UNIVERSITÄT
DUISBURG
ESSEN

Geometrische Modellierung
Seminar im Bereich B
Bild- u. sprachverarbeitende Systeme
Wintersemester 2008/09

Seminarausarbeitung

Binary Space Partitioning Trees

Grundlagen und Verfahren zur Erzeugung von BSP-Trees

Timm Linder

4. Januar 2009

Universität Duisburg-Essen – Fachbereich Ingenieurwissenschaften
Abteilung für Informatik und angewandte Kognitionswissenschaft
Fachgebiet Computergraphik & Wiss. Rechnen – Prof. Dr. W. Luther
Seminar betreut von Dr. E. Dyllong

Inhaltsverzeichnis

1 Einführung

1.1 Motivation und Gliederung

Eines der wesentlichen Ziele der 3D-Computergrafik ist die Darstellung komplexer geometrischer Gebilde in Echtzeit, wie dies etwa in interaktiven Simulationen oder Computerspielen heutzutage häufig erforderlich ist. Schon in den sechziger und siebziger Jahren des vergangenen Jahrhunderts war man auf der Suche nach effizienten Datenstrukturen und Algorithmen, um das Zeichnen von dreidimensionalen geometrischen Modellen so weit wie möglich zu beschleunigen.

Ein grundlegendes Problem, das sich gegenüber dem Entwickler einer 3D-Anwendung auftut, ist die Bestimmung der korrekten Zeichenreihenfolge für die zu rendernden Primitive: Objekte, die sich fernab vom Betrachter befinden, müssen zuerst gezeichnet werden, während näher gelegene Objekte, welche die zuerst genannten womöglich teilweise überdecken, erst später zu berücksichtigen sind. Der einfachste Ansatz zur Lösung dieses Problems ist der so genannte *Maleralgorithmus*, der genau wie gerade beschrieben vorgeht und im Hintergrund befindliche Objekte unter Umständen mehrfach überzeichnet[1]. Dazu müssen alle in der Szene vorhandenen Polygone abhängig von der Position des Betrachters zunächst der Tiefe nach sortiert werden, was sich als recht ineffizient erweist.

Ein anderer Ansatz, der auf einer prioritätensortierten Liste aufbaut, stammt von Schumaker et al. [SBGS69] und wurde von Fuchs et al. [FKN80] unter dem Begriff der *Binary Space Partitioning Trees (BSP-Trees)*, die im Mittelpunkt dieser Ausarbeitung stehen, weiter ausgebaut. BSP-Trees stellen eine Datenstruktur zur hierarchischen Unterteilung des Raums in Regionen dar und eignen sich damit gut zur Darstellung von dreidimensionalen Gebilden in Form von Polyedern.

In den folgenden Kapiteln möchten wir im Anschluss an eine kurze, formale Definition untersuchen, wie man BSP-Trees erzeugen kann und welche Besonderheiten es dabei zu beachten gilt. Anschließend wollen wir Mengenoperationen auf die entstehenden Bäume anwenden, indem wir einen BSP-Tree per Vereinigung, Durchschnitt oder Differenz mit einer sog. *Boundary Representation* verknüpfen. Außerdem stellen wir kurz ein Verfahren zur Umwandlung von *CSG-Bäumen* in BSP-Trees vor. Auf die mengentheoretische Verknüpfung eines BSP-Trees mit einem anderen BSP-Tree per Schnitt oder Vereinigung werden wir hier aus Platzgründen nicht weiter eingehen.

1.2 Begriffserläuterungen

1.2.1 Boundary Representations (B-Reps)

Bevor wir uns überlegen, wie wir geometrische Objekte unter Verwendung von BSP-Trees darstellen können, betrachten wir zunächst die uns am gebräuchlichsten erscheinende Art der Darstellung, die auf *Boundary Representations (B-Reps)* – also Flächenbegrenzungsmodellen – beruht. Darunter verstehen wir die Darstellung geometrischer Gebilde durch ihre Flächen (*faces*), Kanten (*edges*) und Eckpunkte (*vertices*). Im Wesentlichen lässt sich ein Polyeder nämlich in eine Reihe von aneinander angrenzenden Flächen unterteilen, welche wiederum durch jeweils paarweise adjazente Kanten beschrieben werden können. Die Kanten andererseits werden durch ihre Eckpunkte genau festgelegt.

[1] Daher auch der Name *Painter's algorithm*: Wie früher etwa in der Ölmalerei üblich, kann ein bestimmter Bildbereich mehrfach übermalt werden. Aufgrund der Deckkraft der Farbe sind darunter liegende Areale dann nicht mehr sichtbar. Auf das Beispiel des Malers bezogen ist das natürlich im gewissen Sinne Farb- und damit Geldverschwendung. Übertragen auf die Computergrafik bedeutet dies ein schlechteres Laufzeitverhalten des Algorithmus, weil der Betrachter die überzeichneten Gebiete eh nie zu Gesicht bekommen würde (sofern man Transparenzen außer Acht lässt).

Diese Art der Darstellung hat sich in der Praxis bewährt, weil sie intuitiv und flexibel ist. Intuitiv deshalb, weil wir die geometrischen Objekte in einzelne, niedriger-dimensionale Unterobjekte zerlegen können; es ergibt sich also eine Art Hierarchie. Mithilfe der sog. *konstruktiven Festkörpergeometrie*[2], also der Kombination geometrischer Objekte unter Verwendung von Mengenoperationen wie Schnitt und Vereinigung, können wir mit recht einfachen Mitteln aus grundlegenden Objekten kompliziertere Gebilde erschaffen, die sich auch wieder durch Boundary Representations darstellen lassen.

1.2.2 Regularisierte Mengenoperationen

Weil wir im Folgenden darauf zurückgreifen werden, möchten wir kurz den Begriff der regularisierten Mengenoperationen erläutern. Aus der Mengenlehre bereits bekannt sind uns die Begriffe der Vereinigung, des Durchschnitts und der Differenz. Betrachten wir geometrische Objekte als Menge von Punkten, so lassen sich auf diese Objekte – wie bei der konstruktiven Festkörpergeometrie – Mengenoperationen anwenden. Nun kann es jedoch passieren, dass etwa beim Schnitt zweier Objekte ein Objekt niedrigerer Dimension entsteht. Schneiden wir etwa Würfel miteinander, die nur eine Begrenzungsfläche gemein haben, so ist der Schnitt eben jene Fläche (vgl. Abbildung 1.1).

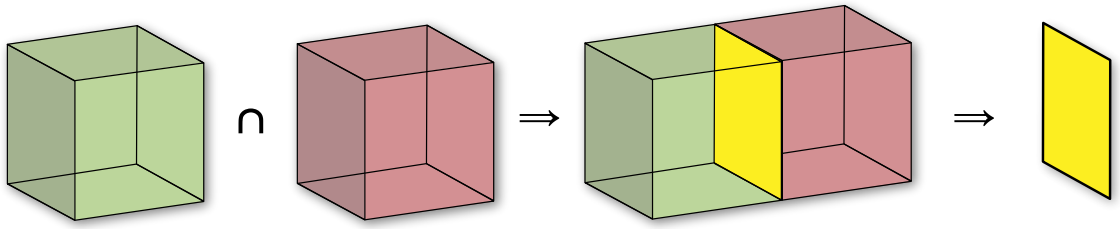


Abbildung 1.1: Der *nicht-regularisierte* Schnitt zweier aneinander angrenzender Würfel liefert ein planares Polygon. Das ist bei der konstruktiven Festkörpergeometrie im Allgemeinen nicht erwünscht, weshalb man hier reguläre Mengen einsetzt.

Die Fläche ist im Gegensatz zu den Würfeln nun aber nicht mehr drei-, sondern nur noch zweidimensional. Gerade im Bereich der dreidimensionalen geometrischen Modellierung möchten wir derartige Phänomene aber verhindern; wir sind nicht an einzelnen Flächen interessiert, die „zusammenhanglos“ im Raum schweben, sondern an dreidimensionalen Festkörpern. Deshalb führen wir den Begriff der regulären Menge *(regular set)* ein [RT78]. Unter einer regulären Menge verstehen wir die abgeschlossene Hülle *(closure)* ihres Inneren (d. h. M regulär $\Rightarrow S = cl(int\ S)$).

Wir bestimmen also zunächst das Innere der Menge, indem wir die Randpunkte entfernen. Im Beispiel des Schnitts zweier aneinander angrenzender Würfel würden wir damit die Fläche (also das zweidimensionale Polygon) entfernen, weil sich diese nicht im Inneren der Menge befindet; es ergibt sich die leere Menge. Anschließend führen wir den mengentheoretischen Abschluss durch. Im vorliegenden Beispiel führt dies zu keiner Änderung, weil der Abschluss der leeren Menge stets wieder die leere Menge ergibt. Schneiden wir also zwei Würfel wie in der Abbildung dargestellt, so ist der *regularisierte(!)* Schnitt dieser beiden Objekte die leere Menge.

In unseren weiteren Überlegungen werden wir daher häufig auf die folgenden, regularisierten Varianten des *Komplements*, der *Vereinigung*, des *Durchschnitts* und der *Differenz* zurückgreifen [RT78]:

$$\mathsf{C}^\star A = (cl(int(\mathsf{C}\ A));\ A \cup^\star B = cl(int(A \cup B));\ A \cap^\star B = cl(int(A \cap B));\ A \setminus^\star B = cl(int(A \setminus B)).$$

[2] *Constructive Solid Geometry*, kurz *CSG*

2 Grundlagen von BSP-Trees

2.1 Definition: Binary Space Partitioning Tree

Unter einem *Binary Space Partitioning Tree (BSP-Tree)* verstehen wir einen **binären Baum**, dessen **innere Knoten** *Hyperebenen* in einem d-dimensionalen Raum sind und den Raum in Regionen unterteilen. Wir gehen dabei an der Wurzel von einer Anfangsregion aus, die üblicherweise dem ganzen d-Raum entspricht (bspw. dem $\mathbb{R}^3$). Vorausgesetzt, es liegt nicht der triviale Fall vor, und die Wurzel besitzt Kinder, so unterteilt die in der Wurzel befindliche Hyperebene den Raum in zwei Halbräume[1]. Begeben wir uns in die Kindknoten, so führt jeder Kindknoten, der kein Blatt darstellt, zu einer **weiteren Unterteilung** der übergeordneten Region, sodass wir durch wiederholtes Zerteilen unserer Ausgangsregion jeden beliebigen, linear begrenzten d-dimensionalen Körper darstellen können (im Dreidimensionalen etwa Polyeder[2]). Die Blätter eines BSP-Trees – die zu keiner weiteren Unterteilung des Raums führen –, bezeichnen wir als **Zellen**. [TN87]

Beschreiben wir die einem inneren Knoten zugehörige Hyperebene H durch ihre Hyperebenengleichung in Normalenform

$$H = \{(x_1, \ldots, x_d) \mid a_1 x_1 + \ldots + a_d x_d - b = 0\},$$

so unterteilt diese Hyperebene den d-dimensionalen Raum in zwei Halbräume

$$H^- = \{(x_1, \ldots, x_d) \mid a_1 x_1 + \ldots + a_d x_d - b < 0\}$$

und

$$H^+ = \{(x_1, \ldots, x_d) \mid a_1 x_1 + \ldots + a_d x_d - b > 0\},$$

wobei wir den erstgenannten als linken (negativen, hinteren) und den letztgenannten als rechten (positiven, vorderen) Halbraum von H bezeichnen [TN87]. Der rechte (positive) Halbraum befindet sich dabei in Richtung der Normalen von H.

Jedem Knoten v unseres binären Baums können wir eine Region $R(v)$ zuordnen, die sich durch „Abschreiten“ der Kanten des Baums – also wiederholte Unterteilung der Ausgangsregion, bis wir den Knoten v erreichen –, ergibt. Ordnen wir jeder Kante e ihren zugehörigen Halbraum $HS(e)$ zu[3], so ergibt sich die zu einem Knoten v gehörende Region $R(v)$ durch Bildung der Schnittmenge aller Halbräume, die auf dem Pfad von der Wurzel zu v liegen [TN87], also

$$R(v) = \bigcap_{e \in E(v)} HS(e).$$

Um BSP-Trees zur Darstellung von Polytopen[4] zu verwenden, müssen wir festlegen, welche Zellen (also Blätter) des Baums das Innere des Polytops und welche das Äußere beschreiben. Aus diesem

[1] Die Halbräume werden durch die beiden Kanten vom Mutter- zu den Kindknoten repräsentiert.

[2] Im Vergleich zu den **B-Reps** sind **BSP-Trees** hier sogar ausdrucksmächtiger: B-Reps besitzen nämlich stets eine endliche Begrenzung. Weil BSP-Trees zunächst aber nur den Raum unterteilen, können wir mit ihrer Hilfe auch geometrische Objekte darstellen, die z. B. zu einer Seite hin bis ins Unendliche reichen. Unter Verwendung von BSP-Trees kann man also unendliche Räume oder Halbräume darstellen, was mit B-Reps ohne weiteres nicht möglich ist.

[3] Abhängig davon, ob die jeweilige Kante in einen linken oder rechten Kindknoten führt, handelt es sich hierbei also entweder um H^- oder H^+.

[4] Ein (lineares) Polytop bezeichnet die Verallgemeinerung des Begriffs „Polyeder“ bezogen auf den beliebigen d-dimensionalen Fall.

Grund weisen wir jeder Zelle ein Attribut *Mitgliedschaft* $\in$ { *innen, außen* } zu[5]. Wie genau wir den Wert dieses Attributs ermitteln können, werden wir später erläutern.

In den folgenden Ausführungen beschränken wir uns der Einfachheit halber teilweise auf den zwei- bzw. dreidimensionalen Fall. Es sei jedoch gesagt, dass sich die hier vorgestellten Verfahren prinzipiell auf jeden beliebig-dimensionalen Fall übertragen lassen.

2.2 Punktprobe-Algorithmus

Nachdem wir den Zellen das Attribut *Mitgliedschaft* zugeordnet haben, können wir einen einfachen, rekursiven Algorithmus formulieren, mit dessen Hilfe wir eine Aussage darüber treffen können, ob ein beliebiger Punkt *innerhalb* einer gegebenen regulären Menge M, *außerhalb* oder *auf dem Rand* dieser Menge liegt. Dabei repräsentieren wir die Menge M durch einen BSP-Tree.

```
Prozedur KlassifizierePunkt(p : Punkt; v : BspTreeKnoten)
  Rückgabe : Aufzählungstyp { innen, außen, aufDemRand }

  Falls v ein Blatt ist
    Gib Mitgliedschaft(v) zurück    /* Liefert 'innen' oder 'außen' zurück */
  Sonst
    /* b ist Abstand der Hyperebene zum Ursprung (falls H in Hessescher Normalform) */
    Berechne d := ⟨ Normalenvektor von H_v, p ⟩ − b    /* Skalarprodukt */

    Falls d < 0
      Gib KlassifizierePunkt(p, linkes Kind von v) zurück
    Sonst, falls d > 0
      Gib KlassifizierePunkt(p, rechtes Kind von v) zurück
    Sonst
      Setze l := KlassifizierePunkt(p, linkes Kind von v)
      Setze r := KlassifizierePunkt(p, rechtes Kind von v)
      Falls l == r
        Gib l zurück
      Sonst
        Gib aufDemRand zurück
```

Listing 1: Algorithmus zur Klassifizierung eines beliebigen Punktes in Bezug auf eine gegebene reguläre Menge M, die durch einen BSP-Tree dargestellt wird. [TN87]

Der in Listing 1 dargestellte Algorithmus arbeitet sich dabei von der Wurzel ausgehend durch den Baum durch, bis das Blatt (also die Region) gefunden ist, in dem sich der angegebene Punkt befindet. Befinden wir uns an einem inneren Knoten, können wir den nächsten aufzusuchenden Kindknoten bestimmen, indem wir durch Bilden des Skalarprodukts die Lagebeziehung des Punktes zur im Knoten gespeicherten Hyperebene bestimmen. Im Sonderfall $d = 0$ befindet sich der Punkt *auf* der jeweiligen Hyperebene[6]. In diesem Fall müssen wir *beide* Kindknoten betrachten und nach weiteren, tiefer geschachtelten Hyperebenen suchen, auf denen der Punkt zusätzlich noch liegen könnte. Unser Ziel hierbei ist es, zu entscheiden, ob der Punkt zwischen zwei gleichartigen Zellen (z. B. zwei Zellen mit dem Wert *innen*) liegt, oder zwischen zwei verschieden gearteten Zellen. Im letztgenannten Fall gehört der Punkt nämlich zu den Randpunkten der Menge, und wir geben *aufDemRand* zurück.

[5] In den Abbildungen verwenden wir aus Gründen der Übersichtlichkeit die englischen Entsprechungen *in* bzw. *out*.

[6] In der Praxis wird der Fall $d = 0$ aufgrund der Fließkomma-Ungenauigkeit natürlich so gut wie nie eintreten. Deshalb macht es Sinn, den Hyperebenen eine gewisse „Dicke“ zuzuweisen. Alle Punkte, die sich innerhalb einer bestimmten ε-Umgebung der Hyperebene befinden, liegen also „auf“ ihr.

2.2.1 Laufzeitverhalten

Gemäß [Kal82] lässt sich die Punktklassifizierung im dreidimensionalen Raum mithilfe von **Boundary Representations** in einer Laufzeit von $O(n)$ durchführen, wobei n die Anzahl der Flächen der B-Rep angibt. Wie von Naylor [Nay81] gezeigt, besitzt der oben aufgeführte Algorithmus für **BSP-Trees** im allgemeinen Fall ebenfalls lineare Laufzeit. Handelt es sich beim verwendeten BSP-Tree jedoch um einen *balancierten* Baum der Größenordnung $O(n)$, so lässt sich das Problem sogar in logarithmischer Laufzeit, also $O(log\ n)$, lösen. Damit ist etwa in Simulationen oder Computerspielen eine Möglichkeit zur schnellen Kollisionserkennung gegeben. Es ist jedoch nicht für jede beliebige Menge M möglich, sie in einen solchen balancierten Baum umzuwandeln [TN87].

2.3 Erzeugung von BSP-Trees aus Boundary Representations

Als nächstes interessiert uns natürlich, wie wir überhaupt BSP-Trees erzeugen können. Am naheliegendsten ist hier die Erzeugung aus einem gegebenen Flächenbegrenzungsmodell, also einer Boundary Representation. Der in Listing 2 dargestellte Algorithmus erwartet dazu als Eingabe eine Menge F von Flächen, die ein einzelnes oder eine Menge von disjunkten Polytopen darstellen, welches bzw. welche wir in einen BSP-Tree umwandeln wollen.

Nun wählt der Algorithmus für jeden zu erzeugenden Knoten v zunächst eine Teilungsebene H_v aus und **unterteilt** dann die gegebene Menge von Flächen F anhand dieser Hyperebene. Die Hyperebene kann dabei prinzipiell beliebig gewählt werden. Weil wir jedoch bezwecken möchten, dass alle Randpunkte des ursprünglichen Polytops auf den Teilungsebenen des BSP-Trees liegen, ist es ratsam, die Hyperebene stets so zu wählen, dass sie eine Fläche aus F enthält. Weil hier häufig mehrere Möglichkeiten existieren, ist der Einsatz einer geeigneten Heuristik ratsam [TN87].

Da es dazu kommen kann, dass die soeben erzeugte Teilungsebene H_v bestimmte Flächen aus der Menge F **schneidet**, müssen wir diese Flächen ihrerseits aufteilen in Teilflächen, die oberhalb, unterhalb oder innerhalb der Teilungsebene liegen[7]. Die Teilflächen, die innerhalb der Teilungsebene liegen, werden in einer Menge $F_{in\ Ebene}$ abgelegt und später ihrer Orientierung entsprechend im gerade erzeugten BSP-Tree-Knoten v gespeichert[8]; die anderen beiden Mengen F_{links} und F_{rechts} werden durch wiederholte rekursive Aufrufe weiter unterteilt, bis schließlich alle Flächen auf der gleichen Seite einer Teilungsebene (oder auf deren Rand) liegen. In einem solchen Fall wissen wir, dass die erzeugte Region **homogen** ist, d. h. es gibt keine weitere Begrenzungsfläche, welche die Region in unterschiedlich geartete Gebiete mit den Werten *innen* und *außen* aufteilen könnte. Wir können also eine Zelle erzeugen, welcher wir genau einen der gerade erwähnten Attributwerte zuweisen.

Um konkret zu bestimmen, ob eine Zelle den Wert *innen* oder *außen* trägt, genügt ein Vergleich des Normalenvektors von H_v mit den Normalenvektoren der Flächen $F_{in\ Ebene}$ (welche die Begrenzungsfläche bilden). Weisen die Vektoren die gleiche Orientierung auf, so handelt es sich um eine äußere Zelle, ansonsten liegt eine innere Zelle vor[9] [TN87].

---

[7] Zur Unterteilung einer Fläche f vermittels einer Teilungsebene H bilden wir die folgenden drei Mengen, wobei mit *int* das Innere bezüglich derjenigen Hyperebene H_f gemeint ist, die f enthält:

$$f^+ = cl(H^+ \cap int\ f),\ f^- = cl(H^- \cap int\ f),\ f^0 = cl(int(H \cap int\ f)).$$

Eine oder zwei der aufgeführten Mengen sind stets leer. Hierbei repräsentiert f^0 diejenige Teilfläche, die innerhalb der Teilungsebene H liegt (vorausgesetzt, es handelt sich dabei tatsächlich noch um eine *echte* Fläche, und keine zu einer Kante degenerierten Fläche – ansonsten gilt durch Bildung der abgeschlossenen Hülle des Inneren hier $f^0 = \emptyset$).

[8] Es macht hier Sinn, die mit der Teilungsebene koinzidierenden Flächen ihrer Orientierung entsprechend in zwei disjunkte Mengen aufzuteilen, die jeweils Flächen mit der gleichen Orientierung wie die Teilungsebene bzw. entgegengesetzter Orientierung beinhalten.

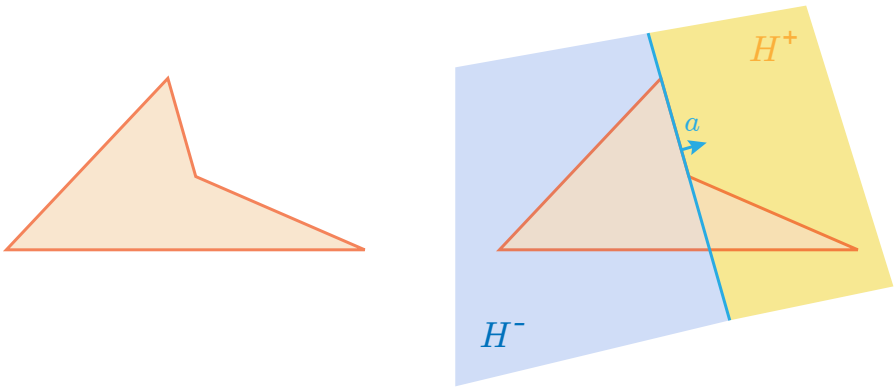
[9] Dies könnte die Frage aufwerfen, wie man zu verfahren hat, wenn die Menge $F_{in\ Ebene}$ unterschiedlich orientierte Flächen enthält. Dieser Fall kann jedoch nur eintreten, wenn sich zwischen den beiden Flächen noch mindestens eine weitere Fläche befindet, deren Hyperebene die ursprüngliche Hyperebene schneidet. Man müsste hier also noch weiter unterteilen und stünde noch gar nicht vor der Entscheidung, welchen Attributwert man den Zellen zuzuweisen hat.

```
Prozedur ErzeugeBspTree(F : Menge von Flächen)
  Rückgabe : BspTreeKnoten

  H := Erzeuge eine Hyperebene, die eine (beliebige) Fläche aus F beinhaltet
  ergebnisKnoten := Erzeuge neuen BspTreeKnoten mit H als Teilungsebene

  Unterteile die Menge F anhand von H in drei Mengen F_links, F_rechts, F_in_Ebene

  /* Wir unterscheiden in der folgenden Zeile zwischen Flächen, die in Richtung der Normalen
     der Teilungsebene zeigen, und Flächen, welche genau umgekehrt orientiert sind. Beide
     werden in unterschiedlichen Listen innerhalb des ergebnisKnotens abgespeichert. */
  Speichere die in F_in_Ebene enthaltenen Flächen in der
        entsprechenden Flächenliste des ergebnisKnotens

  Falls F_links leer ist /* Ist der linke Kindknoten eine Zelle (Blatt)? */
    Falls F_in_Ebene die gleiche Orientierung wie H besitzt
      ergebnisKnoten.linkesKind := 'innen'
    Sonst
      ergebnisKnoten.linkesKind := 'außen'
  Sonst
    ergebnisKnoten.linkesKind := ErzeugeBspTree(F_links)

  Falls F_rechts leer ist /* Ist der rechte Kindknoten eine Zelle (Blatt)? */
    Falls F_in_Ebene die gleiche Orientierung wie H besitzt
      ergebnisKnoten.rechtesKind := 'außen'
    Sonst
      ergebnisKnoten.rechtesKind := 'innen'
  Sonst
    ergebnisKnoten.rechtesKind := ErzeugeBspTree(F_rechts)

  Gib ergebnisKnoten zurück
```

Listing 2: Algorithmus zur Erzeugung eines BSP-Trees aus einer Boundary Representation [TN87]

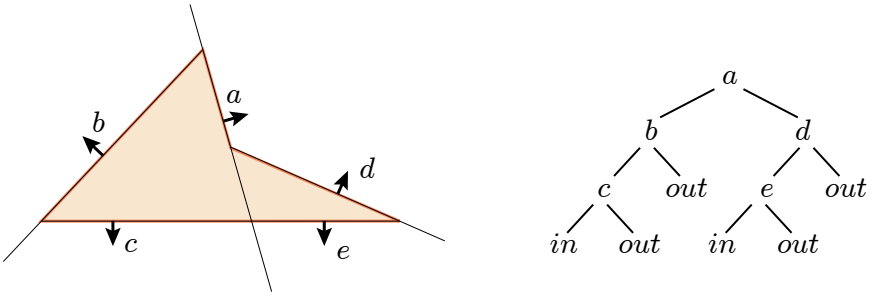
Wählen wir die Teilungsebenen nicht willkürlich, sondern erzeugen sie unmittelbar aus den Hyperebenengleichungen der Flächen, sodass die Normalenvektoren von Teilungsebenen und Flächen immer in die gleiche Richtung zeigen, so lässt sich das Problem sogar noch einfacher lösen. In diesem Fall kann man nämlich zeigen, dass ein linkes Blatt immer eine innere Zelle und ein rechtes Blatt stets eine äußere Zelle sein muss [TN87], vorausgesetzt, die Flächennormalen des ursprünglichen Polytops zeigen alle nach außen. Dabei gehen wir von unserer Definition in 2.1 aus, die besagt, dass eine nach links führende Kante jeweils den hinteren Halbraum und eine nach rechts führende Kante den vorderen Halbraum – bezogen auf die Teilungsebene – repräsentiert.

Es sei angemerkt, dass die Menge von Flächen F, die dem Algorithmus als Eingabe dient, nicht notwendigerweise ein konvexes Polytop – oder eine disjunkte Menge konvexer Polytope – darstellen muss. Interessanterweise, und diesen Vorteil sollte man nicht unterschätzen, führt die Anwendung des Algorithmus nämlich zu einer Zerlegung etwaiger *konkaver* Objekte in **konvexe** Regionen. Das ist in vielerlei Hinsicht hilfreich; unter anderem gestattet dies eine relativ unkomplizierte Kollisionserkennung, bei der man ohne die Notwendigkeit eines Punkt-in-Polygon-Tests (in Bezug auf den $\mathbb{R}^3$) auskommt. Außerdem eignet sich ein mithilfe von BSP-Trees unterteilter Raum damit zum Beispiel als Eingabe für portalbasierte Renderverfahren, um eine automatische Generierung von Portalen zu ermöglichen [TS91].



(a) Umzuwandelnde B-Rep

(b) Erster Unterteilungsschritt



(c) Endgültige Partitionierung

(d) Zugehöriger BSP-Tree

Abbildung 2.1: Erzeugung eines BSP-Trees aus einer Boundary Representation. In (a) ist das durch eine B-Rep spezifizierte Ausgangsobjekt, ein Polygon, zu sehen. Nun wird das Objekt Schritt für Schritt durch Wahl geeigneter Hyperebenen in eine Reihe homogener Regionen unterteilt. (b) zeigt, wie man die zur Wurzel des Baums gehörende erste Hyperebene beispielsweise wählen könnte. Eine mögliche komplette Partitionierung des Polygons ist in (c) zu sehen, Abbildung (d) zeigt den zugehörigen BSP-Tree.

2.4 Einfügen von Flächen in BSP-Trees

Eine weitere Möglichkeit zur Erzeugung eines BSP-Trees aus einer Boundary Representation besteht darin, die einzelnen Flächen des geometrischen Modells Schritt für Schritt zum BSP-Tree hinzuzufügen. Ausgehend von einem trivialen BSP-Tree, der nur aus einer Zelle besteht, kann eine Fläche f zum Baum hinzugefügt werden, indem man die Fläche nach und nach im Baum „absinken“ lässt. Dazu teilt man die Fläche für jeden inneren Knoten v – wie in Abschnitt 2.3 beschrieben – an der Hyperebene H_v zunächst in drei Mengen auf, welche die Teilflächen ober-, unter- und innerhalb der Hyperebene repräsentieren. Die Teilflächen *innerhalb* der Hyperebene werden im Knoten v gespeichert, die anderen beiden Mengen werden an den linken bzw. rechten Kindknoten weitergereicht. Dieser Vorgang wird jeweils solange fortgesetzt, bis ein Blatt erreicht wird. Das Blatt ersetzen wir durch einen neuen Knoten v', der eine auf der durchgereichten Teilfläche basierende Teilungsebene besitzt und dessen Kinder nun Zellen sind, die jeweils mit *innen* bzw. *außen* beschriftet sind [TN87].

3 Einfache Mengenoperationen auf BSP-Trees

Wir wissen nun, wie wir BSP-Trees aus Flächenbegrenzungsmodellen erzeugen können. Interessant werden die BSP-Trees jedoch erst dann, wenn wir Mengenoperationen einsetzen, um verschiedene geometrische Objekte miteinander zu kombinieren. Ähnlich wie bei der konstruktiven Festkörpergeometrie kann man hier beispielsweise bestimmte Basisobjekte miteinander verknüpfen, um dadurch kompliziertere Körper zu erschaffen. So lässt sich etwa ein Tisch aus einer Tischplatte (in Form eines Quaders) und vier Tischbeinen (also Zylindern) konstruieren.

In den folgenden zwei Abschnitten werden wir uns mit dieser Thematik auseinander setzen und zwei Verfahren vorstellen, die es uns gestatten, geometrische Objekte mittels Mengenoperationen zu verknüpfen, wobei das Resultat natürlich stets ein BSP-Tree ist.

3.1 BSP-Tree $\langle$op$\rangle$ Boundary Representation $\rightarrow$ BSP-Tree

Das erste Verfahren, das wir erläutern möchten, kombiniert einen BSP-Tree mit einer Boundary Representation unter Anwendung einer der in Abschnitt 1.2.2 vorgestellten **regulären** Mengenoperationen *Vereinigung*, *Durchschnitt* und *Differenz*. Die Anwendung des unären Komplementoperators auf einen BSP-Tree ist natürlich auch möglich, stellt jedoch keine große Herausforderung dar: Um einen BSP-Tree zu komplementieren, müssen wir lediglich alle inneren Zellen in äußere umwandeln, und umgekehrt; außerdem muss die Orientierung aller in den inneren Knoten gespeicherten Begrenzungsflächen umgekehrt werden [TN87]. Bei B-Reps erfolgt die Komplementbildung, indem wir sämtliche Flächennormalen invertieren.

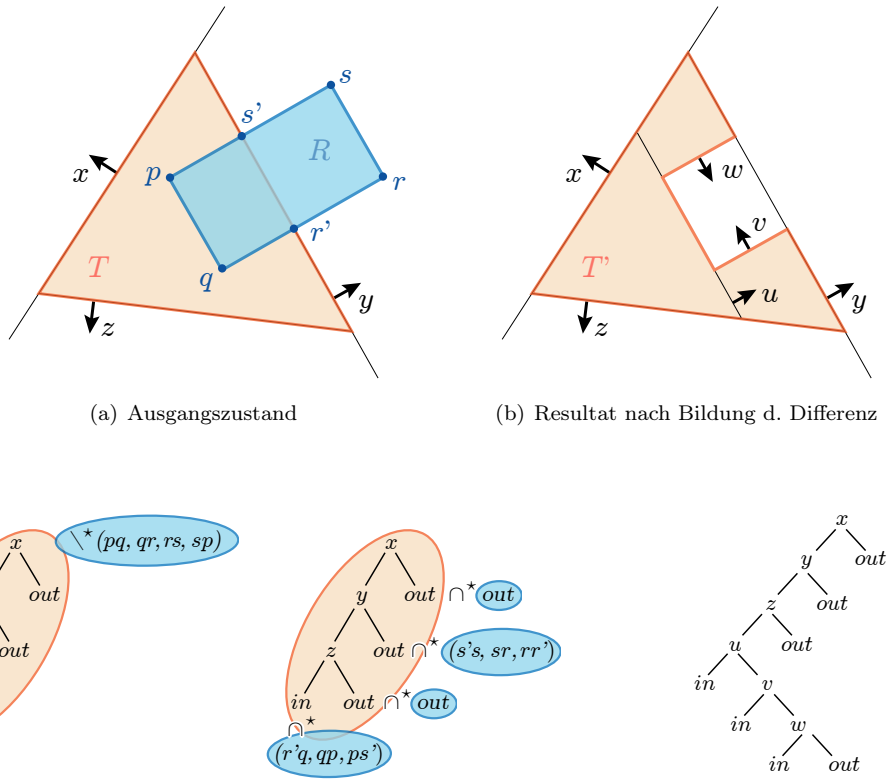
Gehen wir nun also davon aus, dass ein BSP-Tree $\widehat{T}$ gegeben ist, der ein Polytop T repräsentiert, sowie eine zu einem Polytop R gehörende Boundary-Representation $\widehat{R}$. Unser Ziel ist es, die beiden geometrischen Objekte nach dem Muster T *op* R bzw. R *op* T miteinander zu verknüpfen. Im Falle der Bildung der mengentheoretischen Differenz können wir die Operation $M_1 \setminus^\star M_2$ in die äquivalente Form $M_1 \cap^\star \mathsf{C}^\star M_2$ bringen und somit auf Komplement und Schnitt der Mengen zurückführen.

Wir fügen nun schrittweise alle Flächen unseres Polytops R in den BSP-Tree $\widehat{T}$ ein. Dazu nutzen wir das in 2.4 vorgestellte Verfahren zum Einfügen von Flächen in BSP-Trees, mit dessen Hilfe wir die Flächen im Baum „absinken“ lassen und jeweils an den Teilungsebenen des Baumes aufteilen. Tritt dabei der Fall ein, dass an einem Knoten v von $\widehat{T}$ alle eingefügten Flächen auf ein- und derselben Seite der Teilungsebene landen, so handelt es sich bei der jeweils anderen Seite um eine Region, die *homogen bezüglich* R ist, d. h. im Polytop R befinden sich in der entsprechenden Region keine weiteren Begrenzungsflächen [TN87]. Angenommen also, wir fügen alle Flächen F von R in $\widehat{T}$ ein und es existiert ein Knoten v, dessen Teilungsebene H_v die ihn erreichende Teilmenge von Flächen $F' \subseteq F$ weiter unterteilt; außerdem gelte $F_{links} = F'$ und $F_{rechts} = \emptyset$. Dann ist die Region rechts der Teilungsebene homogen bezüglich R (aber nicht notwendigerweise bezüglich T, denn der rechte Kindknoten von v muss nicht zwingend eine Zelle sein!). Es existieren nun zwei mögliche Optionen: Entweder, der (hier) rechte Teilbaum bleibt unverändert, oder aber der dem Teilbaum zugrunde liegende Mengenausdruck wird unter Anwendung gewisser Ersetzungsregeln vereinfacht, auf die wir unten noch eingehen werden.

Um diese Regeln überhaupt erst anwenden zu können, müssen wir zuvor noch bestimmen, ob es sich bei der jeweiligen Region um eine Teilmenge des Inneren oder des Äußeren bezüglich des durch die B-Rep $\widehat{R}$ dargestellten Polytops handelt; dies kann sich insofern als schwierig gestalten, als dass wir –

wie hier im Beispiel des rechten Halbraums H_v^+ – womöglich über keine Flächen verfügen, deren Normalen wir mit der Normalen der Teilungsebene vergleichen könnten (wie dies bei der Baumerzeugung in 2.3 möglich gewesen ist). Aus diesem Grund müssen wir uns ein zusätzliches Entscheidungsverfahren überlegen, auf das wir am Ende dieses Kapitels kurz eingehen werden.

Erreicht eine (Teil-)Fläche von R schließlich ein Blatt, so erzeugen wir in diesem Fall im Gegensatz zu dem in 2.4 vorgestellten Einfügeverfahren nicht sofort einen neuen Knoten, sondern verknüpfen die Werte beider Knoten durch die angegebene Mengenoperation *op* miteinander. Den entstehenden Mengenausdruck können wir nun auswerten, weil der Operand T hier homogen ist (denn wir befinden uns in einem Blatt) – es kann also zu keiner weiteren Zerteilung der jeweiligen Fläche(n) von R kommen. Die sich in den Blättern ergebenden Ausdrücke können anhand der bereits erwähnten Ersetzungsregeln, die in Tabelle 3.1 aufgeführt sind, vereinfacht werden[1]. [TN87]



(a) Ausgangszustand

(b) Resultat nach Bildung d. Differenz

(c) BSP-Tree u. B-Rep zu Beginn

(d) (qp, rq, sr, ps) klassifiziert

(e) Baum nach Vereinfachung

Abbildung 3.1: Mengenoperationen zwischen BSP-Trees und B-Reps. Hier wird die Differenz zwischen dem Polygon T, das durch einen BSP-Tree gegeben ist, und einem durch eine B-Rep spezifizierten Polygon R gebildet. Die Differenz kann dabei durch Komplement und Durchschnitt ausgedrückt werden. Zunächst werden alle Flächen von R in den BSP-Tree eingefügt (c), anschließend lassen wir die Flächen im Baum absinken (d). Zum Schluss können die konkreten Werte der Zellattribute durch eine Reihe von Vereinfachungsregeln bestimmt werden (e).

[1] Mithilfe dieser Vereinfachungsregeln lassen sich in manchen Fällen ganze Teilausdrücke „wegkürzen“, die beliebig kompliziert sein können. Die Idee, die sich dahinter verbirgt, wird unter anderem in [Til84] erläutert. Das wesentliche Ziel bei der Anwendung derartiger Vereinfachungsregeln ist die Vermeidung von Redundanzen, indem z. B. unnötige Flächen, die sich im Inneren eines Objektes befinden und deshalb sowieso keinen Einfluss auf die äußere Darstellung haben, entfernt werden.

Tabelle 3.1: Vereinfachungsregeln für Mengenausdrücke. M sei eine beliebige reguläre Menge, *innen* und *außen* sind Zellattributwerte des BSP-Trees. [TN87]

op	Operand 1	Operand 2	Ergebnis
$\cup^\star$	M	*innen*	*innen*
	M	*außen*	M
	innen	M	*innen*
	außen	M	M

op	Operand 1	Operand 2	Ergebnis
$\cap^\star$	M	*innen*	M
	M	*außen*	*außen*
	innen	M	M
	außen	M	*außen*

op	Operand 1	Operand 2	Ergebnis
$\setminus^\star$	M	*innen*	*außen*
	M	*außen*	M
	innen	M	$\complement^\star M$
	außen	M	*außen*

3.2 Constructive Solid Geometry (CSG) mit B-Reps → BSP-Tree

Eine weitere Möglichkeit, geometrische Körper durch Mengenoperationen miteinander zu verknüpfen und anschließend als BSP-Tree zu repräsentieren, ist die Umwandlung von CSG-Bäumen in Boundary Representations. Unter einem **CSG-Baum** verstehen wir eine hierarchische, binäre Baumstruktur, deren Blätter instanziierte geometrische Primitive (wie z. B. Würfel, Kugeln, Zylinder) darstellen und deren innere Knoten regularisierte Mengenoperationen oder Bewegungen repräsentieren [Req80]. Da CSG-Bäume zum Zeichnen zunächst ausgewertet werden müssen, macht es Sinn, sie in eine explizitere Form zu verwandeln, also beispielsweise in eine B-Rep oder einen BSP-Tree [JLO96]. Naylor und Thibault [TN87] beschreiben ein Verfahren zur Lösung des letztgenannten Problems, auf das wir hier kurz eingehen möchten.

Um einen CSG-Baum auszuwerten und in einen BSP-Tree umzuwandeln, wählen wir zunächst eine Hyperebene, anhand der wir den CSG-Baum unterteilen können[2]. Die Hyperebene bildet dann die Wurzel des neu erzeugten BSP-Trees. Die beiden Teilbäume des CSG-Baums, die wir durch die Unterteilung erhalten, werden als linker bzw. rechter Kindknoten an die Wurzel angehängt. Anschließend wählen wir für jeden der Teilbäume erneut eine Hyperebene, um ihn damit aufzuteilen. Das Verfahren wird rekursiv so lange fortgesetzt, bis der gerade aktuelle CSG-Teilbaum einen trivialen BSP-Tree – d. h. eine Zelle – repräsentiert [TN87]. Dabei kann erneut auf die zuvor erwähnten Vereinfachungsregeln (Tabelle 3.1) zurückgegriffen werden.

Zur Wahl der Teilungsebenen macht es – wie schon in 2.3 erwähnt – Sinn, eine geeignete Heuristik zu nutzen. In [TN87] werden drei mögliche Heuristiken vorgestellt. Generell muss man hier zwischen der Größe des BSP-Trees auf der einen Seite und der Anzahl der Flächen, die zerteilt werden müssen, auf der anderen Seite abwägen.

3.3 Bemerkungen

Bestimmung der Zellzugehörigkeit

In den beiden hier vorgestellten Verfahren, die Mengenoperationen einsetzen, wird eine Möglichkeit zur Bestimmung der Zellzugehörigkeit vorausgesetzt, d. h. wir müssen auf irgendeine Weise bestimmen, ob

[2] Zur Unterteilung erzeugen wir zwei Kopien des ursprünglichen CSG-Baums. In jeder Kopie ersetzen wir nun die enthaltenen Flächen durch genau die Untermenge eben dieser Flächen, die im jeweiligen Halbraum H^- bzw. H^+ liegt.

es sich bei einem Blatt um eine *innen-* oder *außenliegende* Region bezüglich des Ausgangsobjekts (B-Rep/CSG-Baum) handelt. Im Algorithmus zur Erzeugung von BSP-Trees aus Boundary Representations, den wir in Abschnitt 2 vorgestellt haben, konnten wir dazu die Orientierung der Flächennormalen mit der Orientierung der Normalen der Teilungsebene vergleichen. In 3.1 und 3.2 ist dies nun jedoch nicht mehr ohne weiteres möglich, da es hier auch Regionen geben kann, die keine Flächen beinhalten.

Zur Bestimmung, ob es sich nun konkret um eine innere oder eine äußere Zelle handelt, stellt [TN87] daher einen sog. **In/Out-Test** vor. Der diesem Verfahren zugrunde liegende Algorithmus zieht dazu Rückschlüsse aus der Lage desjenigen Eckpunktes (*vertex*), der sich am nähesten zur Teilungsebene befindet – und zwar auf der gegenüberliegenden Seite der Hyperebene, denn die gegenwärtige, zu klassifizierende Region ist ja homogen und enthält deswegen keine weiteren Begrenzungsflächen.

Eine andere Möglichkeit besteht darin, von einem Punkt auf der Teilungsebene ausgehend einen Strahl auszusenden, um die nächstgelegene Fläche zu bestimmen und anhand deren Orientierung eine Klassifizierung der Zelle in *innen* oder *außen* vorzunehmen. Dieses Verfahren, welches man als *Raycasting-Test* bezeichnet, wird, wie [TN87] anmerkt, in [LTH86] genauer beschrieben.

Erzeugung von Begrenzungsflächen

Durch Anwendung der beiden mengenbasierten Algorithmen kann es dazu kommen, dass die Begrenzungsflächen des ursprünglichen geometrischen Modells „verloren“ gehen: Zwar werden in den inneren Knoten des erzeugten BSP-Trees bestimmte Flächen gespeichert – nämlich genau die Flächen, die mit den Teilungsebenen koinzidieren –, diese müssen jedoch nicht mehr unbedingt mit dem Rand des geometrischen Modells übereinstimmen [TN87]. Möchte man den erzeugten BSP-Tree lediglich zum Zwecke des Raytracings oder zur Punktklassifizierung – etwa mithilfe des in 2.2 beschriebenen Algorithmus – nutzen, sind keine weiteren Vorkehrungen nötig.

Will man den erzeugten BSP-Tree jedoch direkt zum Zeichnen von Polygonen verwenden, muss zunächst der Rand des Körpers wiederhergestellt werden. Hier besteht entweder die Möglichkeit, sämtliche in den inneren Knoten gespeicherten Flächen komplett zu verwerfen und den Rand neu zu erzeugen, oder die Begrenzungsflächen anhand der vollzogenen Mengenoperationen zu rekonstruieren. Dazu werden die in den inneren Knoten gespeicherten Flächen durch Absinkenlassen im BSP-Tree klassifiziert und ggf. miteinander verschmolzen [TN87].

4 Fazit

In dieser Ausarbeitung haben wir uns mit den *Binary Space Partitioning Trees*, oder kurz *BSP-Trees* – einer Datenstruktur zur hierarchischen Unterteilung des Raumes in konvexe Regionen – beschäftigt und einen Algorithmus zur Erzeugung von BSP-Trees kennengelernt. Außerdem haben wir untersucht, wie man Mengenoperationen in Verbindung mit BSP-Trees einsetzen kann, indem man eine *Boundary Representation* mit einem BSP-Tree verknüpft oder einen *CSG-Baum* in einen solchen Baum transformiert.

Während BSP-Trees ursprünglich vor allem zur effizienteren Lösung des Polygon-Sortierproblems entwickelt worden waren, wurden sie in diesem Aufgabenbereich in den neunziger Jahren, als Videospeicher und Grafik-Hardwarebeschleunigung allgemein zunehmend billiger wurden, weitestgehend durch den 1974 von Catmull [Cat74] bzw. Straßer [Str74] entworfenen Z-Buffer-Algorithmus verdrängt. Dennoch kamen BSP-Trees in einigen der in den letzten Jahren entwickelten Computerspielen zum Einsatz (z. B. in der *Quake-3*-Engine oder in der *Doom*-Reihe), oftmals kombiniert mit *Potentially Visible Set (PVS)*-Algorithmen, die ein effizientes Entfernen nicht sichtbarer Oberflächen (*hidden surface removal*) gestatten. Der Algorithmus etwa, der in *Doom* hierfür zum Einsatz kommt, wurde 1991 von Chen und Gordon [GC91] entworfen.

Aufgrund der Vielseitigkeit von BSP-Trees finden diese aber auch heute noch in anderen Gebieten Anwendung: So eignen sich BSP-Trees zum Beispiel gut zur Kollisionserkennung. Ein weiteres Anwendungsfeld ist – wie schon von Naylor und Thibault [TN87] beschrieben – das Raytracing. Ein naher Verwandter des BSP-Trees, der Octree – im Gegensatz zu den BSP-Trees sind die Hyperebenen hier fest an den Koordinatenachsen ausgerichtet –, wird in Computerspielen oftmals zum Aussortieren ganzer Objekte, die mitunter aus einer Vielzahl an Polygonen oder Polyedern bestehen können, verwendet (Stichwort *frustum culling*).

5 Literaturverzeichnis

[Cat74] Edwin Earl Catmull. *A subdivision algorithm for computer display of curved surfaces.* PhD thesis, 1974.

[FKN80] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 124–133, New York, NY, USA, 1980. ACM.

[GC91] Dan Gordon and Shuhong Chen. Front-to-back display of bsp trees. *IEEE Comput. Graph. Appl.*, 11(5):79–85, 1991.

[JLO96] A. Janser, W. Luther, and W. Otten. *Computergraphik und Bildverarbeitung.* Vieweg Verlag, 1996.

[Kal82] Y. E. Kalay. Determining the spatial containment of a point in general polyhedra. *Computer Graphics and Image Processing*, 19:303–334, August 1982.

[LTH86] David H. Laidlaw, W. Benjamin Trumbore, and John F. Hughes. Constructive solid geometry for polyhedral objects. *SIGGRAPH Comput. Graph.*, 20(4):161–170, 1986.

[Nay81] Bruce Fountain Naylor. *A priori based techniques for determining visibility priority for 3-d scenes.* PhD thesis, 1981.

[Req80] Aristides G. Requicha. Representations for rigid solids: Theory, methods, and systems. *ACM Comput. Surv.*, 12(4):437–464, 1980.

[RT78] Aristides A.G. Requicha and Robert B. Tilove. Mathematical foundations of constructive solid geometry: General topology of closed regular sets. Technical report, Production Automation Project, University of Rochester, New York, 1978.

[SBGS69] R.A. Schumaker, B. Brand, M. Gilliland, and W. Sharp. Study for applying computer-generated images to visual simulation. Technical report, US Airforce Human Resources Laboratory, 1969.

[Str74] Wolfgang Straßer. *Schnelle Kurven- und Flächendarstellung auf graphischen Sichtgeräten.* PhD thesis, 1974.

[Til84] Robert B. Tilove. A null-object detection algorithm for constructive solid geometry. *Commun. ACM*, 27(7):684–694, 1984.

[TN87] William C. Thibault and Bruce F. Naylor. Set operations on polyhedra using binary space partitioning trees. *SIGGRAPH Comput. Graph.*, 21(4):153–162, 1987.

[TS91] Seth J. Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. *SIGGRAPH Comput. Graph.*, 25(4):61–70, 1991.