, L, L*,L_+, L*_+,L_-, L*_- );
         }

         make_BSP(P_+, L_+, L*_+);
         make_BSP(P_-, L_-, L*_-);

      }else
         markiere P mit IN;
   }
}
```

```
Fallunterscheidung( H_f, L, L*, L+, L*+, L-, L*- ){

    H := H_f;

     1.Fall   f' ⊄ H
      1.1     f' ⊆ H°+{
                 if(f' ∈ L)    append(f', L+);
                 if(f' ∈ L*)   append(f', L*+);
              }
      1.2     f'/H = {f+, f-}{
                 append(f+, L+);
                 append(f-, L-);
              }
      1.3     f' ⊆ H°-    //(analog wie 1.1.)

     2.Fall   f' ⊆ H{
                 if( f' und H sind gleichorientiert )  append(f', L*+);
                 else                                  append(f', L*-);
              }

}
```

(append(f, L) d.h. hänge die Fläche f an die Liste L)

Invarianten

- Zu Beginn der Operation *make_BSP(P, L, L*)* gilt:
  - Sind L und L^* leere Listen, dann gilt: $P \cap \hat{P} = B_P \cap B_{\hat{P}}$
    (d.h. P und $\hat{P}$ sind disjunkt bis auf ihre Ränder.)
  - Sei $f' \in L^*$, dann:
    $\exists!\ H \subseteq \mathbb{R}^3$ (Ebene) derart, daß: $f' \subseteq H$, und f' und H sind gleich orientiert, und es gilt: $P \cap H^\circ_+ = P$
  - $\forall\ f' \in (L \cup L^*)$ gilt $f' \subseteq P$
  - $\forall\ f' \in (L \cup L^*)$ gilt $\exists f \in \{f_1, ..., f_m\}$ derart, daß $f' \subseteq f$
  - Seien $f_1, f_2 \in (L \cup L^*)$ beliebig mit $f_1 \neq f_2$, dann gilt:
    $\nexists f \in \{f_1, ..., f_m\}$ derart, daß $(f_1 \cup f_2) \subseteq f$
- Unmittelbar nach der Operation *Fallunterscheidung(...)* gilt:
  $|L_+| < |L|$ und $|L_-| < |L|$

Betrachtet man sich die oben zuletzt genannte Invariante genauer, so erkennt man, daß der Baum höchstens eine Tiefe von m haben kann, und daß die Operation

B_BSP_Konvertierung() endliche Laufzeit hat. Als Ergebnis erhalten wir dann:

$$\hat{P} = Vereinigung\, der\, mit\, IN\, markierten\, Blätter$$

Laufzeitanalyse

In [Ed87] finden wir folgenden Satz.

Satz 15

> *Seien m Ebenen $H_1, ..., H_m \subseteq \mathbb{R}^3$ gegeben. Sei m_p die Anzahl der Partitionen, in welche diese Ebenen den Raum $\mathbb{R}^3$ partitionieren, dann gilt:*
>
> $$m_p \leq m^3$$

Mit Satz 15 hat der durch die Operation *B_BSP_Konvertierung()* entstehende BSP-Tree eine Komplexität von $O(m^3)$. Folglich hat der Baum $O(m^3)$ Knoten. An jedem Knoten befindet sich ein Polyeder P mit einer Flächenliste L und einer Flächenliste L^*. Da wir nicht wissen welche Fläche $f \in L$ das Polyeder P splittet, müsen wir nacheinander die Flächen aus L daraufhin testen. Aus der zuletzt genannten Invarianten, folgert man, daß dazu höchsten m Splitvorgänge nötig sind ($|L| \leq m$), welche uns je $O(n)$ (Splitzeit) kosten. (Dabei ist n die Komplexität des Polyeders $\hat{P}$, gemessen in Anzahl der Kanten n.) Haben wir eine Fläche f und damit eine Ebene H ($f \subseteq H$) gefunden, welche das Polyeder P splittet, so müssen wir pro Fläche $f \in (L \cup L^*)$ die Lagebeziehungen von f zu H berechnen (*Fallunterscheidung*), was uns pro Fläche eine Zeit von (höchstens) $O(n)$ kostet. Weiterhin gilt mit den beiden zuletzt genannten Invarianten $(|L| + |L^*|) \leq m$. Folglich kostet der Aufenthalt in einem Knoten $O(m \cdot n)$. Der Baum hat eine Komplexität von $O(m^3)$. Somit erhalten wir für obiges Verfahren eine Gesamtlaufzeit von:

$$O(m^4 \cdot n)$$

Da wir hier die Laufzeit für den worst-case berechnet haben, haben wir in der Praxis durchaus bessere Laufeiten zu erwarten. Betrachtet man beispielsweise das Problem durchschnittlich, so halbiert sich Mächtigkeit der Flächenlisten pro Splitvorgang.

Kapitel 6

Boolesche Operationen auf Polyedern

Mit den beiden Basisoperationen *Split-Polyeder* und *Paste-Polyeder* sind wir in der Lage uns ein mächtiges Werkzeug im Bereich der virtuellen Modellierung zu schaffen — die Durchführung von Booleschen Operationen auf Polyedern.

Gegeben seien zwei beliebige Polyeder P_1 und P_2 — mit einer Gesamtkomplexität von n — in einer der beiden Repräsentationsarten BSP oder *Boundary*. Wir interessieren uns für die Boolesche Verknüpfung der beiden Polyeder P_1 und P_2 wie folgt:

- $P_1 \cap P_2$
- $P_1 \cup P_2$
- $P_1 \setminus P_2$

Zunächst sorgt man dafür (mit Hilfe der Konvertierungsoperation Split-Polyeder (S. 100)), daß beide Polyeder in der BSP-Repr gegeben sind.
(Laufzeit: $O(n^5)$)

Wir bilden die minimale Box, welche beide Polyeder umschließt und partitionieren die Box durch die markierten Blätter der BSP-Trees von P_1 und P_2 mit Hilfe der Operation Split-Polyeder. Mit der Partitionierung der Box erhalten wir einen weiteren BSP-Tree, an welchem wir das Ergebnis der Booleschen Operation — in Form der BSP-Repr — unmittelbar ablesen können.
(Laufzeit: $O(n^5)$ Herleitung analg zu Abschnitt 5.2.2 (S. 87) bzw. Abschnitt 5.4 (S. 103))

Mit Hilfe der Operation Paste-Polyeder können wir dann wiederum die BSP-Representation umwandeln in die Boundary-Representation. (Laufzeit: siehe Kapitel 5)

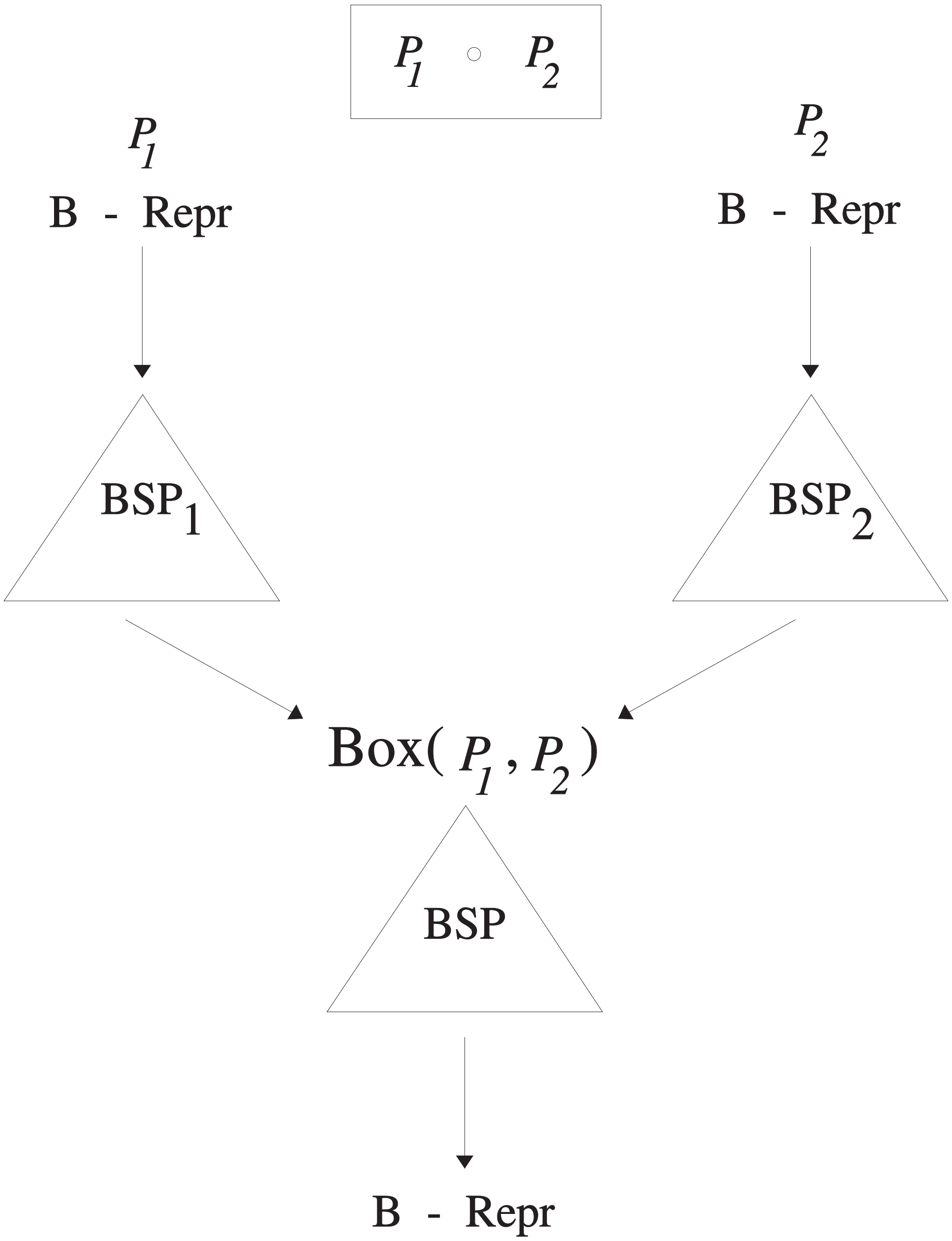

Abbildung 6.1: Schaubild zur Booleschen Operation

6.1 Kernverfahren der Booleschen Operation

Der Kern des Verfahrens der Booleschen Operation, in welchem der neue BSP-Baum berechnet wird, wird wie folgt in vier Teilverfahren durchgeführt. Seien nun die beiden Polyeder P_1, P_2 gegeben durch die sie repräsentierenden BSP-Trees:

- T_{BSP_1} repräsentiere P_1
- T_{BSP_2} repräsentiere P_2

1. **Box**
   Zunächst bestimme man die minimale Box P_{Box}, welche die Vereinigung der beiden Polyeder P_1 und P_2 als Untermenge enthält. Wir durchlaufen dazu einmal die Knotenlisten von P_1 und P_2 und bestimmen die Maximalkoordinaten: $x_{max}, x_{min}, y_{max}, y_{min}, z_{max}, z_{min}$, woraus wir die Box konstruieren.
   (Laufzeit: $O(n)$.)

2. **Partitionierung der Box durch $\mathbf{P_1}$**
   Wir betrachten P_{Box} als BSP-Tree mit dem Namen T_{BSP}, wobei P_{Box} die Wurzel und einziger Polyeder des Baumes ist.

   $\mathcal{P}1_{Blatt^*} = \{P_{11}, ..., P_{1m_1}\}$ die Menge der markierten Blätter des Baumes T_{BSP_1}.
   $\mathcal{P}2_{Blatt^*} = \{P_{21}, ..., P_{2m_1}\}$ die Menge der markierten Blätter des Baumes T_{BSP_2}.

   Wir lassen nun sukzessive die Polyeder $\{P_{11}, ..., P_{1m_1}\}$ mit der Operation *Partitonierung* (s. S. 70) in den Baum T_{BSP} *sickern.* (s. S. 71) Dabei soll die Operation *Partitionierung* wie folgt modifiziert sein. Das Blatt, welches mit IN markiert werden soll, soll nun mit **blau** markiert werden. Die anderen Blätter, welche mit OUT markiert werden sollen, sollen nun mit **weiß** markiert werden.

   Analog zur Laufzeitabschätzung der Flächenpartitionierung Abschnitt 5.2.2 Seite 87 benötigen wir dazu eine Laufzeit von $O(n^4 \cdot m_1)$.

3. **Partitionierung der Box durch $\mathbf{P_2}$**
   Analog lassen wir die markierten Polyeder $\mathcal{P}2_{Blatt^*} = \{P_{21}, ..., P_{2m_2}\}$ mit der Operation *Partitionierung* in den Baum T_{BSP} *sickern.* Auch hier wenden wir wieder eine Modifikation der Operation Partitionierung an, um anschließend in der Lage zu sein, für jedes Blatt des Baumes zu entscheiden, ob es im Komplement von $P_1 \cup P_2$, ob es in P_1 oder ob es in P_2 oder ob es in beiden enthalten ist. Die Operation Partitionierung ist dabei wie folgt modifiziert.

Partitionierung(P^*, P_{2i}) {

P^* ein Blatt aus T_{BSP} werde durch $P_{2i} \in \mathcal{P}_{2_{Blatt^*}}$ partitioniert gemäß der Operation *Partitionierung* (siehe S. 70);

// Dabei wird zusätzlich folgende Fallunterscheidung betrachtet:

(a) **if**(P^* ist **weiß** markiert){ $(P^* \subseteq \overline{P_1})$
    markiere **grün** anstelle mit IN ;
    markiere **weiß** anstelle mit OUT;
  }
(b) **else**{ // (P^* ist **blau** markiert) $(P^* \subseteq P_1)$
    markiere **rot** anstelle mit IN ;
    markiere **blau** anstelle mit OUT;
  }

}

Analog wie oben schätzen wir auch hier die Laufzeit mit $O(n^4 \cdot m_2)$ ab.

4. **Interpretation**

- P_1: Vereinigung der mit **blau** und **rot** markierten Blätter.
- P_2: Vereinigung der mit **grün** und **rot** markierten Blätter.
- $P_1 \cup P_2$: Vereinigung der mit **blau** und **grün** und **rot** markierten Blätter.
- $P_1 \setminus P_2$: Vereinigung der mit **blau** markierten Blätter.
- $P_2 \setminus P_1$: Vereinigung der mit **grün** markierten Blätter.
- $P_1 \cap P_2$: Vereinigung der mit **rot** markierten Blätter.

Schlußwort

Implementation

Die hier besprochenen Operationen machen den Kern der Implementation aus. Ein Großteil der Implementation wird jedoch bestimmt durch Operationen, welche ein komfortables Arbeiten mit dem Modelliereditor erst ermöglichen.
Dazu gehören:

1. Operationen zum Betrachten der Objekte in beliebigen Ansichten
2. Hidden - Surface - Operationen
3. Meßwerkzeuge zur Spezifizierung von beliebigen Raumpunkten
4. Automatisierung bestimmter Konstruktionsvorgänge: z.B. *Bohren* durch reguläre Objekte
5. Konstruktion von primitiven regulären Objekten, wie Rotationskörper oder Prismen aus Polygonen
6. Zeichenbrett zur Erstellung beliebiger Polygone
7. Benutzerdialogfenster um interaktives Arbeiten zu ermöglichen

Obwohl die Implementation dieser Operationen schon einiges an Programmierzeit verschlingt, sind dies doch nur Ansätze und Hinweise darauf, wo man sich als Benutzer bei der Modellierung mehr Komfort wünscht.

Rechenungenauigkeit

Zur Rechengenauigkeit beim Modellieren ist folgendes anzumerken. Die Variablen v_1 und v_2 sollen den gleichen geometrischen Punkt im Raum repräsentieren. Durch Rechenungenauigkeit kann es jedoch passieren, daß die Zahlendarstellung der Koordinaten zwischen den Punkten v_1 und v_2 Differenzen aufweisen. Um in beiden dennoch denselben Punkt zu erkennen, tolerieren wir einen Höchstabstand zwischen v_1 und v_2 von $eps \in \mathbb{R}_+$ Folglich müssen wir eps genügend **groß** wählen.

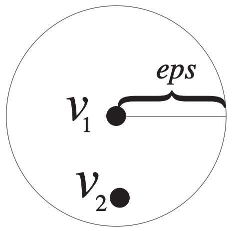

Abbildung 6.2: Toleranzgebiet des Raumpunktes v_1

Diesbezüglich fordern wir als Flächeninvariante, daß ein Knoten einer Polygonfläche sich nicht im Toleranzbereich eines anderen Knotens dieser Fläche befinden darf. Folglich müssen wir *eps* genügend **klein** wählen.

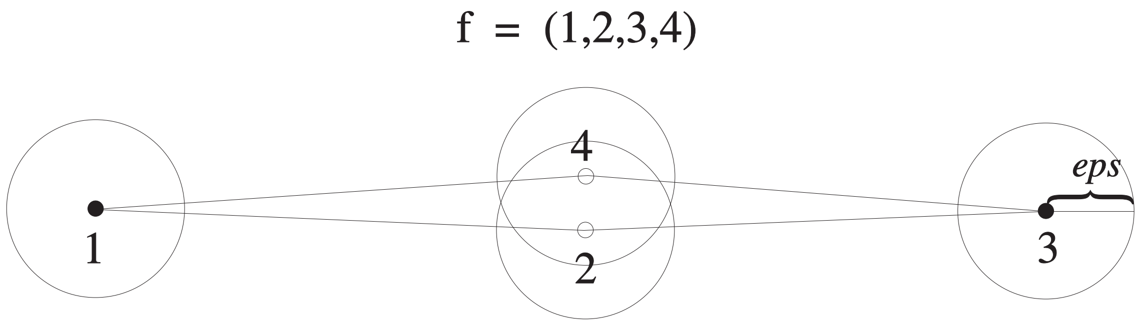

Abbildung 6.3: Unerlaubte Fläche f

Wir haben damit entgegengesetzt gerichtete Forderungen.

Abhilfe
Flächen, insbesondere neue Flächen, sollten stets bezüglich dieser Invariante untersucht werden. Neue Flächen entstehen in unseren Operationen nur durch Splitvorgänge. Man könnte beispielsweise für jeden am Split beteiligten Knoten den Lotabstand zur Splitebene berechnen und fordern, daß dieser Abstand die Zahl *eps* nicht unterschreitet. Wir fordern weiter, daß der Abstand, zwischen den beim Split einer Fläche neu entstehenden Knoten, ebenfalls *eps* nicht unterschreitet. Damit fordern wir sogenannte *fat objects*. (siehe auch Abbildung 6.4 Seite 110)

Anmerkung zur Laufzeitanalyse

Zur Laufzeit der in dieser Arbeit vorgestellten Operationen, insbesondere den Konvertierungsverfahren und dem Verfahren zur Durchführung von Booleschen Operationen

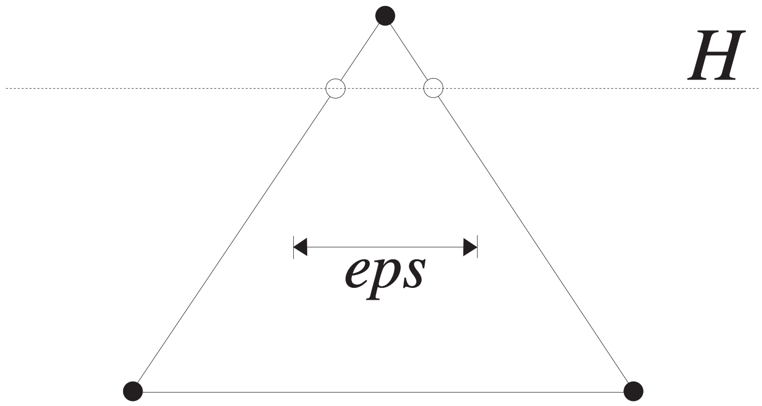

Abbildung 6.4: Unerlaubter Split

ist folgendes noch anzumerken. In den Laufzeitanalysen wurden vergleichsweise hohe, unerwünschte Laufzeiten ermittelt. Dies ist darauf zurückzuführen, daß diese Analysen recht grob durchgeführt wurden und diese Laufzeiten nur im *worst-case* ereicht werden können. Es ist zu vermuten, daß detailiertere Analysen zu besseren Ergebnissen in der Laufzeit dieser Verfahren führen können. Daß diese Laufzeiten dennoch kein Beinbruch sind, kann man in der Praxis am Modelliereditor beobachten. So benötigt man beispielsweise für die Umrechnung der BSP – Repr, des Hauses in Abbildung A.6 im Anhang, in die Boundary-Repr gerade einmal zehn Sekunden — was für den Konstrukteur am Bildschirm keine Belastung darstellen würde. Nicht zuletzt kommt es nicht nur auf Zeiteffizienz an. Es ist ebenso wichtig, daß das Modellierprogramm auch zuverlässig arbeitet. Man bedenke, daß zur Durchführung der booleschen Operationen nur zwei Basisoperationen als Kern der Implementation dienen, welche ohne große Schwierigkeiten implementierbar sind. Dies schafft enorme Vorteile bezüglich Programmsicherheit und des Programmierzeitaufwandes gegenüber herkömmlichen CAD-Systemen.

Anhang A

Beispiel aus dem Modelliereditor

Die nun folgenden Abbildungen, sollen einen kleinen Eindruck über die Möglichkeiten des bezüglich dieser Arbeit implementierten Modellierungseditor für reguläre Objekte geben.

Der **Stuhl** den wir in Abbildung A.1 sehen, wird durch den BSP-Tree in Abbildung A.2 repräsentiert. An dem Baum ist deutlich zu erkennen, wie die konvexen Einzelteile des Stuhles aus einem Quader durch sukzessive Anwendung der Operation Split-Polyeder herausmodelliert wurden. Der Stuhl ergibt sich dann aus der Vereinigung der dunkel markierten Blätter.

Der **Tisch** in Abbildung A.3 mit zugehörigem BSP-Tree in Abbildung A.4 beziehungsweise in Abbildung A.5 wurde aus einem Zylinder herausmodelliert, der zunachst in drei Teile zerlegt wurde. Der obere Teil entspricht der Tischplatte. Aus dem unteren Teil wurden die Tischfüße modelliert. Der mittlere Zylinder wurde durchbohrt, wobei der dabei entstandene Bohrzylinder als Tischbein verwendet wurde. Für die Bohrung wurde im Modellierungseditor eine spezielle Operation implementiert, welche folgendermaßen funktioniert. Als Bohrkreis dient ein Polygon mit n Ecken. Durch sukzessive orthogonale Schnitte entlang den Kanten des Polygones wird der Zylinder wie in Abbildung A.5 zu sehen ist partitioniert. Die Partitionen, welche den Bohrmantel bilden fallen dabei nach rechts weg. Damit erhalten wir den Bohrzylinder schließlich als linkestes Blatt des dadurch entstandenen Unter-BSP-Trees mit dem mittleren Zylinder als Wurzel.

In gleicher Weise modelliert man das **Haus** das in Abbildung A.6 zu sehen ist. Der BSP-Tree in Abbildung A.7 zeigt wie wir das Grundmodell des Hauses in Abbildung A.8 erhalten. Die Abbildungen A.8 bis A.15 zeigen einzelne Bauabschnitte bis zur Fertigstellung des Hauses. In Abbildung A.16 und Abbildung A.17 sehen wir das Haus mit zum Haus unabhängigem Möbiliar. Das Haus selbst ist aus ca. 50 konvexen Teilpolyedern zusammengesetzt. Die Umrechnung von der BSP-Representation in die Boundary-Representation dauerte 10 Sekunden.

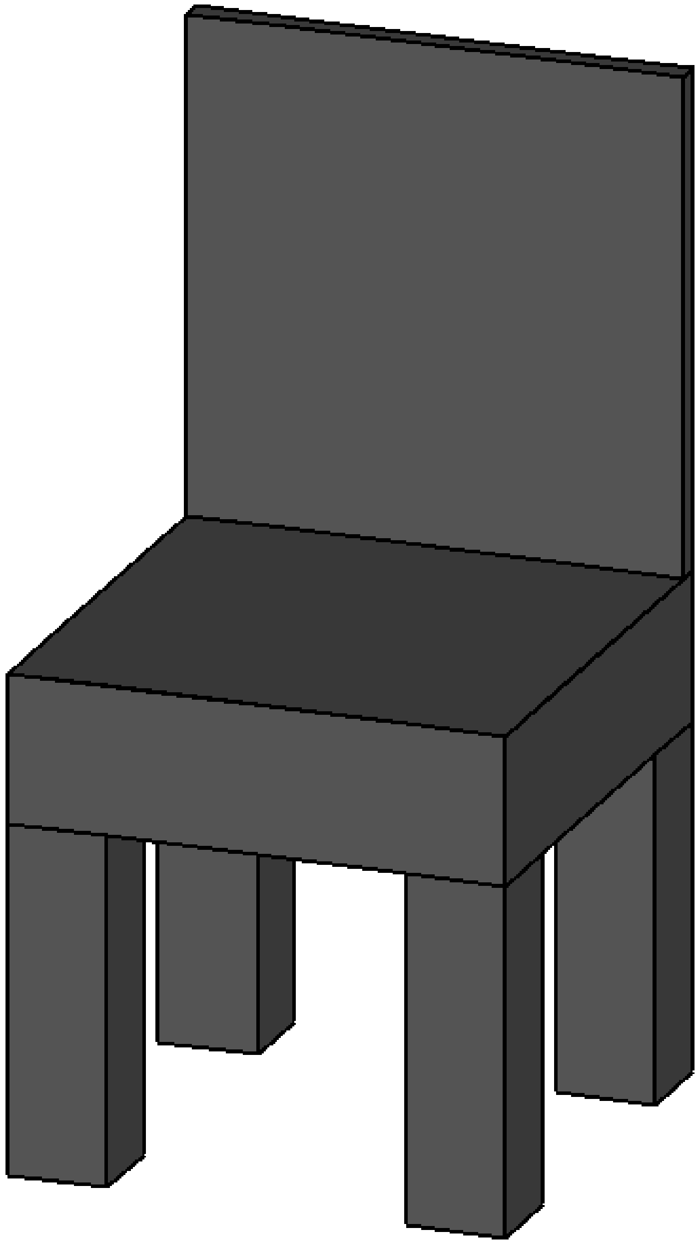

Abbildung A.1: Stuhl

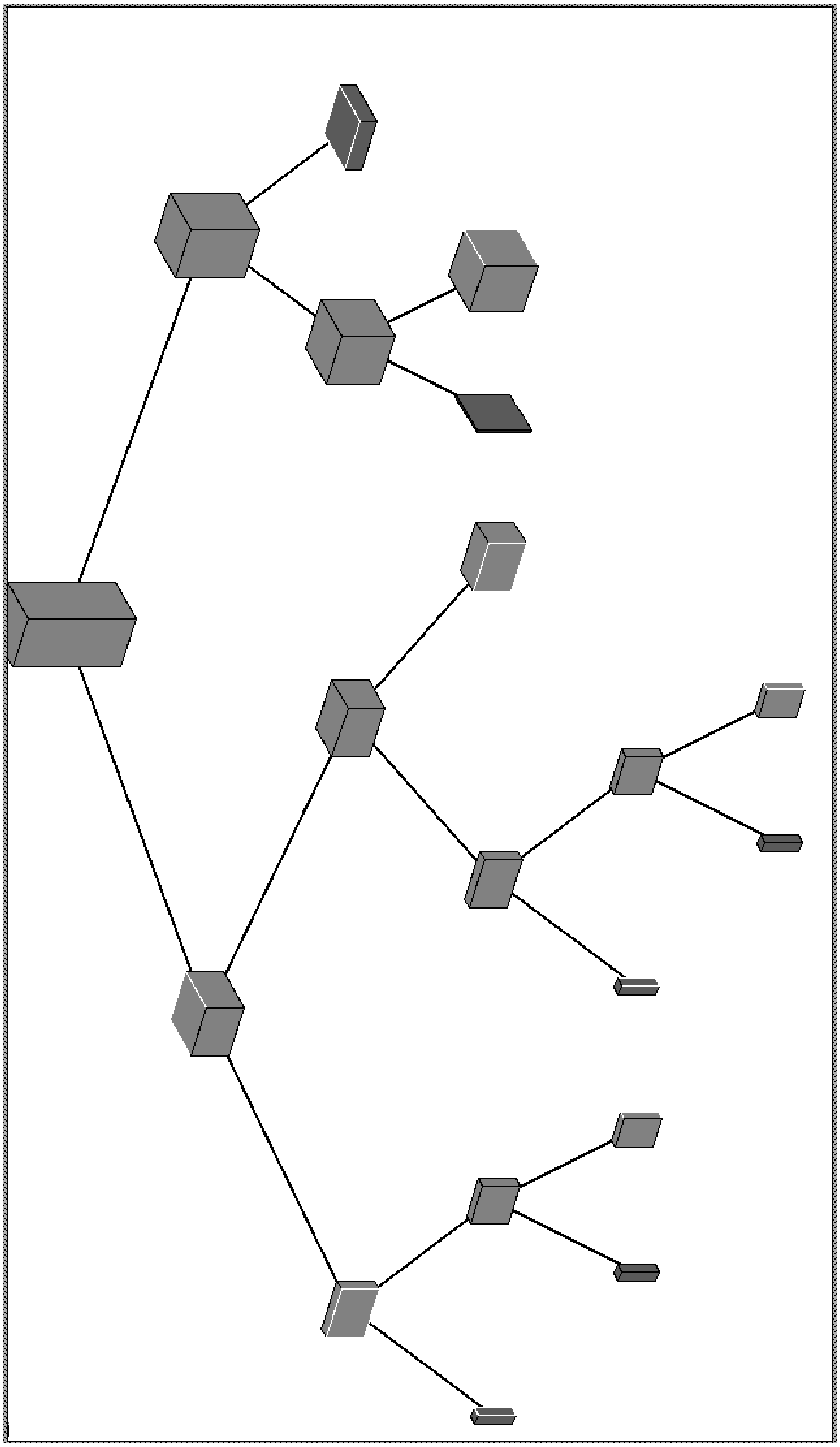

Abbildung A.2: BSP-Stuhl

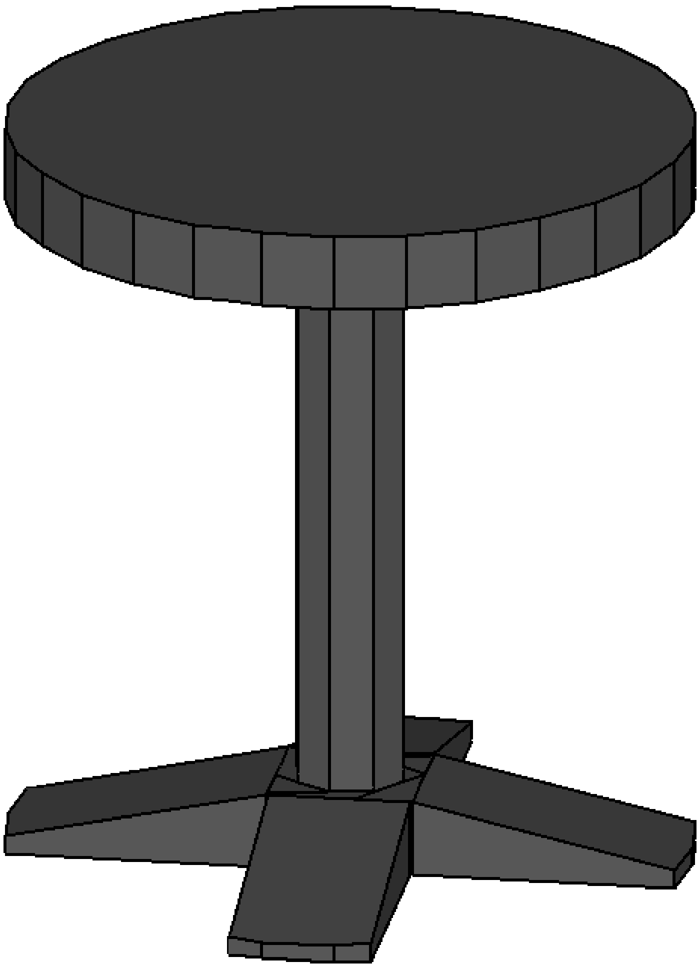

Abbildung A.3: Tisch

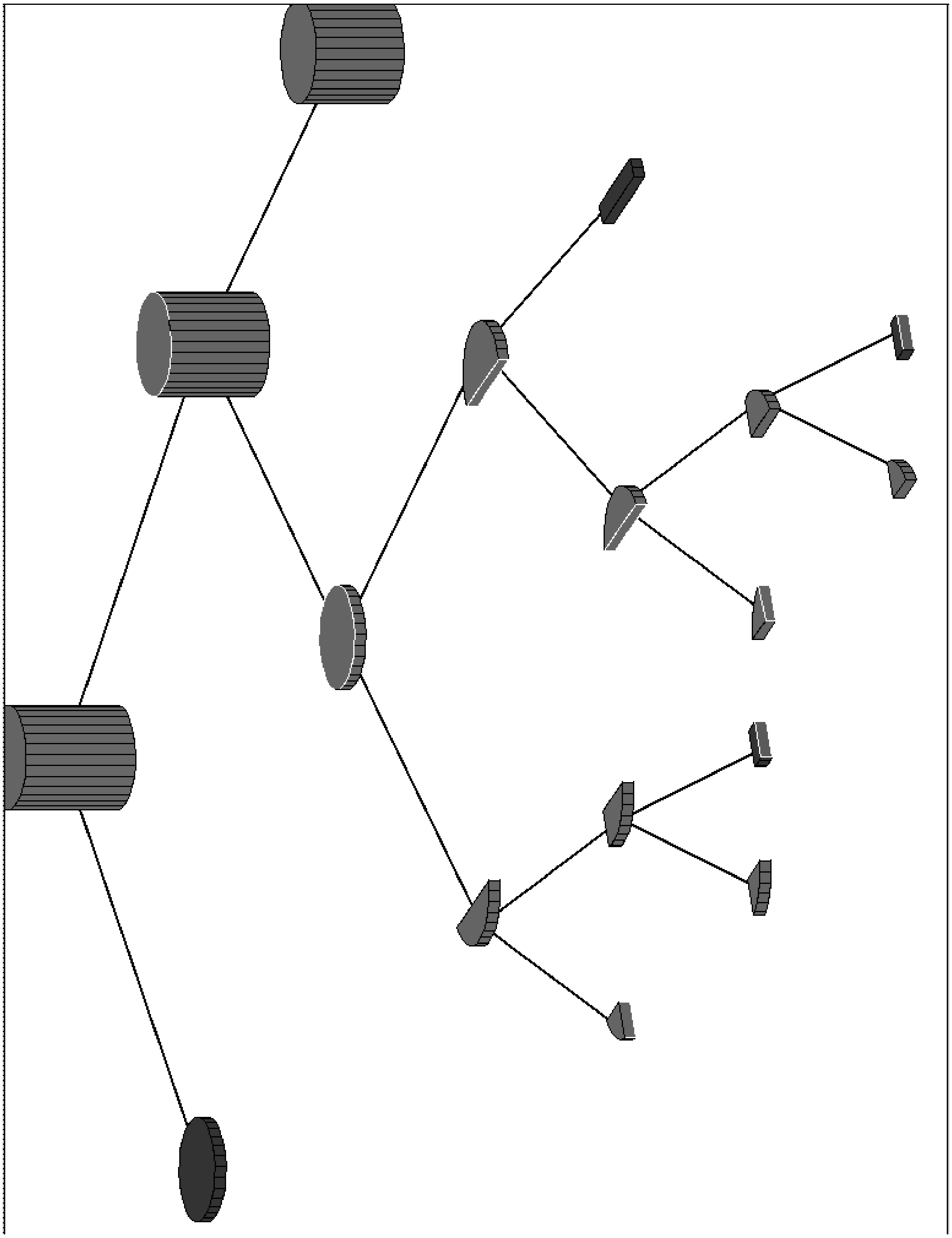

Abbildung A.4: BSP_1 - Tisch

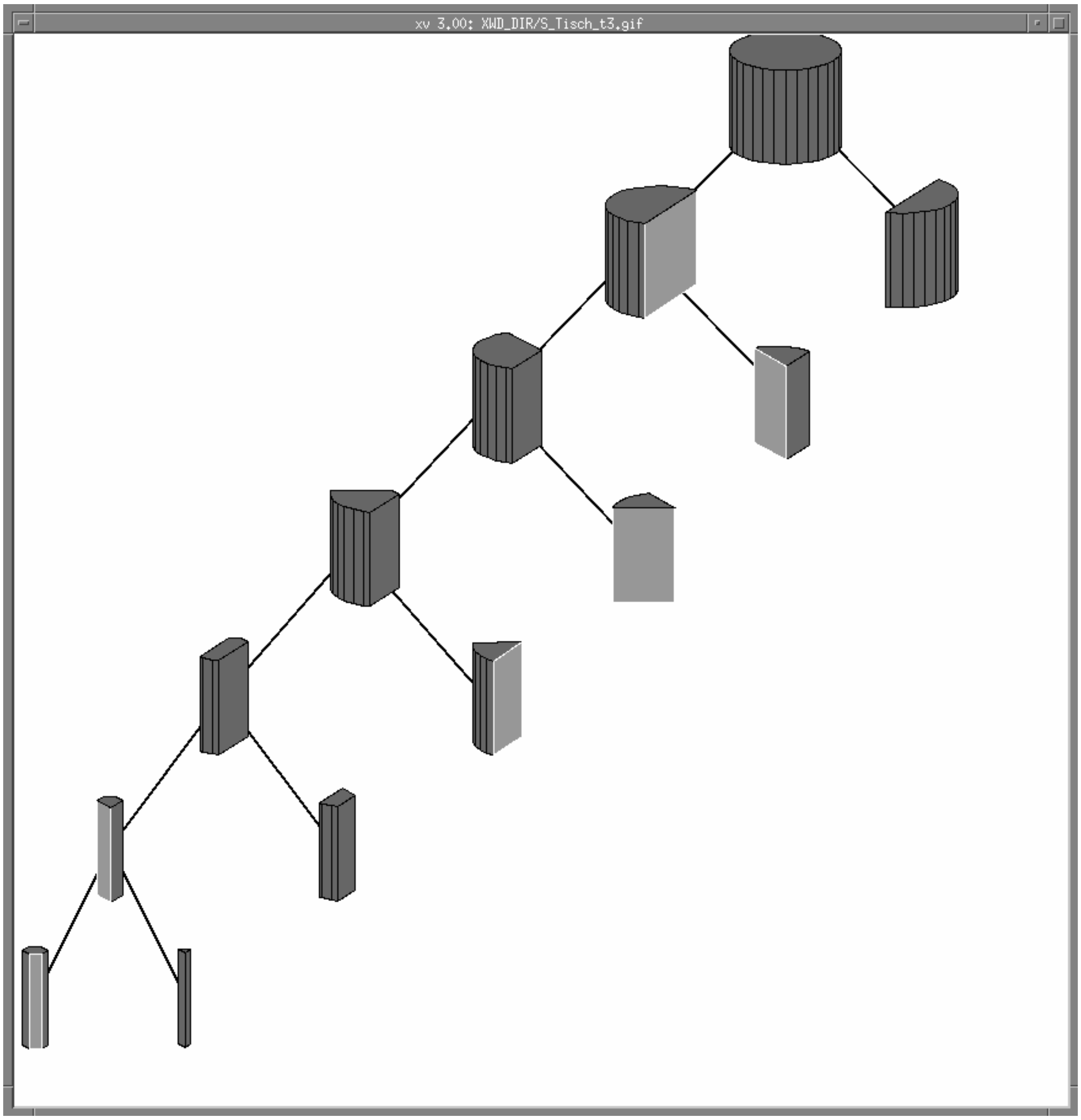

Abbildung A.5: BSP_2 - Tisch

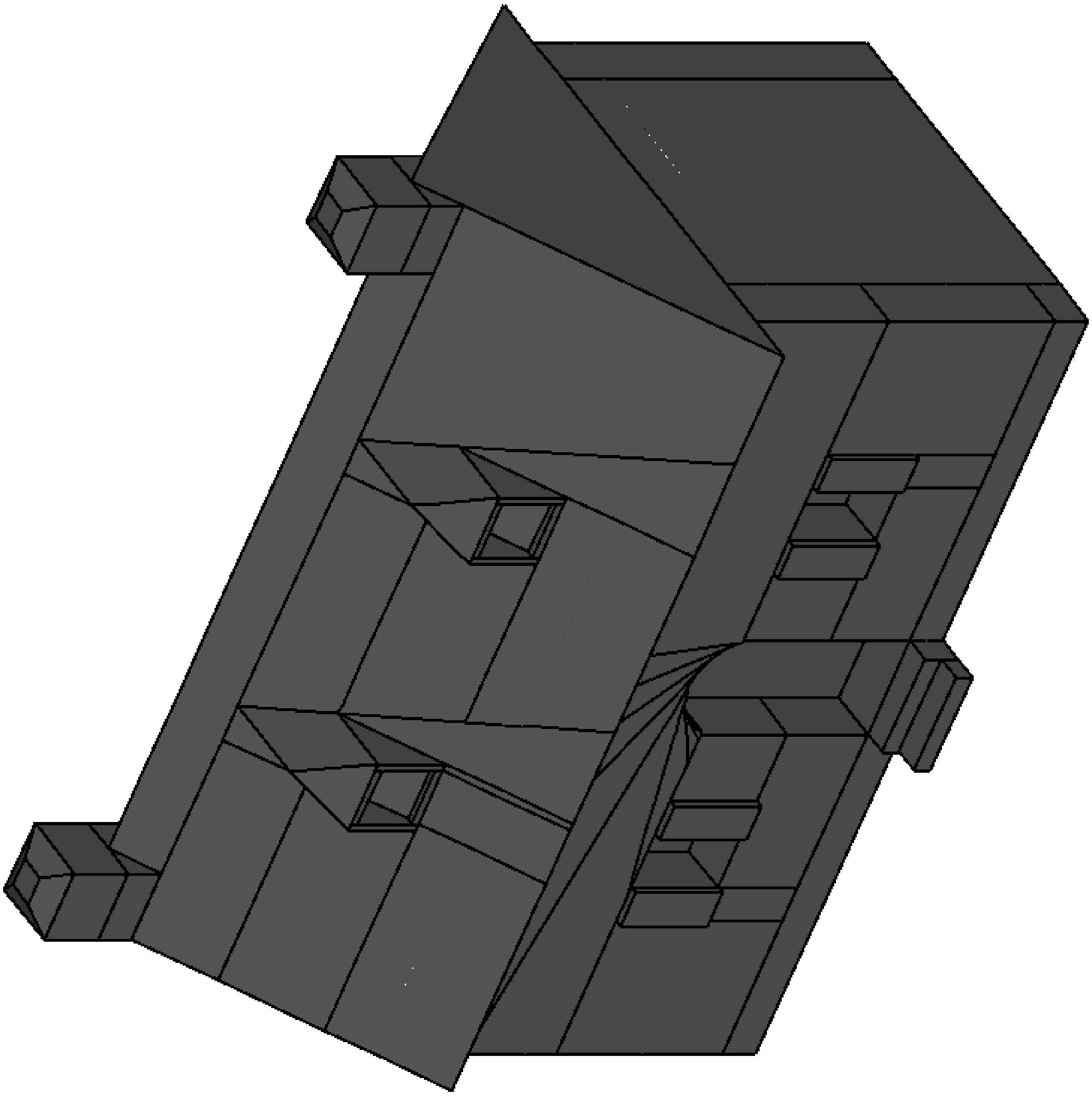

Abbildung A.6: Haus

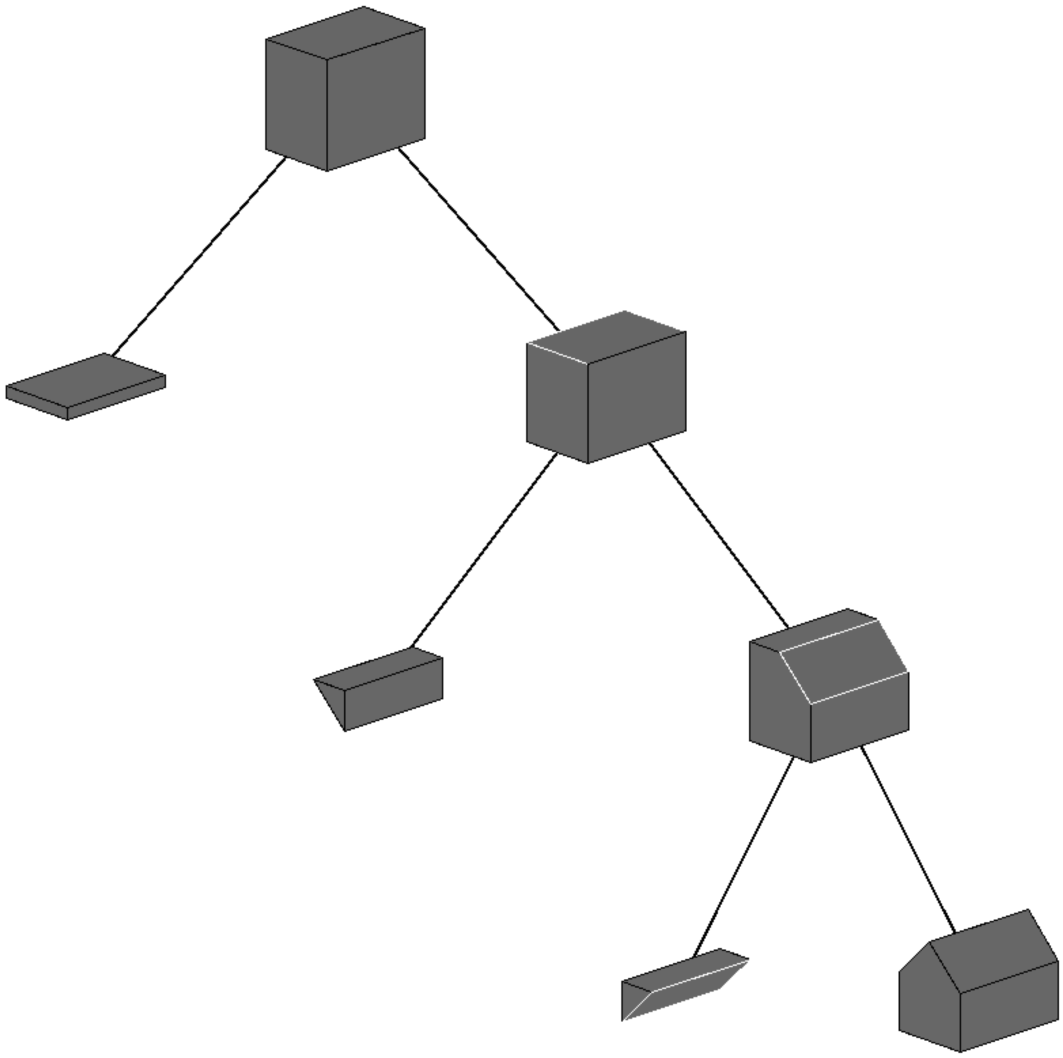

Abbildung A.7: BSP-Haus

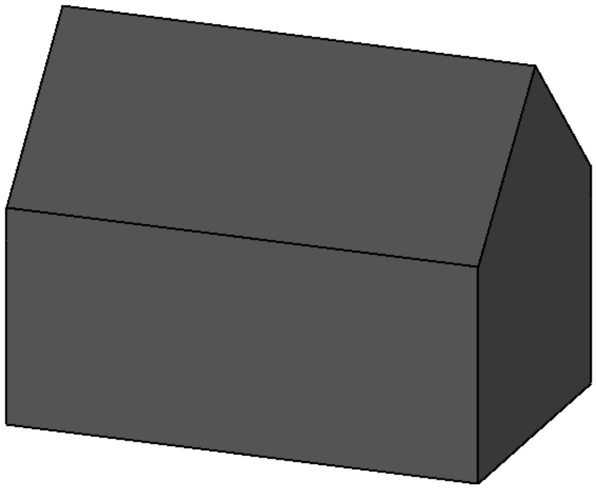

Abbildung A.8: Bauabschnitt 1

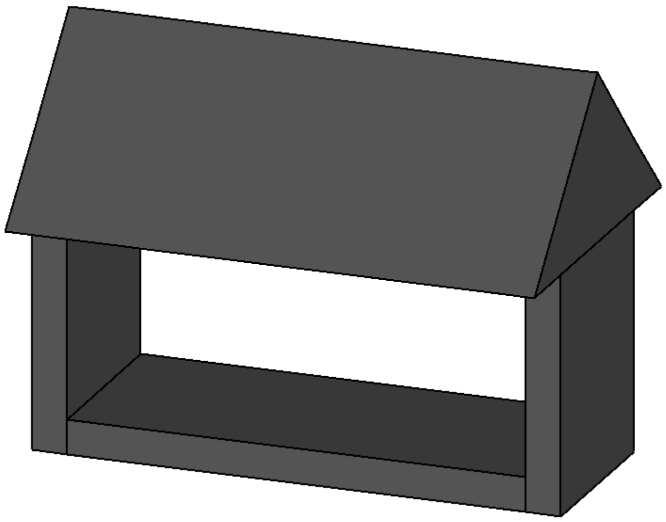

Abbildung A.9: Bauabschnitt 2

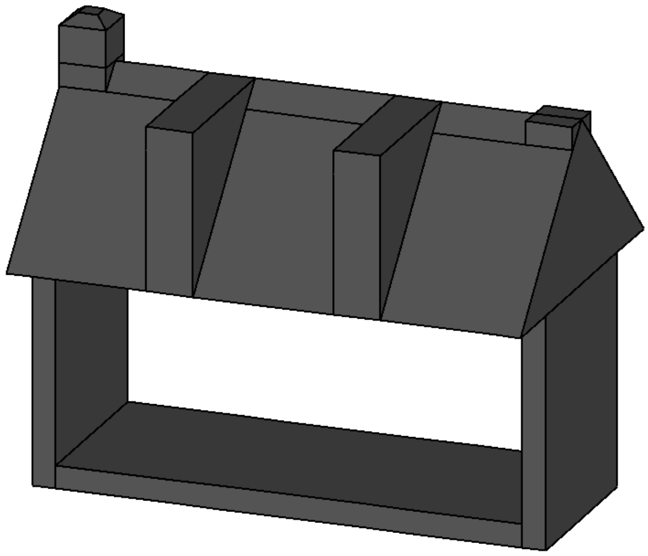

Abbildung A.10: Bauabschnitt 3

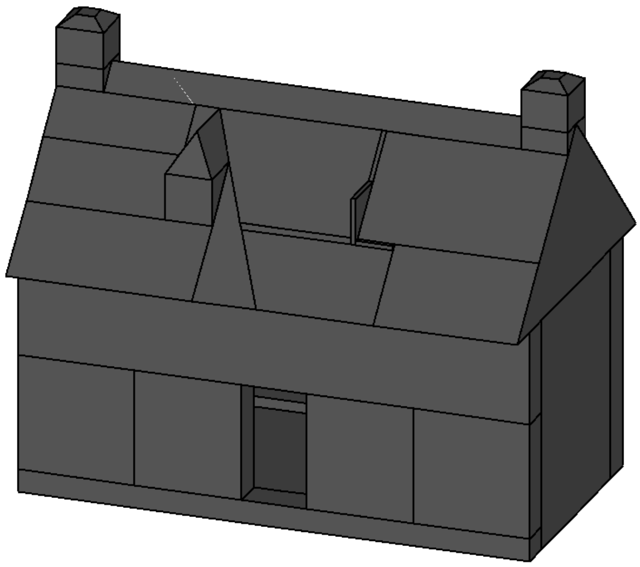

Abbildung A.11: Bauabschnitt 4

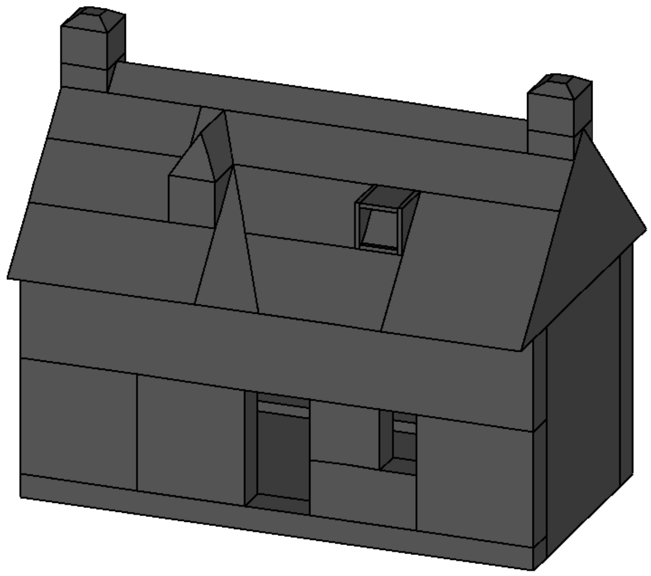

Abbildung A.12: Bauabschnitt 5

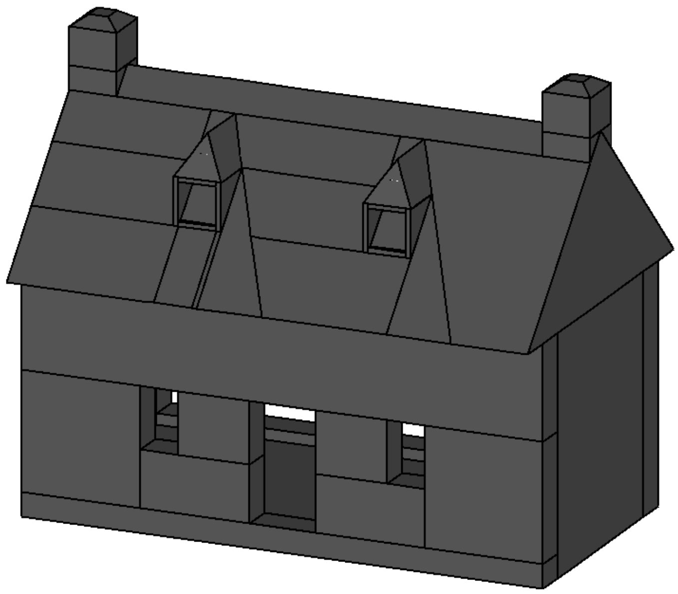

Abbildung A.13: Bauabschnitt 6

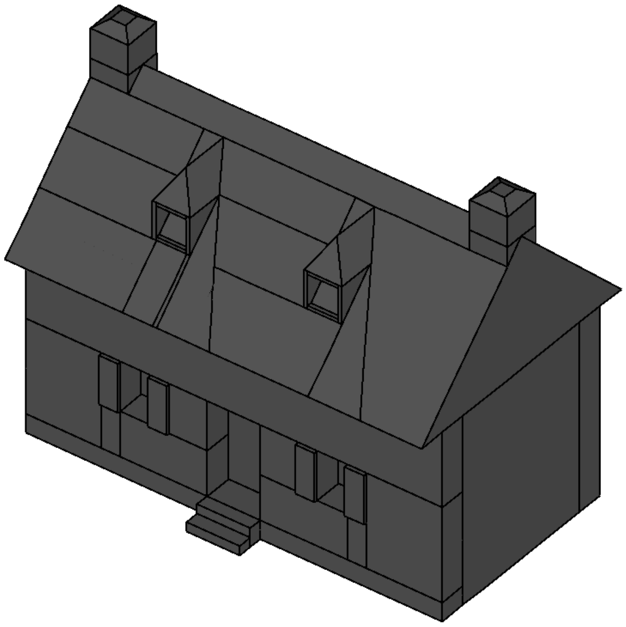

Abbildung A.14: Bauabschnitt 7

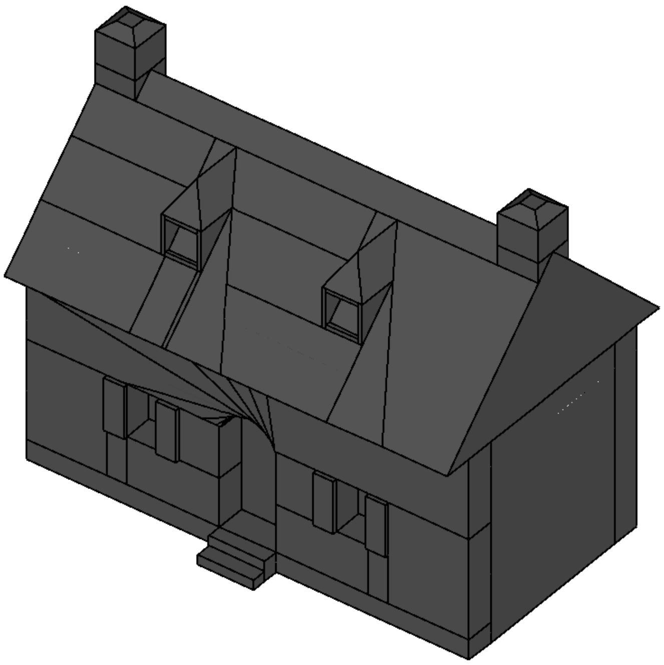

Abbildung A.15: Bauabschnitt 8

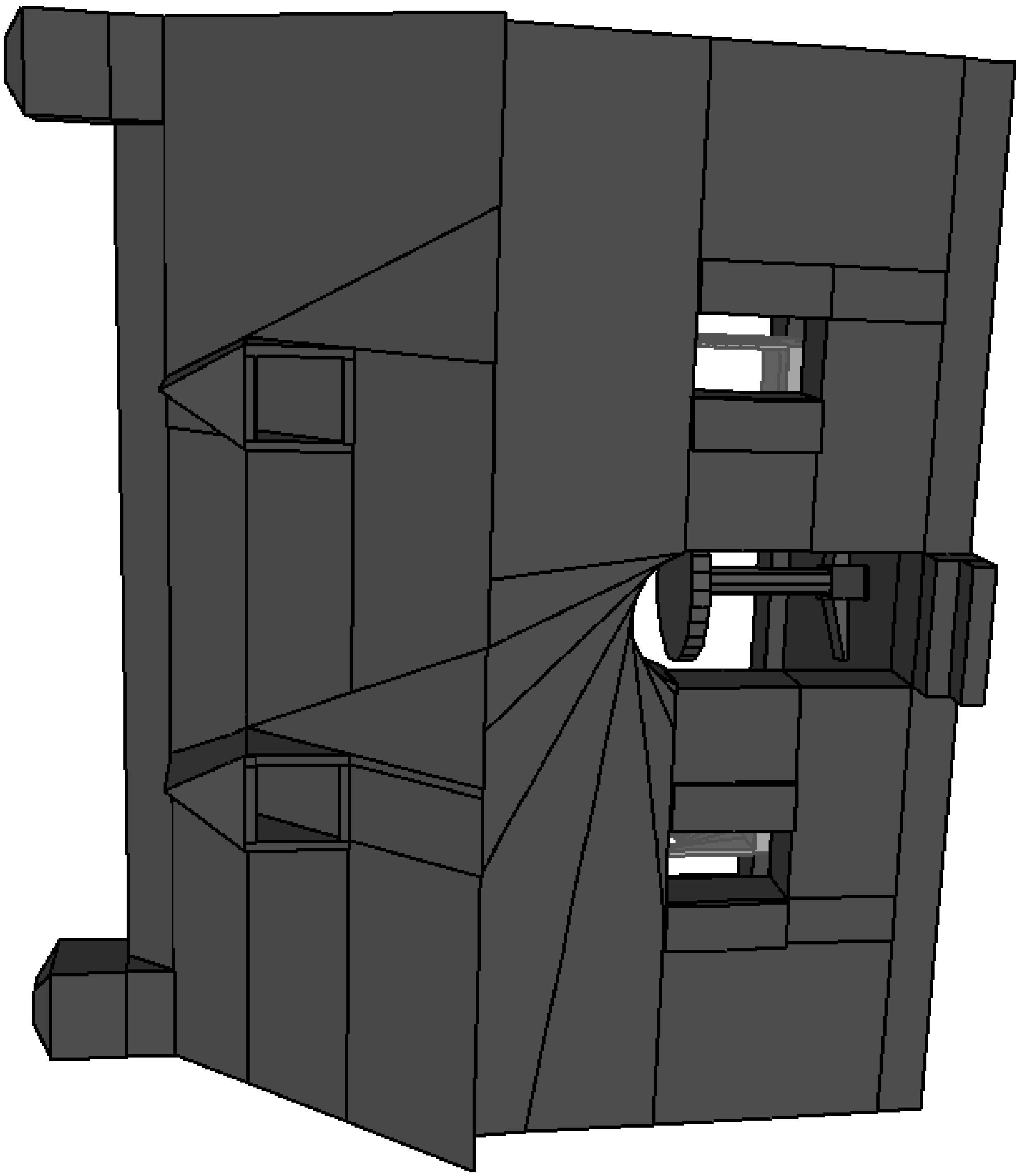

Abbildung A.16: Haus mit Einrichtung - Front

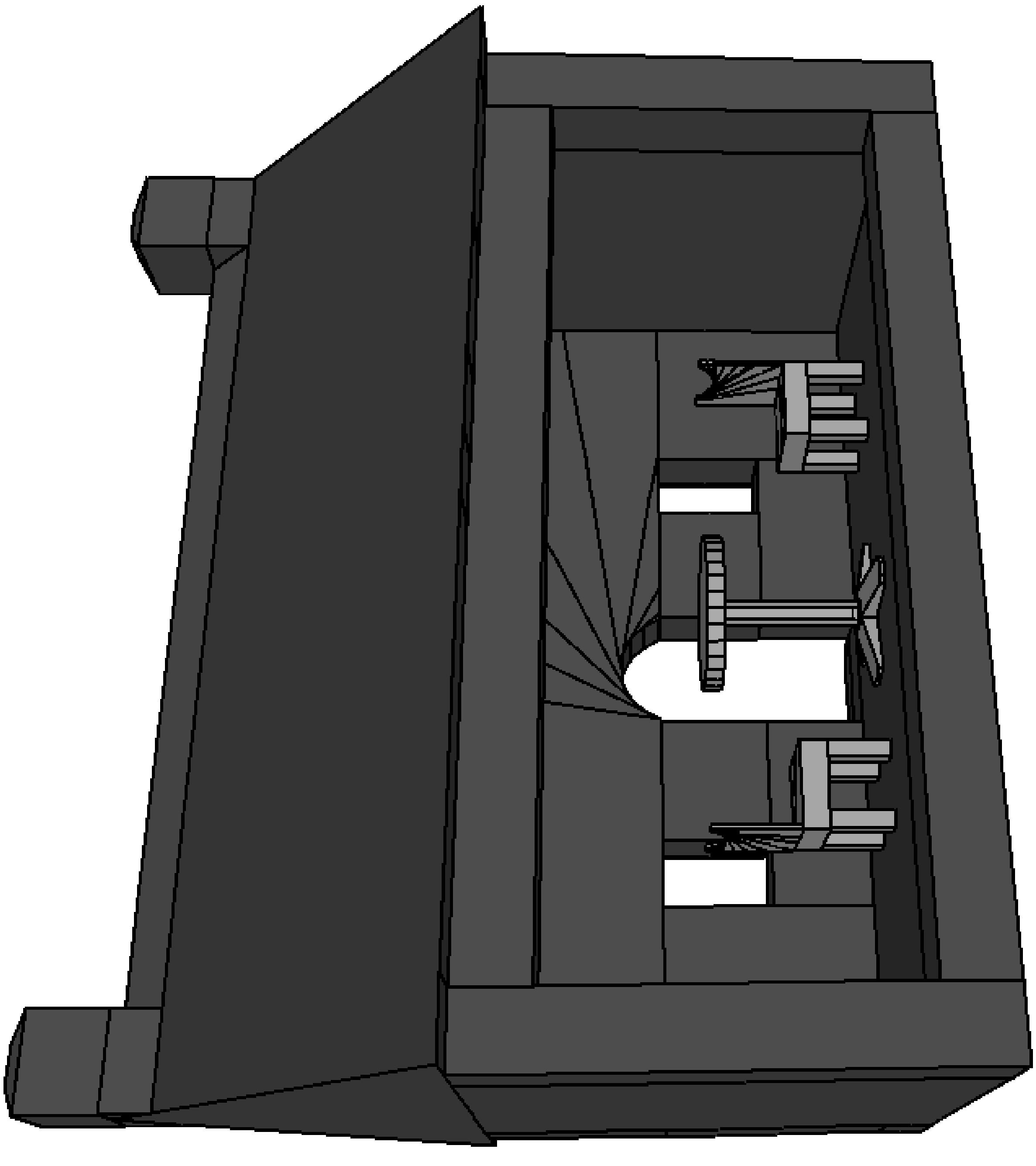

Abbildung A.17: Haus mit Einrichtung - Back

Literaturverzeichnis

[Bro91] I.N. Bronstein, K.A. Semendjajew:
Taschenbuch der Mathematik, Verlag Nauka, Moskau (1991)

[Ed87] H. Edelsbrunner:
Algorithms in Combinatorial Geometry, Springer Verlag, Berlin (1987)

[FKN80] H. Fuchs, Z. Kedem, B. Naylor:
On visible surface generation by a priory tree structures, Computer Graphics SIGGRAPH'80 Conference Proceedings), S. 124-133

[KeRi83] Brian W. Kernighan, Dennis M. Ritchie:
Programmieren in C, Hanser, München-Wien (1983)

[KHK74] J. Köhler, R. Höwelmann, H. Krämer:
Analytische Geometrie in vektorieller Darstellung, Verlag Moritz Diesterweg, Frankfurt a.M. (1974)

[L89] S.B. Lippmann:
C++ Primer, Addison Wesley, Publishing Company (1989)

[Meh84] K. Mehlhorn:
Data Structures and Algorithms, Vol. 1-3, Springer-Verlag, Berlin-Heidelberg (1984)

[Mei86] A. Meier:
Methoden der graphischen und geometrischen Datenverarbeitung, B.G. Teubner, Stuttgart (1986)

[Mul94] K. Mulmuley:
Computational Geometrie — an Introduction through randomized Algorithms, Prentice Hall Verlag S. 97-105

[Nae92] S. Näher:
LEDA User Manual: Library of Efficient Data Types and Algorithms, Version 3.0, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken (1992)

[Nay90] B. Naylor:
Binary space partitioning trees as an alternative representation of polytopes, computer-aided design, volume 22 number 4 (may 1990) S. 250-252

[PY89] M.S. Paterson, F.F. Yao:
Binary partitions with applications to hidden-surface removal and solid modeling, Proc. Fifth Symp. Computational Geometry (1989) S. 23-32

[PK87] R.A. Plastock, G.Kalley:
Theorie and Problems of the Computer Graphics, McGraw-Hill, Hamburg (1987)

[PrSh85] Franco P. Preparata, M.I. Shamos:
Computational Geometrie — an Introduction, Springer-Verlag, New-York (1985)

[Rau93] Th. Rauber:
Algorithmen in der Computergraphik, B.G. Teubner, Stuttgart (1993)

[S86] B. Stroustrup:
The C++ Programming Language, Addison-Wesley Publishing Company (1986)

[TN87] W. Thibault, B. Naylor:
Set operations on polyedra using binary space partitioning trees, Comput. Graph. Vol 21 No 4 (1987) S. 153-162

[XLPro] A. Nye:
The definitive Guides to the X Window System, Volume Two. Xlib Programming Manual for Version 11, O'Reilly & Associates, Inc. (1990)

[XLRef] A. Nye:
The definitive Guides to the X Window System, Volume One. Xlib Reference Manual for Version 11, O'Reilly & Associates, Inc. (1990)

[XMot] D. Heller:
The definitive Guides to the X Window System, Volume Six. Motiv Programming Manual for OSF/Motiv Version 1.1, O'Reilly & Associates, Inc. (1991)

[XTPro] A. Nye, T. O'Reilly:
The definitive Guides to the X Window System, Volume Four. X

Toolkit Intrinsics Programming Manual Second Edition for X11 Release 4, O'Reilly & Associates, Inc. (1990)

[XTRef] The Staff of O'Reilly & Assosiates, Inc.:
The definitive Guides to the X Window System, Volume Five. X Toolkit Intrinsics Reference Manual, O'Reilly & Associates, Inc. (1990)