Binary Space Partitioning Trees

Grundlagen und Verfahren zur Erzeugung von BSP-Trees

Vortrag im Rahmen des Seminars "Geometrische Modellierung"
(Wintersemester 2008/09)

Timm Linder

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

05. Januar 2009

Inhaltsübersicht

Motivation

- wichtiges Ziel der 3D-Computergrafik: Darstellung komplexer geometrischer Strukturen in Echtzeit
  - z. B. in Simulationen oder Computerspielen
- Suche nach effizienten Datenstrukturen und Zeichenalgorithmen geht schon in die sechziger und siebziger Jahre zurück
- eines der grundlegenden Probleme: Zeichenreihenfolge
  - im Vordergrund befindliche Polygone müssen *nach* den Polygonen im Hintergrund gezeichnet werden
  - Reihenfolge abhängig von Position des Betrachters

Motivation

Maleralgorithmus (I)

Eine erste Idee zur Lösung dieses sog. *Sichtbarkeitsproblems* war der Maleralgorithmus:

Maleralgorithmus (Painter's algorithm)

- Sämtliche Polygone in einer Szene werden in jedem Renderdurchlauf (bzw. nach Veränderung von Objekt- bzw. Betrachterpositionen) der Tiefe nach neu sortiert
- im Hintergrund befindliche Polygone werden unter Umständen mehrfach überzeichnet – selbst wenn sie vollständig verdeckt sind

Motivation

Maleralgorithmus (II)

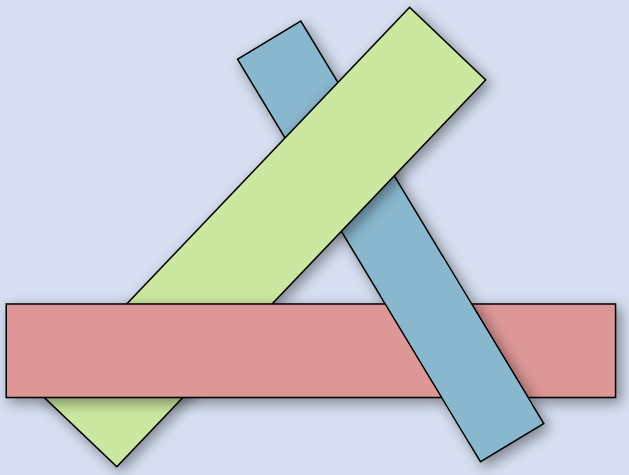
Maleralgorithmus (Painter's algorithm) (Forts.)

- daher auch der Name „Maleralgorithmus“: wie auf einer Ölleinwand werden mehrere Schichten Farbe aufeinander aufgetragen
- einfach umzusetzen, aber durch ständiges Neusortieren und Überzeichnen *(overdraw)* langsam
- weiterer Nachteil: überlappende Polygone bereiten Probleme

Motivation

Maleralgorithmus (III)

Beispiel: Überlappende Polygone



Bei Verwendung von *BSP-Trees* löst sich dieses Problem später von selbst.

Motivation

Z-Buffer-Algorithmus (I)

Ein besserer Ansatz ist der *Z-Buffer-Algorithmus* [JLO96]:

Z-Buffer (Depth-Buffer)-Algorithmus

- bereits 1974 von Catmull [Cat74] und Straßer [Str74] unabhängig voneinander entwickelt
- für jeden bereits gezeichneten Pixel wird eine Tiefeninformation abgespeichert
- beim Rastern von Polygonen wird für jeden Pixel überprüft, ob sich dieser näher am Betrachter befindet als der zuvor an dieser Stelle befindliche Bildpunkt
- Zeichenreihenfolge damit (fast) egal

Motivation

Z-Buffer-Algorithmus (II)

Z-Buffer (Depth-Buffer)-Algorithmus (Forts.)

- auf den heutigen Grafikkarten in Hardware implementiert
  - dadurch sehr schnell
- für damalige Verhältnisse sehr speicheraufwändig (16-Bit-Puffer bei Auflösung 800×600: etwa 1 MB)
- hat sich deshalb erst in den 1990er Jahren durchgesetzt
- nicht Thema dieses Vortrags

Motivation

BSP-Trees (I)

Eine andere Lösung sind die sog. BSP-Trees, mit denen wir uns im Folgenden auseinandersetzen möchten:

BSP-Trees

- Vorgänger der BSP-Trees war eine listen- bzw. graphenbasierten Zuweisung von Prioritäten zu den einzelnen Polygonen (Schumaker et al., 1969 [SBGS69])
- von Fuchs et al. [FKN80] weiterentwickelt und 1980 erstmals unter dem Begriff der *Binary Space Partitioning Trees* erwähnt
- Datenstruktur zur **hierarchischen Unterteilung des Raums** in Regionen

Motivation

BSP-Trees (II)

BSP-Trees

- relativ speicherschonend
- ermöglichen schnelle Bestimmung der Zeichenreihenfolge, müssen (bei unveränderlicher Geometrie) **nur einmal erzeugt werden** und sind unabhängig von Betrachterposition
  - verhindern für sich genommen zunächst keinen Overdraw
  - es gibt aber zusätzliche Algorithmen, die auf BSP-Trees aufbauen und weitere Vorteile bieten:

    z. B. *Potentially Visible Sets (PVS)*, *Portal Rendering* [TS91]

Motivation

BSP-Trees (III)

BSP-Trees

- in den letzten Jahren weit verbreitet
  - z. B. in der Quake 3-Engine oder Half-Life 1 und 2 im Einsatz
- oftmals nicht ausschließlich zur Lösung des Sichtbarkeitsproblems (dafür würde auch der Z-Buffer-Algorithmus genügen), sondern in Kombination mit anderen Algorithmen eingesetzt
  - schnelle **Kollisionserkennung** möglich
  - Verwendung zum **Raytracing**

Motivation

BSP-Trees (IV)

BSP-Trees

- gut geeignet zur Darstellung statischer Welten (BSP-Tree muss nur einmal erzeugt werden)
- verlieren heutzutage wegen des Interesses an „lebendigen“ Welten etwas an Bedeutung
- weisen gewisse Ähnlichkeiten zu **Quadtrees** bzw. **Octrees** auf (diese lassen sich auch durch BSP-Trees repräsentieren)

Begriffserläuterungen (I)

Def.: (Konvexes) Polytop

- allgemeiner Oberbegriff für Polygone beliebiger Dimension
- im $\mathbb{R}^1$: Kante; $\mathbb{R}^2$: Polygon; $\mathbb{R}^3$: Polyeder
- die folgenden Ausführungen über BSP-Trees lassen sich prinzipiell auf beliebig-dimensionale Polytope übertragen

Def.: Hyperebene

- Verallgemeinerung des aus dem $\mathbb{R}^3$ bekannten Begriffs der **Ebene**
- entspricht im $\mathbb{R}^2$ einer Geraden

Begriffserläuterungen (II)

Def.: Boundary Representation (B-Rep)

- auf deutsch: *Flächenbegrenzungsmodell*
- intuitive, übliche Art der Darstellung von geometrischen Modellen (Polytopen)
- hierarchische Unterteilung in niedriger-dimensionale „topologische Primitive“
  - z. B. im $\mathbb{R}^3$: Modell besteht aus aneinander angrenzenden Flächen *(faces)*, Flächen aus Kanten *(edges)*, Kanten aus Eckpunkten *(vertices)* (= **Begrenzungen**)

Gliederung

Grundlagen von BSP-Trees

Was sind BSP-Trees? (I)

Def.: Binary Space Partitioning Tree [TN87]

Ein *Binary Space Partitioning Tree*

- ist ein binärer Baum
  - **innere Knoten** sind *Hyperebenen* im d-dimensionalen Raum (z. B. dem $\mathbb{R}^3$)
  - Hyperebenen unterteilen Raum in zwei Halbräume H^- u. H^+
  - Wurzel entspricht üblicherweise dem ganzen d-Raum *(Anfangsregion)*
  - **Kindknoten** führen zur weiteren Unterteilung übergeordneter Regionen
  - **Blätter** *(„Zellen“)* enthalten keine weiteren Teilungsebenen

Grundlagen von BSP-Trees

Was sind BSP-Trees? (II)

Def.: Binary Space Partitioning Tree (Forts.)

- ermöglicht die Darstellung von beliebigen, linear begrenzten d-dimensionalen Polytopen
- oder auch von Mengen von Polytopen
- ausdrucksstärker als *B-Reps*, weil auch offene (d. h. nicht endlich begrenzte) Körper dargestellt werden können
  - Beispiel ist der triviale BSP-Tree, der nur aus der Wurzel ($\rightarrow$ Zelle) besteht und den ganzen $\mathbb{R}^d$ beinhaltet

Grundlagen von BSP-Trees

Halbräume und Hyperebenen (I)

Hyperebenengleichung in Normalenform:

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

Wenn der Normalenvektor $(a_1, \ldots, a_n)$ auf die Länge 1 normiert ist, gibt b den Abstand der Ebene zum Ursprung an.

Linker (negativer/hinterer) Halbraum von H:

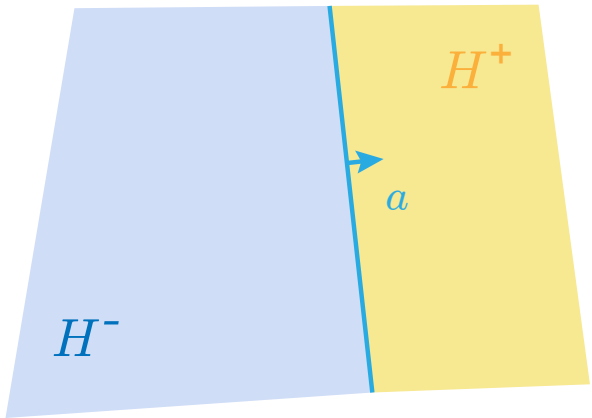
$$H^- = \{(x_1, \ldots, x_d) \mid a_1 x_1 + \ldots + a_d x_d - b < 0\}$$

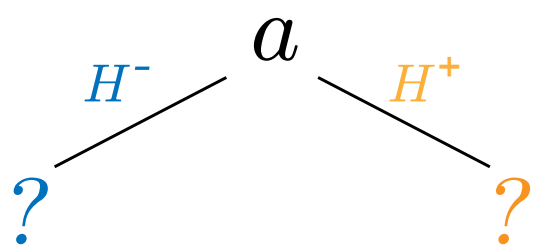
Rechter (positiver/vorderer) Halbraum von H:

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

Grundlagen von BSP-Trees

Halbräume und Hyperebenen (II)





In der Baumstruktur eines BSP-Trees entspricht die zum linken Kindknoten führende Kante dem H^-, die zum rechten Kindknoten führende dem H^+.

H^+ liegt in Richtung der Normalen der Hyperebene.

Grundlagen von BSP-Trees

Regionen

Jedem Knoten v eines BSP-Trees ordnen wir eine Region $R(v)$ zu:

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

wobei $HS(e)$ den zur Kante e gehörigen Halbraum angibt und $E(v)$ den Pfad von der Wurzel zu v darstellt – also eine Menge von Kanten.

Grundlagen von BSP-Trees

Attribute von inneren Knoten

Wir weisen inneren Knoten bestimmte Attribute zu [TN87]:

Attribute von inneren Knoten

- die Teilungsebene H_v
- zwei Mengen $F_{\textit{gleiche Orientierung}}$ und $F_{\textit{umgekehrte Orientierung}}$
  - enthalten die **Begrenzungsflächen** des Polytops, die zum Knoten gehören, d. h. die innerhalb der Teilungsebene liegen
  - Aufteilung in zwei Mengen von Flächen, deren Flächennormalen die *gleiche* bzw. die *umgekehrte* Orientierung wie die Teilungsebene besitzen
  - üblicherweise disjunkt
- ggf. weitere Attribute

Grundlagen von BSP-Trees

Attribute von Blättern (Zellen)

Auch die Blätter (Zellen) des Baums erhalten Attribute:

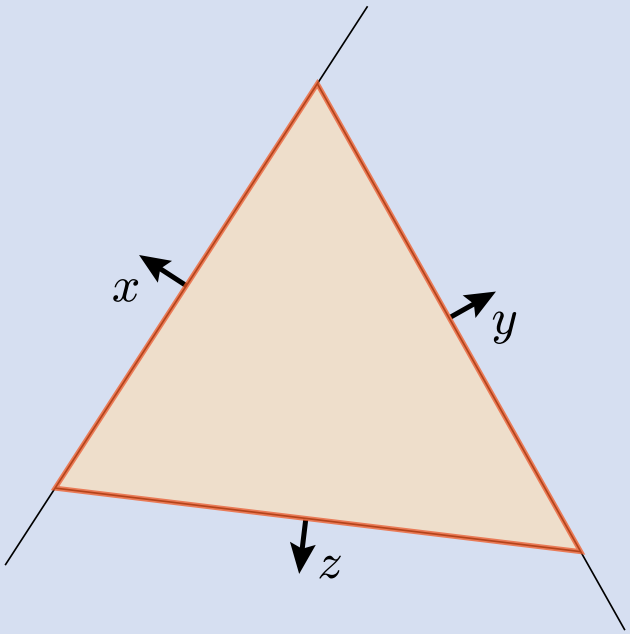
Zellattribute

- Attribut *Mitgliedschaft*
  - gibt an, ob es sich um eine *innere* oder eine *äußere* Zelle handelt (*in* / *out*)
  - ermöglicht eine Unterscheidung zwischen dem Inneren des Polytops (z. B. eines Würfels) und dem Äußeren
- weitere Attribute möglich

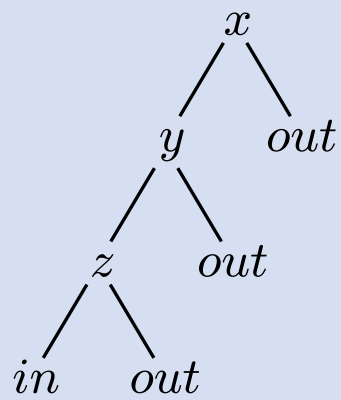
Grundlagen von BSP-Trees

Ein Beispiel

Beispiel für einen BSP-Tree



Geometrisches Objekt (Dreieck)

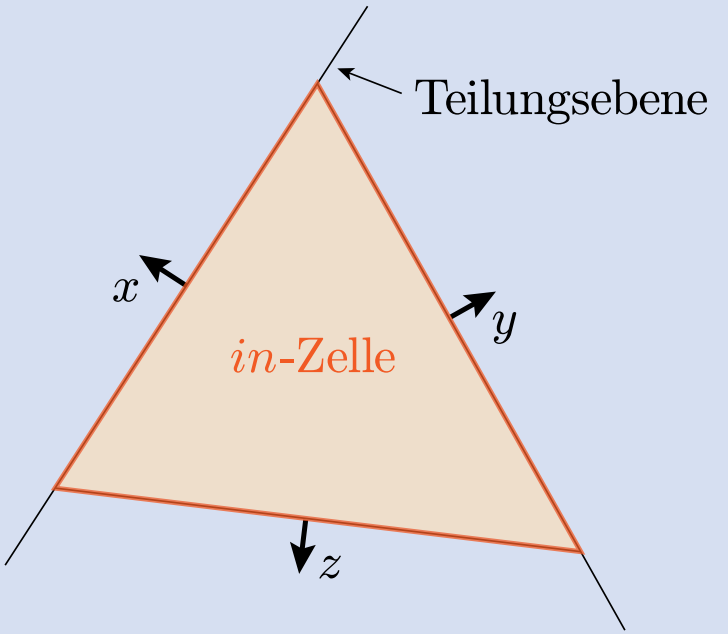


Zugehöriger BSP-Tree

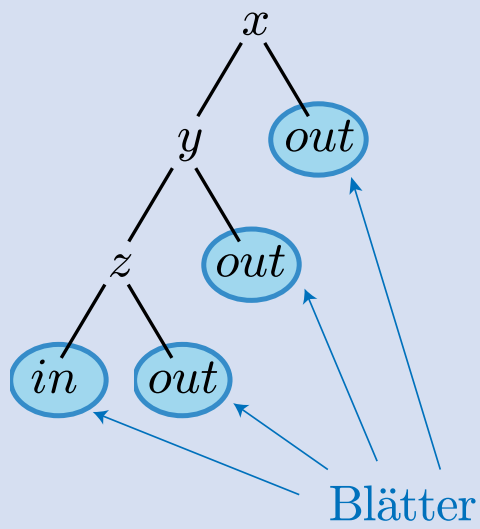
Grundlagen von BSP-Trees

Ein Beispiel

Beispiel für einen BSP-Tree



Geometrisches Objekt (Dreieck)



Zugehöriger BSP-Tree

Grundlagen von BSP-Trees

Punktprobe-Algorithmus (I)

Mithilfe von BSP-Trees kann man auf einfache Weise überprüfen, ob ein Punkt innerhalb des durch den Baum dargestellten Objektes liegt oder nicht.

Zur Klassifizierung eines Punktes dient ein rekursiver Algorithmus.

Grundlagen von BSP-Trees

Punktprobe-Algorithmus (II)

Punktprobe-Algorithmus

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

  Falls v ein Blatt ist
    /* Liefert 'innen' oder 'außen' zurück */
    Gib Mitgliedschaft(v) zurück

  Sonst
    /* Skalarprodukt berechnen;
       b ist Abstand der Hyperebene zum Ursprung
      (falls H in Hessescher Normalform) */
    Berechne d := ⟨ Normalenvektor von H_v, p ⟩ − b
```

Grundlagen von BSP-Trees

Punktprobe-Algorithmus (III)

Punktprobe-Algorithmus (Forts.)

```
Falls d < 0
  Gib Punktprobe(p, v.linkesKind) zurück
Sonst, falls d > 0
  Gib Punktprobe(p, v.rechtesKind) zurück
Sonst
  Setze l := Punktprobe(p, v.linkesKind)
  Setze r := Punktprobe(p, v.rechtesKind)
  Falls l == r
    Gib l zurück
  Sonst
    Gib aufRand zurück
```

Grundlagen von BSP-Trees

Punktprobe-Algorithmus (IV)

- Punktklassifizierung im $\mathbb{R}^3$ mithilfe von **B-Reps** hat Aufwand $O(n)$ bei n Flächen [Kal82]
- eben beschriebener Algorithmus für **BSP-Trees** besitzt im Allgemeinen auch lineare Laufzeit [Nay81]
- für *balancierte* Bäume der Größe $O(n)$ ist jedoch Laufzeit $O(\log n)$ möglich [TN87]
- leider existiert nicht immer ein solcher balancierter Baum.

Gliederung

Erzeugung von BSP-Trees

B-Rep → BSP-Tree (I)

- gegeben: eine Boundary Representation
  - d.h. im Wesentlichen eine Menge F von Flächen, die ein Polytop oder eine Menge (disjunkter) Polytope darstellt
- gesucht: zugehöriger BSP-Tree
- lässt sich gut mit einem rekursiven Algorithmus lösen

Erzeugung von BSP-Trees

B-Rep $\longrightarrow$ BSP-Tree (II)

- grobe Vorgehensweise:
  1. beliebige Fläche wählen und zugehörige Hyperebene erzeugen
  2. die restlichen Flächen anhand dieser Hyperebene unterteilen in drei Mengen F_{links}, F_{rechts}, $F_{in\ Ebene}$
  3. aus F_{links} und F_{rechts} jeweils wieder eine Hyperebene wählen, die beiden Mengen anhand der jeweiligen Hyperebene unterteilen, usw.
  4. Vorgang solange rekursiv durchführen, bis alle Flächen auf einer Seite der entsprechenden Hyperebene liegen ($\rightsquigarrow$ *homogene Region*)

Erzeugung von BSP-Trees

B-Rep → BSP-Tree (III)

- sobald man eine homogene Region erhält, muss man noch bestimmen, ob es sich um eine *innere* oder *äußere* Zelle handelt
  - eine Möglichkeit: Orientierung der Flächen in $F_{in\ Ebene}$ mit Orientierung der Normalen der Hyperebene vergleichen
  - je nachdem, ob es sich um einen linken oder rechten Kindknoten handelt, liegt bei gleicher Orientierung dann eine innere bzw. äußere Zelle vor
  - allgemeineres Verfahren, auch bei Mengenoperationen anwendbar: In/Out-Test von Naylor & Thibault [TN87]

Erzeugung von BSP-Trees

Ein Beispiel

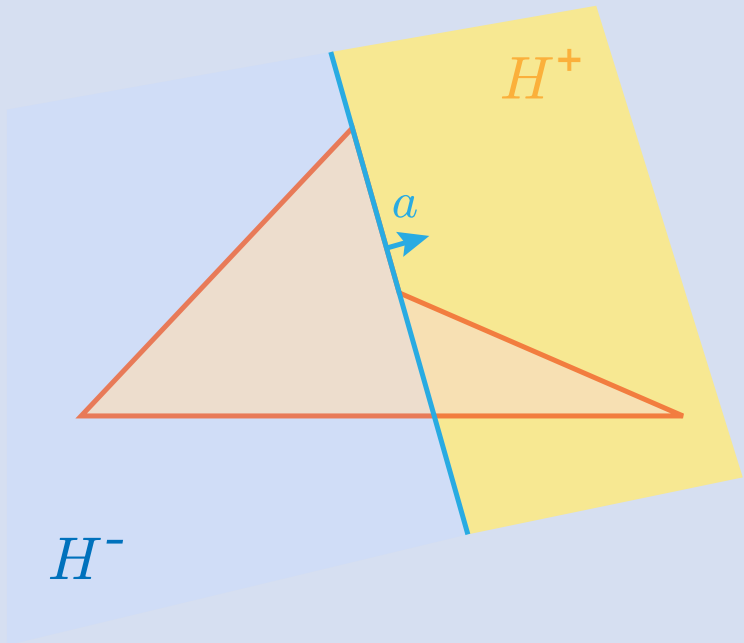
Beispiel: Erzeugung eines BSP-Trees aus einer B-Rep

Geometrisches Objekt (B-Rep)

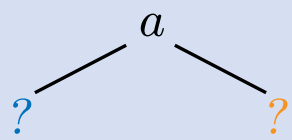
Erzeugung von BSP-Trees

Ein Beispiel

Beispiel: Erzeugung eines BSP-Trees aus einer B-Rep



Partitionierung (Schritt 1)

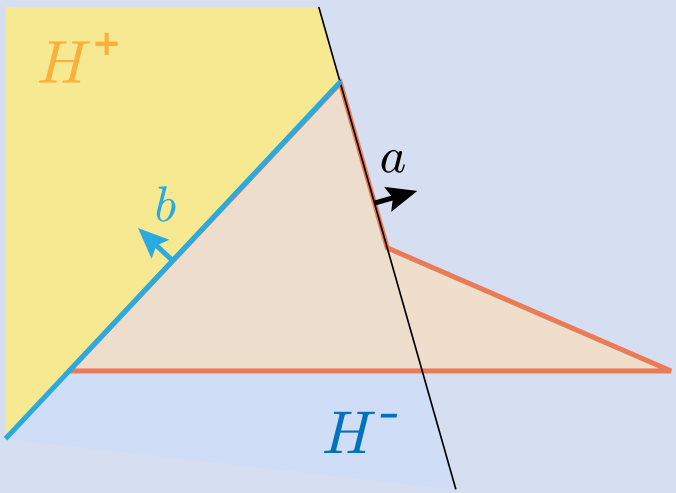


Zugehöriger BSP-Tree

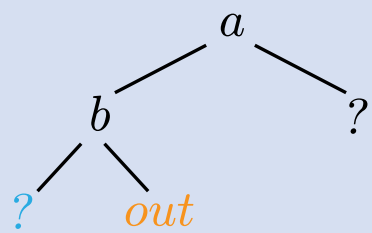
Erzeugung von BSP-Trees

Ein Beispiel

Beispiel: Erzeugung eines BSP-Trees aus einer B-Rep



Partitionierung (Schritt 2)

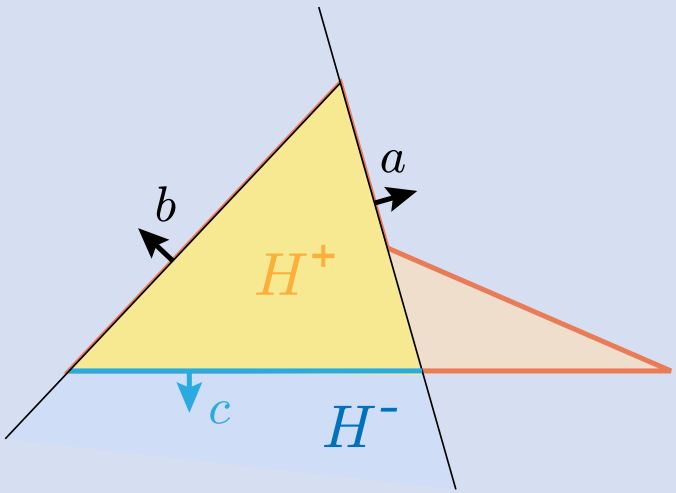


Zugehöriger BSP-Tree

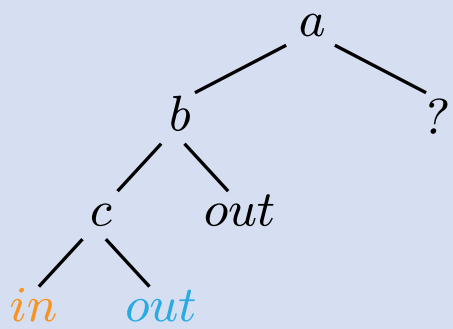
Erzeugung von BSP-Trees

Ein Beispiel

Beispiel: Erzeugung eines BSP-Trees aus einer B-Rep



Partitionierung (Schritt 3)

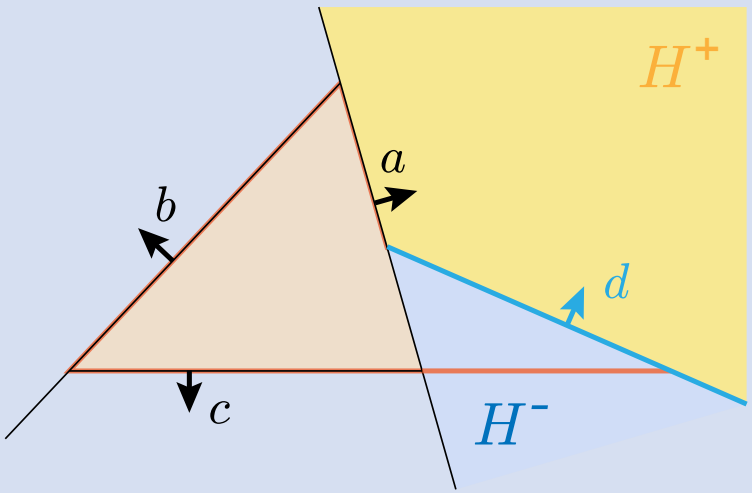


Zugehöriger BSP-Tree

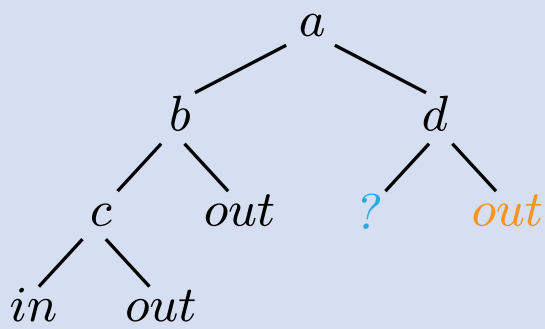
Erzeugung von BSP-Trees

Ein Beispiel

Beispiel: Erzeugung eines BSP-Trees aus einer B-Rep



Partitionierung (Schritt 4)

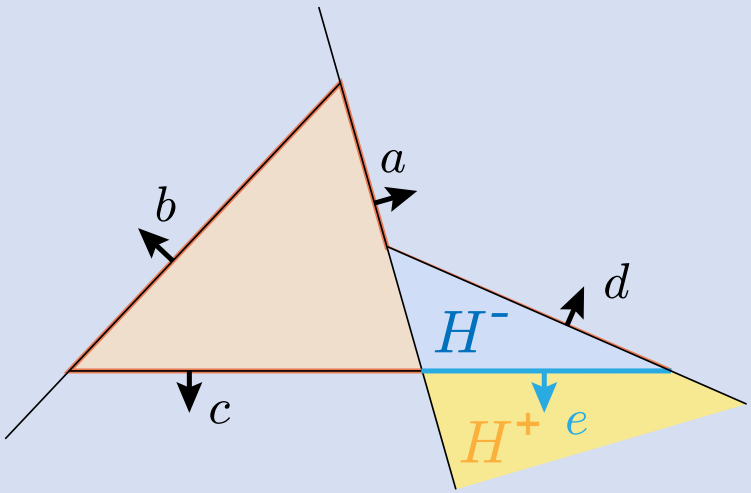


Zugehöriger BSP-Tree

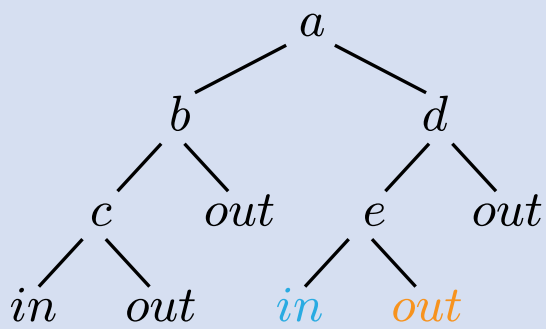
Erzeugung von BSP-Trees

Ein Beispiel

Beispiel: Erzeugung eines BSP-Trees aus einer B-Rep



Partitionierung (Schritt 5)

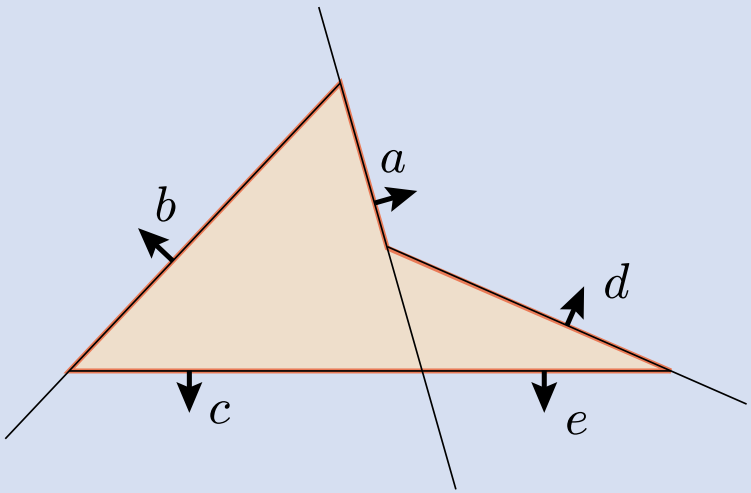


Zugehöriger BSP-Tree

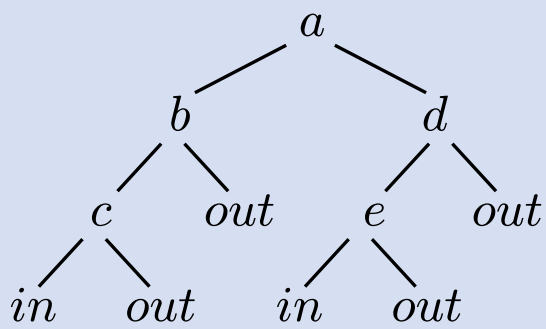
Erzeugung von BSP-Trees

Ein Beispiel

Beispiel: Erzeugung eines BSP-Trees aus einer B-Rep



Partitionierung (Schritt 6)



Endgültiger BSP-Tree

Erzeugung von BSP-Trees

Bemerkungen

- Struktur und Größe des erzeugten BSP-Trees ist abhängig von Wahl der Hyperebenen
- man muss abwägen zwischen Größe des Baumes und Anzahl an Flächen, die zerteilt werden
- im vorigen Beispiel gäbe es eine „bessere“ Zerlegung
- ein BSP-Tree teilt die darzustellenden Körper immer in **konvexe** Regionen auf
- Aufwand zur Bestimmung einer *optimalen* Zerlegung ist NP-hart [Cha84]
- Einsatz bestimmter Heuristiken sinnvoll [TN87]

Gliederung

Mengenoperationen auf BSP-Trees

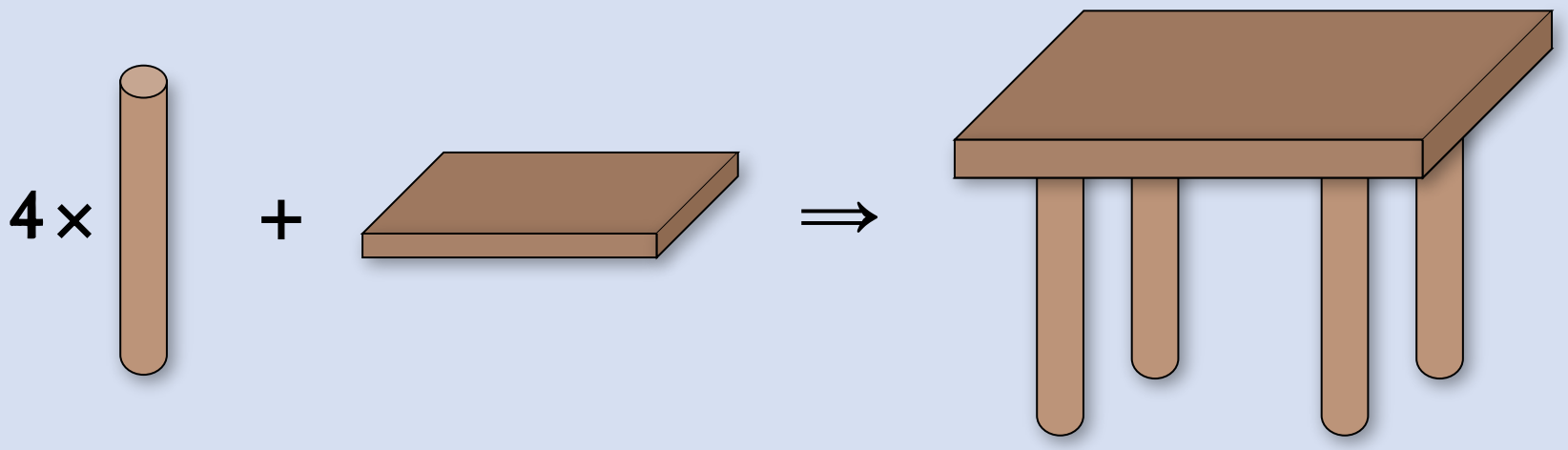
Idee / Motivation

- Ziel: geometrische Objekte durch Mengenoperationen miteinander kombinieren
  - Vereinigung: $A \cup B$
  - Durchschnitt: $A \cap B$
  - Differenz: $A \setminus B$
  - Komplement: C A
- im Folgenden: erster Operand ist **BSP-Tree**, zweiter eine **Boundary Representation**
  - z. B. zur Manipulation eines Levels von einem Computerspiel (gespeichert als BSP-Tree) mithilfe eines Formwerkzeuges (B-Rep)

Mengenoperationen auf BSP-Trees

Idee / Motivation

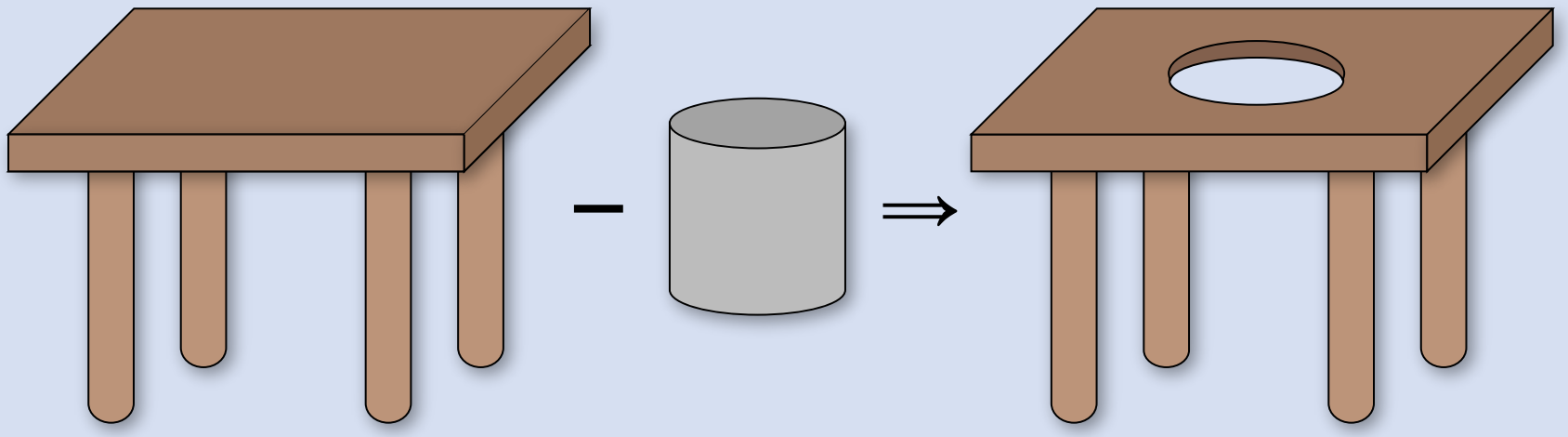
Beispiel: Mengentheoretische Vereinigung



Mengenoperationen auf BSP-Trees

Idee / Motivation

Beispiel: Mengentheoretische Differenz



Mengenoperationen auf BSP-Trees

Regularisierte Mengenoperationen (I)

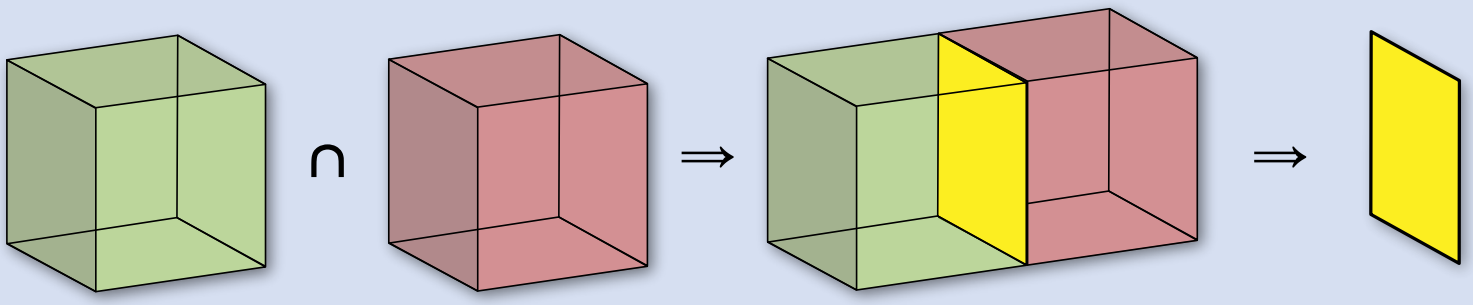
Problem:

Bei Bildung des „normalen“ (nicht-regularisierten) **Durchschnitts**, **Komplements**, der **Differenz** oder der **Vereinigung** kann es dazu kommen, dass voluminöse Modelle zu Flächen entarten.

Mengenoperationen auf BSP-Trees

Regularisierte Mengenoperationen (II)

Beispiel: Schnitt von zwei adjazenten Würfeln



Der Schnitt zweier aneinander angrenzender Würfel liefert eine Fläche (statt eines dreidimensionalen Festkörpers). Das ist unerwünscht.

Mengenoperationen auf BSP-Trees

Regularisierte Mengenoperationen (III)

Lösung: Einführung **regularisierter** Mengenoperationen

Regularisierte Mengenoperationen [RT78]

- **Komplement:** $\complement^\star A = (cl(int(\complement\, A))$
- **Vereinigung:** $A \cup^\star B = cl(int(A \cup B))$
- **Durchschnitt:** $A \cap^\star B = cl(int(A \cap B))$
- **Differenz:** $A \setminus^\star B = cl(int(A \setminus B))$

Dabei steht *cl* für die *abgeschlossene Hülle (closure)* und *int* für das *Innere* einer Menge. Die abgeschlossene Hülle besteht aus dem Inneren und den Randpunkten der Menge.

Mengenoperationen auf BSP-Trees

Regularisierte Mengenoperationen (IV)

Eine **reguläre Menge** besteht somit aus der abgeschlossenen Hülle ihres Inneren, d. h.

$$M \text{ regulär } \Rightarrow S = cl(int\ S).$$

Man entfernt also durch Bildung des Inneren zunächst alle Randpunkte der Menge (daraus ergibt sich im vorigen Beispiel die leere Menge), und fügt der Menge durch Bildung des Abschlusses dann wieder einen Rand hinzu (es gilt aber $cl(\emptyset) = \emptyset$).

Mengenoperationen auf BSP-Trees

BSP-Tree $\langle op \rangle$ B-Rep $\rightarrow$ BSP-Tree (I)

Bevor wir das allgemeine Vorgehen grob beschreiben, zunächst zwei Grundregeln:

- **Komplementbildung**
  - bei BSP-Trees: innere Zellen in äußere umwandeln, und umgekehrt; außerdem Normalen aller Begrenzungsflächen invertieren, die in den inneren Knoten gespeichert sind
  - bei B-Reps: alle Flächennormalen invertieren
- Bildung der mengentheoretischen **Differenz**:
  $M_1 \setminus^* M_2 \Leftrightarrow M_1 \cap^* C^* M_2$

Mengenoperationen auf BSP-Trees

BSP-Tree $\langle op \rangle$ B-Rep $\rightarrow$ BSP-Tree (II)

Sei nun ein Polytop T durch einen BSP-Tree $\widehat{T}$ gegeben, sowie ein weiteres Polytop R durch eine B-Rep $\widehat{R}$. Wir berechnen $T\ op\ R$ wie folgt:

- schrittweises Einfügen aller Flächen von R in den BSP-Tree $\widehat{T}$
  - jede Fläche im Baum „einsinken“ lassen und an den Hyperebenen der Knoten zerteilen
  - sobald an einem Knoten alle „ankommenden“ Teilflächen auf der gleichen Seite der Teilungsebene landen, ist die andere Seite (also der Halbraum) *homogen bzgl. R*
  - in R befinden sich in der entsprechenden Region dann also keine weiteren Begrenzungsflächen mehr

Mengenoperationen auf BSP-Trees

BSP-Tree $\langle op \rangle$ B-Rep $\rightarrow$ BSP-Tree (III)

- bis hierhin ist der Algorithmus unabhängig von der jeweiligen Mengenoperation
- sobald eine Fläche ein Blatt (Zelle) erreicht, können wir die eigentliche **Mengenoperation** anwenden
- dabei können bestimmte *Vereinfachungsregeln* angewendet werden
- vorher muss man ggf. noch bestimmen, ob die durch die Flächen repräsentierten Regionen aus dem Inneren oder Äußeren der B-Rep stammen
  - In/Out-Test von Naylor & Thibault [TN87]

Mengenoperationen auf BSP-Trees

BSP-Tree $\langle op \rangle$ B-Rep $\rightarrow$ BSP-Tree (IV)

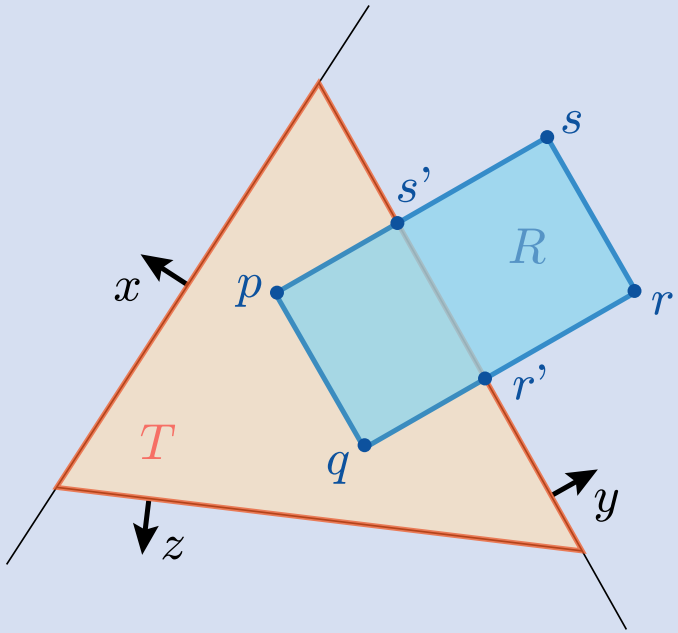
Vereinfachungsregeln für den Durchschnitt

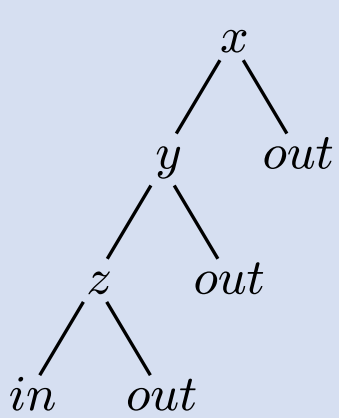
op	Operand 1	Operand 2	Ergebnis
$\cap^\star$	*M*	*innen*	*M*
	M	*außen*	*außen*
	innen	*M*	*M*
	außen	*M*	*außen*

Mengenoperationen auf BSP-Trees

BSP-Tree $\langle op \rangle$ B-Rep $\rightarrow$ BSP-Tree

Beispiel: Bildung der mengentheoretischen Differenz





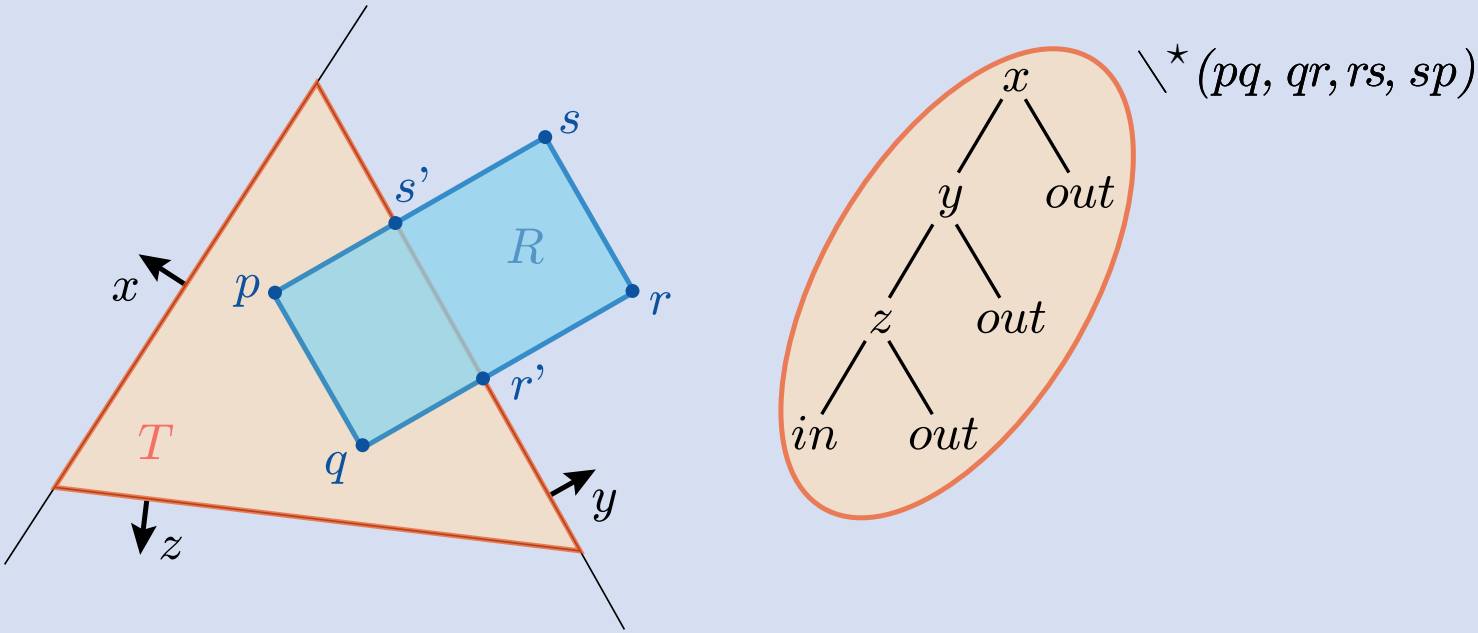
$\setminus^{\star}(pq, qr, rs, sp)$

Ausgangssituation. BSP-Tree in orange, B-Rep in blau.

Mengenoperationen auf BSP-Trees

BSP-Tree $\langle op \rangle$ B-Rep $\rightarrow$ BSP-Tree

Beispiel: Bildung der mengentheoretischen Differenz

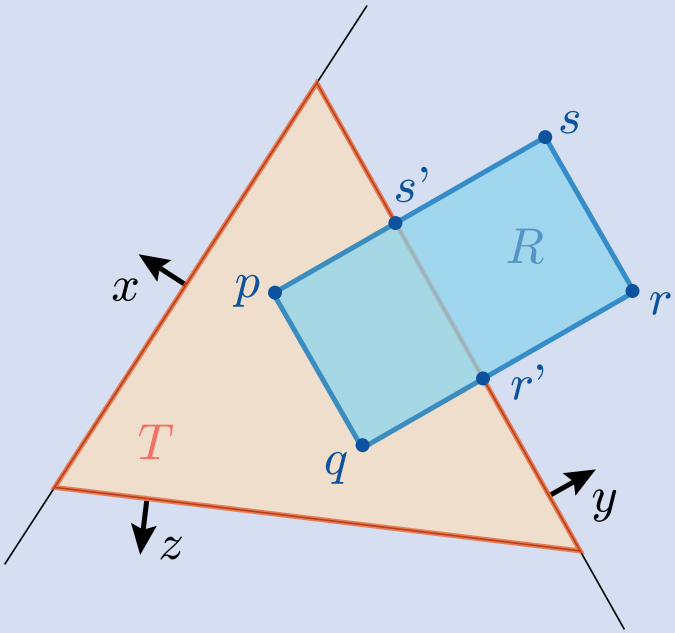


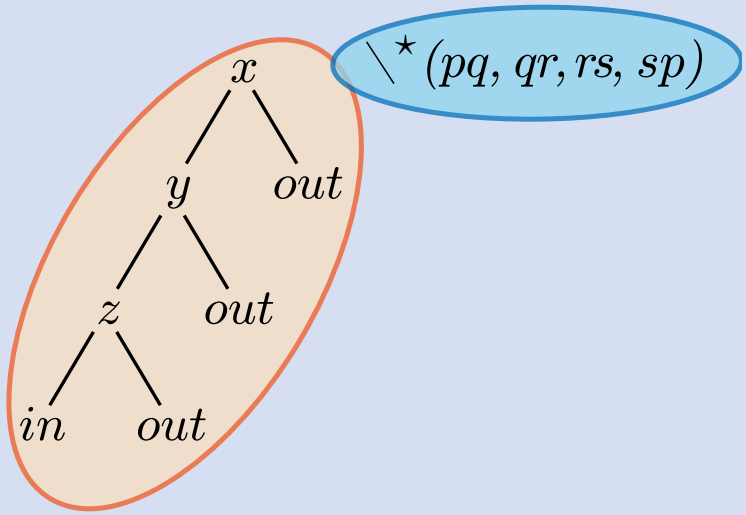
Ausgangssituation. BSP-Tree in orange, B-Rep in blau.

Mengenoperationen auf BSP-Trees

BSP-Tree $\langle op \rangle$ B-Rep $\longrightarrow$ BSP-Tree

Beispiel: Bildung der mengentheoretischen Differenz



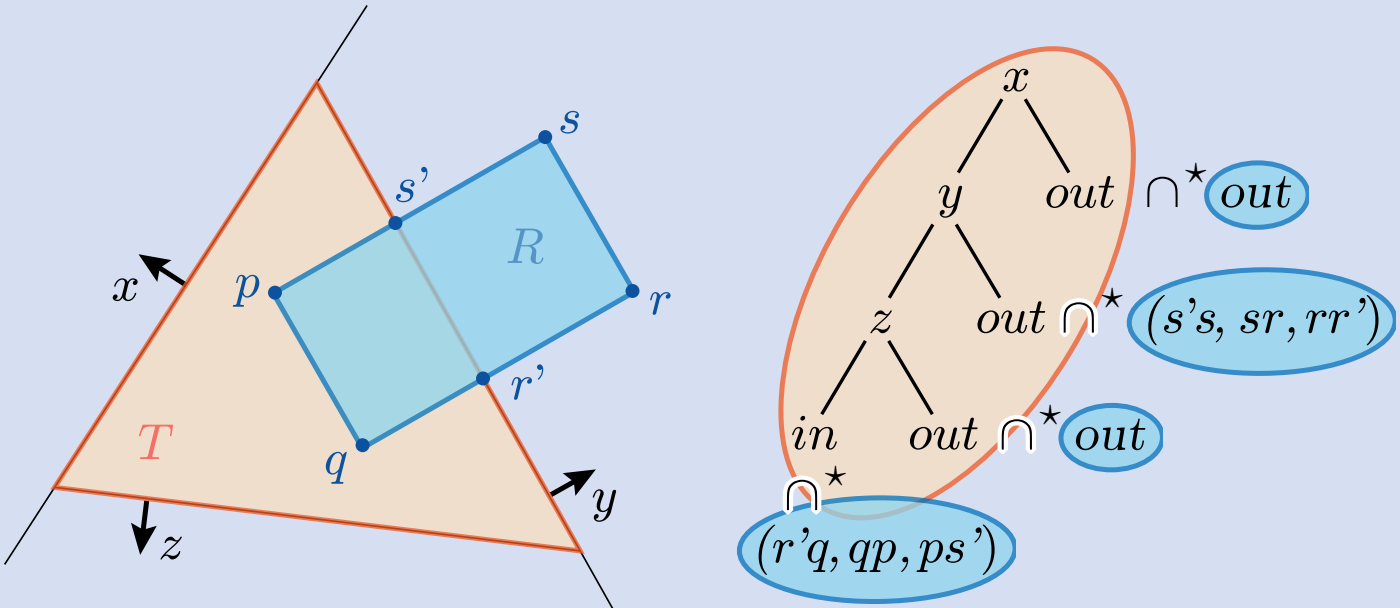


Ausgangssituation. BSP-Tree in orange, B-Rep in blau.

Mengenoperationen auf BSP-Trees

BSP-Tree $\langle op \rangle$ B-Rep $\longrightarrow$ BSP-Tree

Beispiel: Bildung der mengentheoretischen Differenz

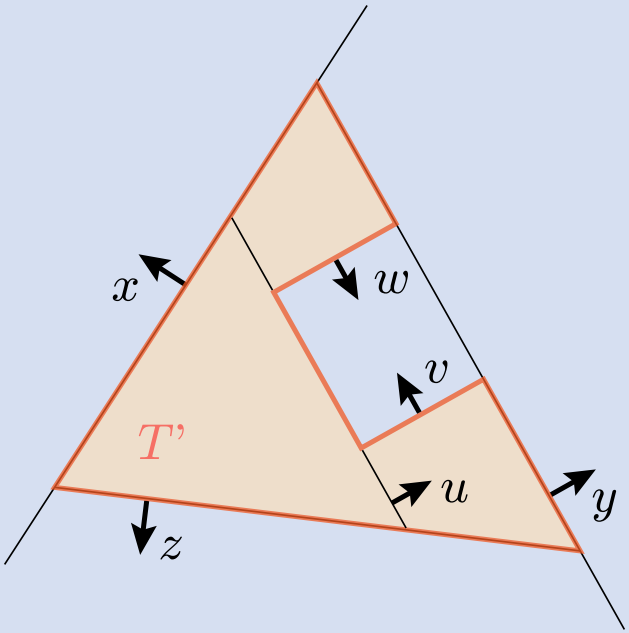


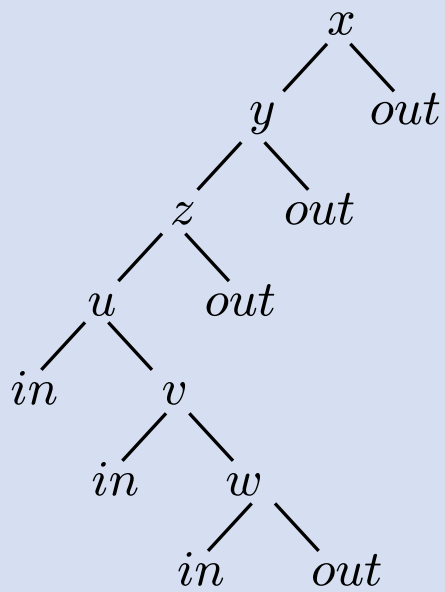
Nach Klassifizierung der B-Rep (blau)

Mengenoperationen auf BSP-Trees

BSP-Tree $\langle op \rangle$ B-Rep $\rightarrow$ BSP-Tree

Beispiel: Bildung der mengentheoretischen Differenz





Resultat (nach Anwendung von Vereinfachungsregeln)

Mengenoperationen auf BSP-Trees

Weitere Möglichkeiten

Es sind noch andere Arten von Mengenoperationen denkbar, die auf BSP-Trees aufbauen:

- Umwandlung von CSG-Bäumen *(Constructive Solid Geometry)* in BSP-Trees
  - CSG-Baum: hierarchische Verknüpfung von B-Reps mithilfe boolescher Ausdrücke
  - vgl. Naylor & Thibault (1987) [TN87]
- direkte Verknüpfung zweier BSP-Trees

Gliederung

Schlusswort

Mangels Zeit wollen wir hier keine großen Worte mehr verlieren, sondern...

Fragen/Diskussion

Fragen!?

Vielen Dank für die Aufmerksamkeit!

Timm Linder
timm.linder*(at)*stud.uni-due.de

Bibliographie (1)


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


Bernard Chazelle.
Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm.
SIAM Journal on Computing, 1984.

Bibliographie (2)


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.


A. Janser, W. Luther, and W. Otten.
Computergraphik und Bildverarbeitung.
Vieweg Verlag, 1996.

Bibliographie (3)


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


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

Bibliographie (4)


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.


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.

Bibliographie (5)

Wolfgang Straßer.
Schnelle Kurven- und Flächendarstellung auf graphischen Sichtgeräten.
PhD thesis, 1974.

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

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