1 3D-Datenstrukturen

1.1 Anforderungen an die interne Darstellung

Die geometrischen Daten der Objekte, die modelliert werden, müssen in einer geeigneten Datenstruktur abgelegt werden.

Jetzt schon muß die Entscheidung getroffen werden, wie "realistisch" das erzeugte Bild werden soll. Verliert man hier schon (geometrische) Information, so verliert man auch an Realismus im erzeugten Bild.

Folgende Anforderungen werden vom Modeller gestellt:

- Lineare Transformationen
- Kombinationsoperatoren
- Möglichst allgemeine Objekte repräsentierbar
- Geringer Speicherbedarf.
- Möglichkeit der Interaktion
- Automatische Generierung der Datenstruktur anhand von digitalisierten realen Objekten, z.B. mittels 3D-Scanner oder CT

Außerdem verlangt das Renderingsystem zusätzlich:

- Schnelle Darstellbarkeit
- Objekte sollen möglichst exakt repräsentiert sein

1.2 Möglichkeiten der internen Darstellung

Im folgenden Kapitel werden verschiedene Darstellungsformen vorgestellt und analysiert. Bei jeder Repräsentationsform wird darauf geachtet, inwieweit sie die oben angeführten Anforderungen erfüllt.

1.2.1 Punktwolke

Ein Objekt wird durch eine Menge von Punkten repräsentiert. Das ist eine sehr einfache Repräsentation. Man hat dabei keine direkte Information über die Oberfläche zwischen den Punkten oder das Volumen des Objekts. Eine Punktwolke wird z.B. von Eingabegeräten wie einem Digitalisier-Stift geliefert. Aus solch einer Punktwolke kann dann heuristisch eine Objektoberfläche ermittelt werden, indem z.B. Polygone zwischen den jeweils am nähesten benachbarten Punkten gespannt werden.

Datenstruktur:

Objekt = Liste von Punkten
Punkt = 3 Koordinaten (x,y,z) im Raum

Elementarobjekte:

Man benötigt eine große Anzahl von Punkten um gekrümmte Oberflächen ausreichend exakt repräsentieren zu können.

Transformationen:

Man muß nur die Punkte transformieren um das Objekt zu transformieren.

Kombinationen:

Nicht möglich ohne zusätzliche Annahmen, da man keine Information über Innen und Außen der Objekte besitzt. Man kann lediglich die Punkte mehrerer Objekte zu einem einzelnen Objekt zusammenfassen, was jedoch nur teilweise der Vereinigungsoperation entspricht.

Speicherplatz:

Um gekrümte Oberflächen zu repräsentieren benötigt man viele Punkte, was entsprechend viel Speicherplatz verbraucht.

Rendering:

Zur einfachen Darstellung muß man nur die Punkte ins Bild projizieren. Einige Grafikprogramme verwenden Punktwolken um Objekte in einer schnellen vereinfachten Form darzustellen. Objekte lassen sich sogar exakt direkt als Punktwolke darstellen wenn man genug Punkte hat um damit geschlossene Oberflächen im Bild zu erzielen (≥ 1 Punkt pro Pixel). Die Sichtbarkeitsermittlung erfolgt dabei mittels z-Buffer. In der Praxis werden heutzutage zwar kaum Punktwolken anstelle von Polygonen zur exakten Darstellung von Oberflächen eingesetzt, doch vor allem wegen der immer leistungsfähiger werdenden Grafik-HW kommt man teilweise schon in Bereiche wo man pro Bild mehr Primitive (Polygone) als Pixel verwendet. In solch einer Situation ist die Verwendung von Punkten effizienter als die Verwendung von Polygonen die in der Größenordnung eines Pixels sind.

Vorteile:

+ Schnelle einfache Darstellbarkeit
+ exakte Darstellung mit korrekter Sichtbarkeit möglich
+ schnelle einfache Transformationen
+ Modellgenerierung durch Digitalisierung realer Objekte möglich

Nachteile:

- Viele Punkte für krumme Oberflächen nötig
- Sehr viele Punkte für exaktes Rendering nötig
- Hoher Speicherbedarf z.B. bei krummen Oberflächen
- Kombinationsoperationen nur beschränkt möglich

1.2.2 Wire-Frame-Modell

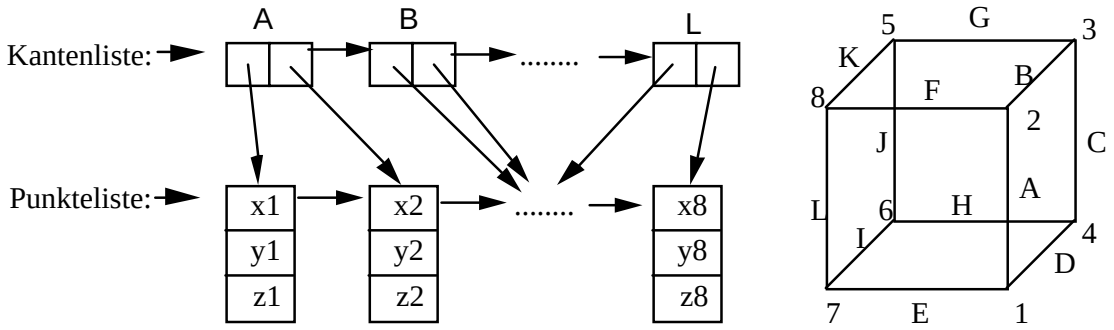
Im Wire-Frame-Modell (Drahtmodell) wird jeder Körper durch einige gerade Kanten repräsentiert. Ein Würfel wird daher nur durch seine zwölf Kanten dargestellt, man hat keine

geometrische Information über die Bereiche zwischen den Kanten oder zwischen den Flächen des Würfels.

Datenstruktur:

Eine übliche hierarchische Organisation der Daten ist folgende:

Objekt = Liste von Verweisen auf Kanten.
Kantenliste = Liste von allen Kanten im Modell.
Kante = Zwei Verweise auf Punkte.
Punkteliste = Liste aller Punkte im Modell.
Punkt = Drei Koordinaten (x,y,z) im 3D-Raum.



Elementarobjekte:

Man braucht eine große Anzahl von Primitiven, da man durch Kombination kaum neue Objekte erzeugen kann. Dazu eignen sich alle Primitive, die gerade Kanten haben, sie lassen sich sehr gut darstellen. Eine Kugel, die eine gekrümmte Oberfläche besitzt, kann nur durch ein spinnwebenartiges Liniengerüst approximiert werden. Um keine Ecken in solch einer gekrümmten Oberfläche zu sehen benötigt man eine große Anzahl an Kanten.

Transformationen:

Lassen sich sehr leicht durchführen. Man braucht nur alle Punkte in der Punkteliste mit linearen Transformationsmatrizen zu transformieren und hat damit das gesamte Objekt transformiert.

Kombinationen:

Lassen sich nicht machen, da man keine Information über Innen oder Außen der Objekte besitzt. Man kann höchstens zwei Objekte zu einem neuen vereinigen, indem man sowohl die Punkte, als auch die Kantenliste vereinigt, was jedoch nur teilweise dem Vereinigungsoperator entspricht.

Speicherplatz:

Diese Datenstruktur hat den Vorteil, daß Punkte von denen mehrere Kanten ausgehen nicht doppelt aufgehoben werden. Will man aber z.B. eine Kugel darstellen, so braucht man, um sie gut zu approximieren, sehr viele Kanten und Punkte, was Speicherplatz kostet.

Rendering:

Dieses Modell wurde vor allem für Ausgabegeräte entwickelt, die nur Linien zeichnen können, also vor allem für Vektorbildschirme oder Plotter. Auf solchen Geräten kann man aber auch keine sehr realistischen Bilder erzeugen, deswegen genügt diese interne Darstellung.

Man braucht nur alle Punkte aus der Punkteliste auf die Bildebene zu projizieren, und die dazwischenliegenden Kanten zu zeichnen und hat damit das gesamte Objekt dargestellt. Bei der entsprechenden Hardware ist das ein sehr schnelles Verfahren und kann dazu genützt werden, um einen ersten, schnellen Eindruck von dem Objekt zu bekommen.

Zeichnet man aber alle Linien aus der internen Darstellung, also auch solche, die eigentlich von davor liegenden Flächen verdeckt werden, so entsteht sehr oft ein Liniengewirr, aus dem man das eigentliche Aussehen des Objekts nicht mehr erkennen kann. Ein Hidden-Line Verfahren kann im Drahtmodell nicht angewendet werden, da zu wenig geometrische Information vorhanden ist.

Vorteile:

+ Schnelle Darstellbarkeit.
+ Transformationen leicht zu implementieren und sehr schnell auszuführen.

Nachteile:

- Hoher Speicherbedarf besonders bei krummen Objekten (Punkteliste, Kantenliste).
- Hoher Informationsverlust (sogar verdeckte Kanten können nicht mehr eliminiert werden).
- Kombinationsoperatoren nur beschränkt möglich.
- Alle runden Objekte müssen durch gerade Kanten approximiert werden.

1.2.3 Boundary Representation (B-Rep)

Alle Objekte werden durch ihre Oberflächen (Boundaries) dargestellt.

Im Flächenmodell wird mehr Information über das Objekt abgelegt als im Kantenmodell. Es werden auch noch die Seitenflächen der Objekte aufgehoben.

Zusätzlich kann man auch noch zu jeder Seitenfläche geometrische Information ablegen die später ein leichteres Rechnen möglich macht, wie z.B. die zugehörige Ebenengleichung oder den Normalvektor, wodurch man idirekt Information über den Raum zwischen den Seitenflächen erhält.

Auch bei der B-Rep entsteht das Problem der gekrümmten Oberflächen, die durch ebene Polygonflächen approximiert werden müssen.

Datenstuktur:

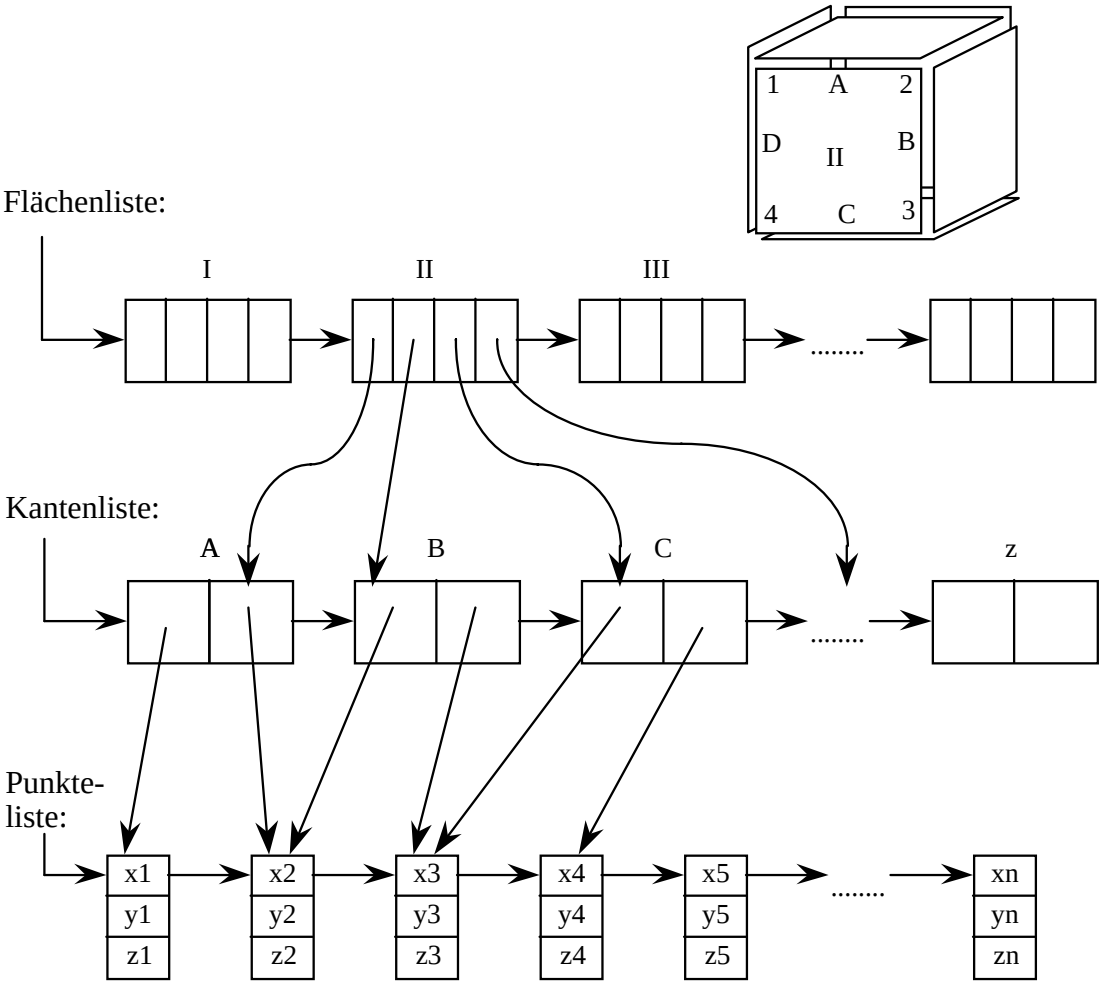
In den meisten Fällen wird man als Begrenzungsflächen nur ebene Polygone zulassen und dafür folgende hierarchische Datenstruktur verwenden:

Objekt = Liste von Verweisen auf Polygone.

Polygonliste = Liste aller Polygone im Modell.

Polygon = Liste von Verweisen auf Kanten.

Kanten- und *Punkteliste* wie beim Wire-Frame-Modell



Elementarobjekte:

Wie im Kantenmodell.

Transformationen:

Bei linearen Transformationen werden die eventuell vorhandenen Ebenengleichungen und Normalvektoren (in der Flächenliste) mittransformiert oder nach der Punkttransformation neu berechnet.

Kombinationen:

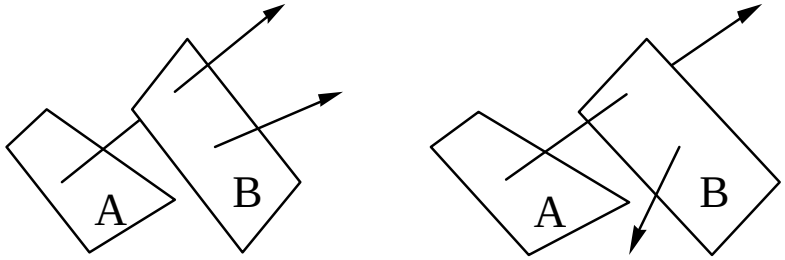
Einfache Kombinationen, wie Vereinigung von zwei Objekten oder Schnitt mit einer Ebene lassen sich durchführen. Man definiert aber oft zusätzlich zu der internen Darstellung eine Semantik, die nur geschlossene Objekte (verboten sind einzelne Polygone oder Objekte mit fehlenden Seitenwänden) zuläßt. Dabei werden der Einfachheit halber die Ebenen in der Flächenlste so definiert, daß jeder Normalvektor eines Polygons nach außen zeigt, also vom Objekt weg. Mit dieser Semantik ist es möglich auch Kombinationen wie Schnitt, Differenz oder Vereinigung zu berechnen:

Kombinationen in der Boundary Representation erreicht man mit folgenden Algorithmus:

1. Zerschneide alle Polygone von Objekt A mit allen Polygonen von Objekt B, wenn notwendig.
2. Zerschneide alle Polygone von Objekt B mit allen Polygonen von Objekt A entsprechend.
3. Klassifiziere die Polygone von Objekt A als "im Objekt B", "außerhalb von Objekt B" oder als "an der Oberfläche von Objekt B".
4. Klassifiziere die Polygone von Objekt B in Bezug auf Objekt A entsprechend.
5. Entferne, je nach Operator, die überflüssigen Polygone aus A und B und füge die beiden Darstellungen zu einer zusammen.

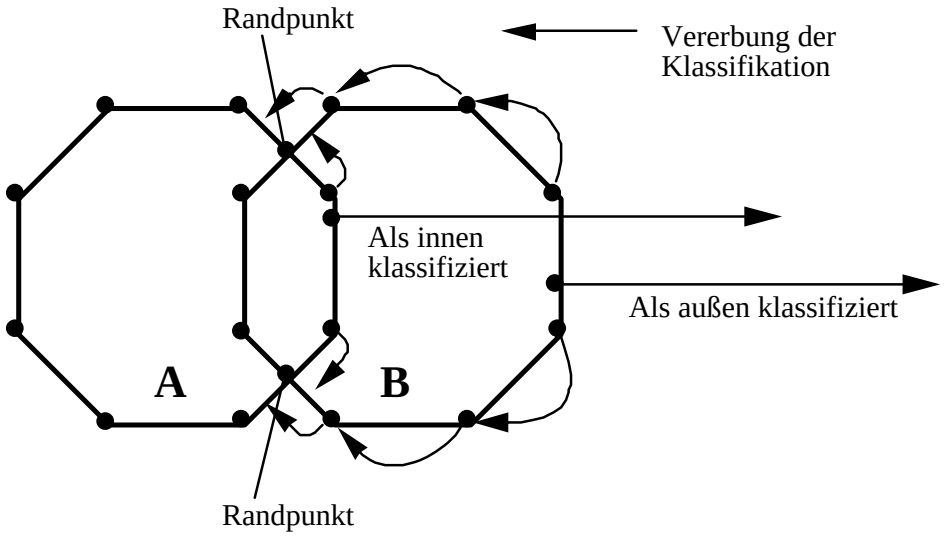
ad 1+2: Jedes Polygon erhält eine Quaderumgebung. Damit ist der Test, ob sich Polygone überhaupt beeinflussen leicht durchzuführen. Nimmt man nur konvexe Polygone als Ausgangssituation an und versucht man beim Zerschneiden wieder nur konvexe Polygone zu erhalten, werden die Schnittalgorithmen einfacher und schneller.

ad 3+4: Es wird ein Strahl vom zu testenden Polygon A in Richtung seines Normalvektors gelegt und mit allen Polygonen von B geschnitten. Das vorderste Polygon von B, das getroffen wird, entscheidet die Klassifikation von A:



A innerhalb von B A außerhalb von B

Verbesserung: Punkte von A, die auf der Oberfläche von B liegen (und umgekehrt), werden (z.B. während des Zerschneidens) als Randpunkte markiert. Zwei benachbarte Polygone haben immer die gleiche Klassifikation, höchstens sie werden durch einen Randpunkt getrennt. Da kann es sein, daß sich die Klassifikation ändert. Über die Punkte kann die Klassifikation also weitergegeben werden ohne sie neu berechen zu müssen (also einen neuen Strahl zu schießen). Nur nach Randpunkten muß neu klassifiziert werden. Dadurch müssen nur noch wenige Polygone auf die komplizierte (ursprüngliche) Weise berechnet werden, sondern die Klassifikation kann von einem Punkt an seine Nachbarpunkte (bis zu einem Randpunkt) weitervererbt werden.



ad 5: Polygone werden nach folgender Entscheidungstabelle **entfernt**:

Für Polygone von A				
	in B	außerhalb B	auf B (coplanar) NV gleich	NV verschieden
A or B	ja	nein	nein	ja
A and B	nein	ja	nein	ja
A minus B	ja	nein	ja	nein

Für Polygone von B				
	in A	außerhalb A	auf A (coplanar) NV gleich	NV verschieden
A or B	ja	nein	ja	ja
A and B	nein	ja	ja	ja
A minus B	nein	ja	ja	ja

Die vorderen beiden Spalten der Tabelle sind intuitiv sofort klar, bei den beiden anderen Spalten muß anhand der Normalvektoren der Polygone entschieden werden, welches in das Ergebnisobjekt aufgenommen wird und welches nicht.

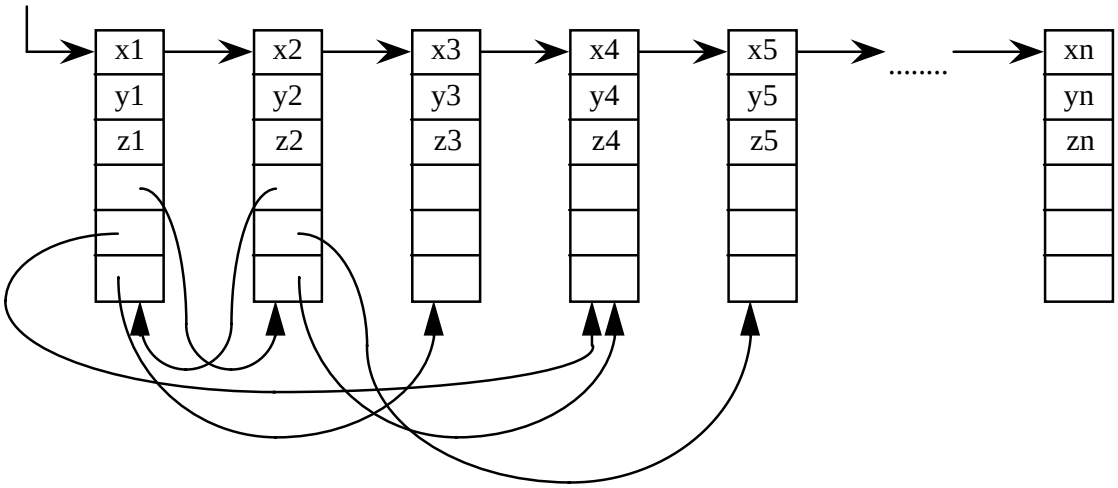
z.B. 1.Zeile: Bei der Vereinigung von 2 Objekten werden all jene Polygone von A eliminiert, die innerhalb des Objekts B liegen. Polygone von A, die auf der Oberfläche von B liegen, werden nur dann eliminiert, wenn die Normalvektoren der Oberflächen von B und A in verschiedene Richtungen zeigen, also die Oberflächen in entgegengesetzte Richtungen schauen.

Aus diesem Algorithmus ergeben sich weitere Anforderungen für die Boundary Representation:

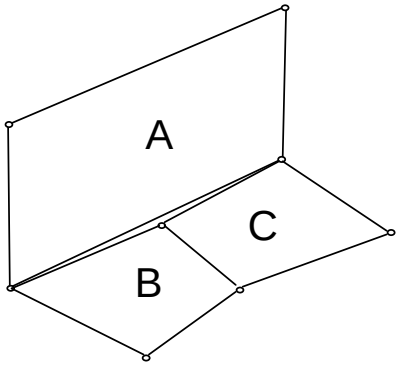
- *nur konvexe Polygone*: erleichtern die Schnittberechnung und die IN-OUT-Tests.
- *keine Doppelpunkte*: sonst kann das weitergeben der Klassifikation über die Punkte falsche Ergebnisse liefern.
- *keine offenen Objekte*: sonst ist eine IN-OUT-Klassifikation überhaupt nicht möglich.
- *eine weitere Verkettung* in der Datenstruktur:

In der Punkteliste, die jetzt alle Polygonpunkte der zerschnittenen Polygone und auch alle Randpunkte enthält, wird über Zeiger vermerkt, welche Punkte von einem bestimmten Punkt die Klassifikation (innen oder außen) erben. Wenn alle Punkte klassifiziert sind, kann diese Klassifikation auch auf die Polygone übertragen und damit entschieden werden, welche Polygone zum neuen Objekt gehören, und welche wegzulassen sind.

Punkteliste:



Bei einer normalen B-Rep kann es vorkommen, daß zwei aneinandergrenzende Polygone nicht die selbe Kante miteinander teilen (T-Crack) (siehe Skizze)

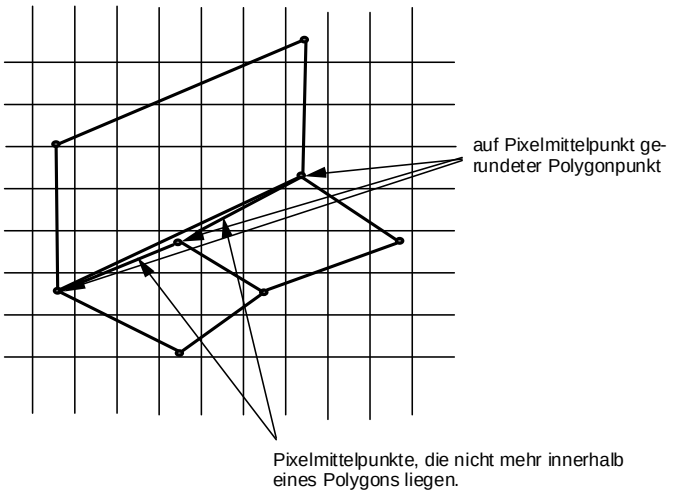


A und B bzw A und C grenzen aneinander, in der Datenstruktur wird ihre gemeinsame Grenze nicht durch dieselbe Kante representiert.

Das führt bei vielen Algorithmen wie beim Rendering, beim Verschneiden zweier Objekte oder beim einfachen IN-OUT-Test zu falschen Ergebnissen.

Z.B.: Beim IN-OUT-Test wird ein Strahl mit allen Polygonen geschnitten um den vordersten Treffer zu finden.Dieser Strahl kann aus numerischen Gründen zwischen den beiden Polygonen durchgehen und damit falsche Ergebnisse liefern.

Bei der Rasterkonversion der Polygone können Pixel-drop-outs entstehen (siehe Skizze).



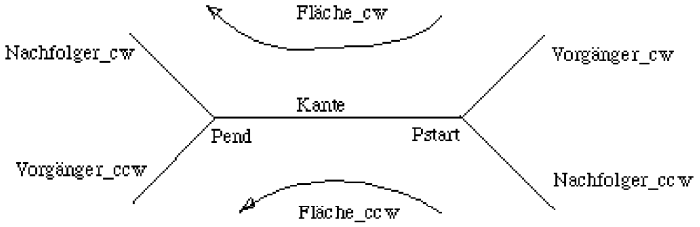
Die hierarchische Stuktur der B-Rep ist nicht in allen Fällen zielführend.Will man, wie im letzten Beispiel, Informationen von den tieferen Ebenen (also z.B. von den Punkten) in die höheren Ebenen (also z.B. Polygone) propagieren, so ist das durch die Hierarchie unmöglich.Mit dieser Struktur kann man auch keine Nachbarschaftsbeziehung feststellen, d.h. das Auffinden von benachbarten Polygonen eines vorgegebenen Polygons kann nur durch testen aller anderen Polygone in der Datenstruktur erfolgen.

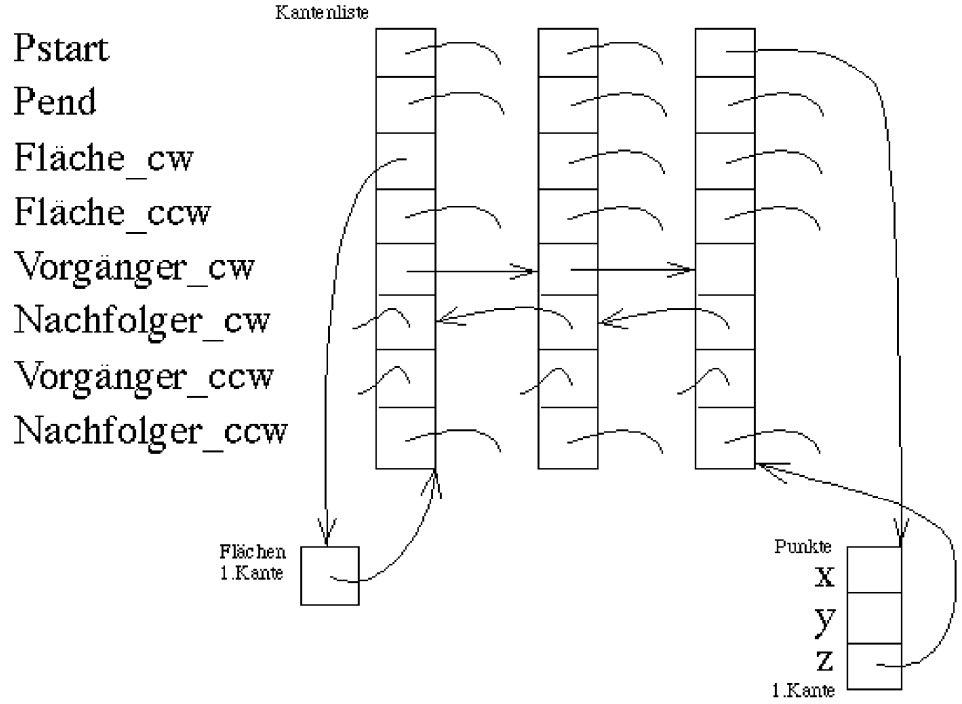
Eine Winged-Edge Datenstruktur bietet Abhilfe:

Winged-Edge-Datenstruktur

Die Zentrale Datenstruktur ist dabei die Kante eines Objektes. Die Kanten werden in einer Liste verwaltet und jede Kante besitzt Verweise auf die beiden Punkte, (P_{start} und P_{end}) die sie begrenzt, als auch Verweise auf die beiden Flächen (Fläche_cw und Fläche_ccw), die sie trennt.

Für die vollständige Definition einer Fläche gibt es noch vier Verweise auf Nachfolge- bzw. Vorgängerkanten in Bezug auf die beiden Flächen. Also je ein Verweis auf die Vorgängerkante und einer auf die Nachfolgekante die zur Fläche_cw gehören, und entsprechend zwei Verweise für die andere Fläche_ccw.



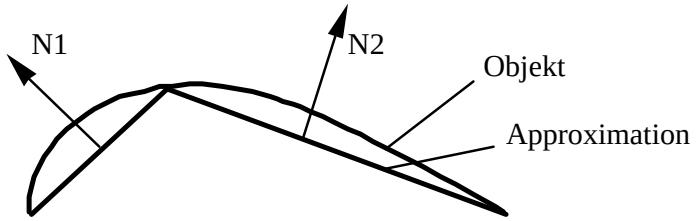


Zerteilung von Objektoberflächen:

Bei runden Flächen muß für viele Renderingverfahren die Fläche in kleine ebene Teilflächen zerlegt werden:

Parametrisierbare Oberflächen (Freiformflächen, Quadrics, Superquadrics):
Der zweidimensionale Parameterraum wird unterteilt, die Kurven des Parameterraums werden als (Geradenstücke) approximiert. Jedem Teilintervall des Parameterraums entspricht eine Teilfläche.

Nicht parameterisierbare Oberflächen (Verbogene Polygone):
Die nicht-planaren Polygone werden in kleinere Polygone (meist Dreiecke) zerlegt. Dieser Vorgang wird als **Tesselation** bezeichnet.



Wenn die Normalvektoren N_1, N_2 von zwei benachbarten Oberflächen fast die gleiche Richtung haben, müssen die beiden Oberflächen nicht mehr unterteilt werden; rechnerisch wenn

$$N_1 \cdot N_2 \geq 1-\varepsilon.$$

Rendering:

Man braucht zur Darstellung unbedingt ein Hidden-Line oder Hidden-Surface Verfahren, das meist mit der Polygonliste arbeitet.

Speicherplatz :

Es gibt auch hier wieder Ersparnis von Speicherplatz durch nur einmaliges Abspeichern von Kanten, die zu mehreren Polygonen gehören. Muß man aber wieder gekrümmte Oberflächen approximieren, so entsteht wie im Kantenmodell ein hoher Speicheraufwand, weil viele Polygone für eine gute Approximation notwendig sind.

Vorteile:

+ Transformationen sind leicht zu implementieren.
+ Generelle Repräsentationsform für viele Objekte.
+ Modellgenerierung durch Digitalisierung realer Objekte möglich.

Nachteile:

- Hoher Speicheraufwand (Punkteliste, Kantenliste, Flächenliste).
- Kombinationen relativ aufwendig und numerisch problematisch.
- Alle runden Objekte müssen approximiert werden.

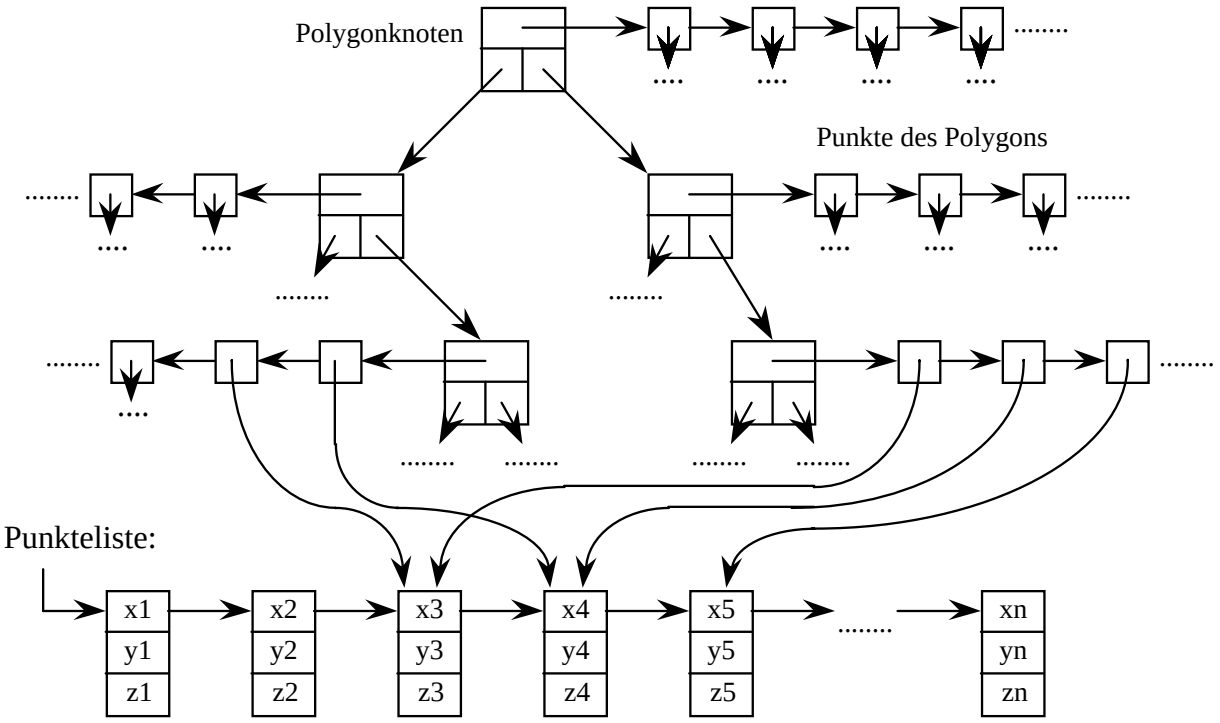
1.2.4 Binary Space Partioning Tree

Der Binary Space Partioning Tree (BSP-Baum) ist eine spezielle Art der Boundary Representation. BSP-Bäume eignen sich vor allem für statische Szenen, deren interne Darstellung sich nicht sehr oft ändert (keine animierten Objekte, wo für jedes Bild der Animation die interne Repräsentation verändert wird). Dabei werden die Polygone in eine gewisse Ordnung zueinander gebracht. Diese Ordnung erleichtert das Ausgabeverfahren erheblich. Man kann aus dem BSP-Baum bei beliebigem Augpunkt die Polygone von hinten nach vorne auf den Bildschirm zeichnen. Weiter hinten liegende werden so von weiter vorne liegenden übermalt und dadurch verdeckt. Diese Datenstruktur eignet sich vor allem für die zumeist statische Umgebungsszenerie in Flug- oder Autosimulatoren oder in Spielen, da die Kameraposition (Flugzeug/Auto/Spieler) verändert und die statische Szenerie mit richtiger Sichtbarkeit rasch dargestellt werden kann, ohne einen zeitaufwendigeren Hidden-Surface Algorithmus verwenden zu müssen.

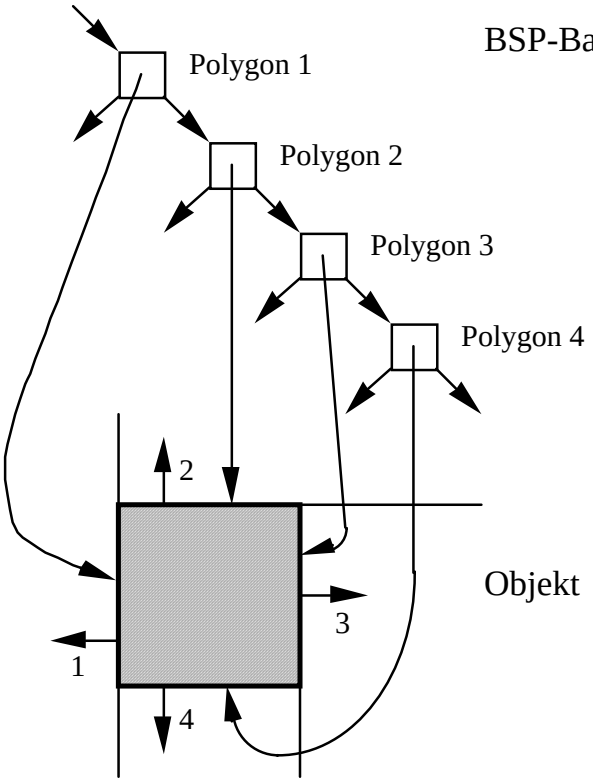
Datenstruktur:

Objekt = Verweis auf einen Knoten
Knoten = Polygon, zwei Verweise auf Knoten

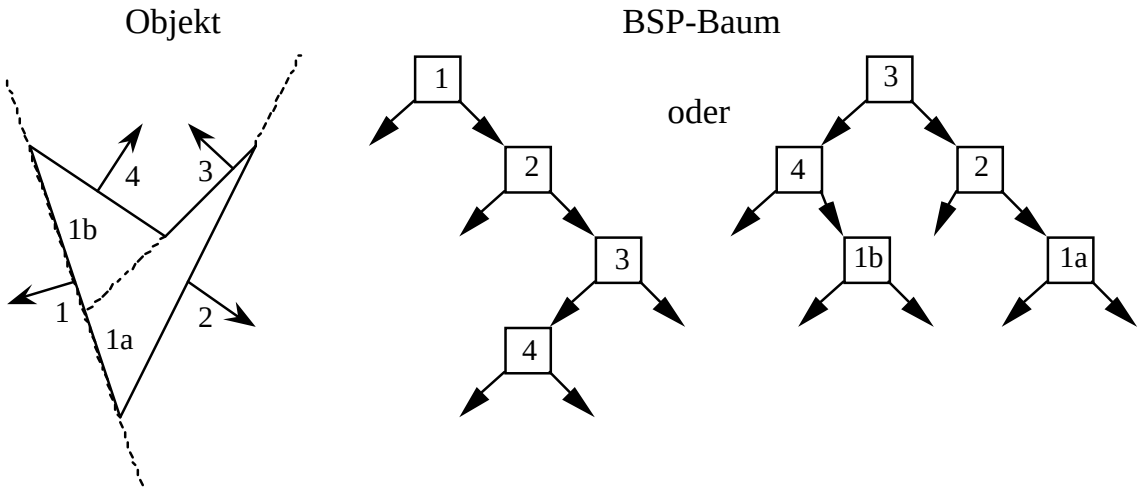
Polygon siehe B-rep

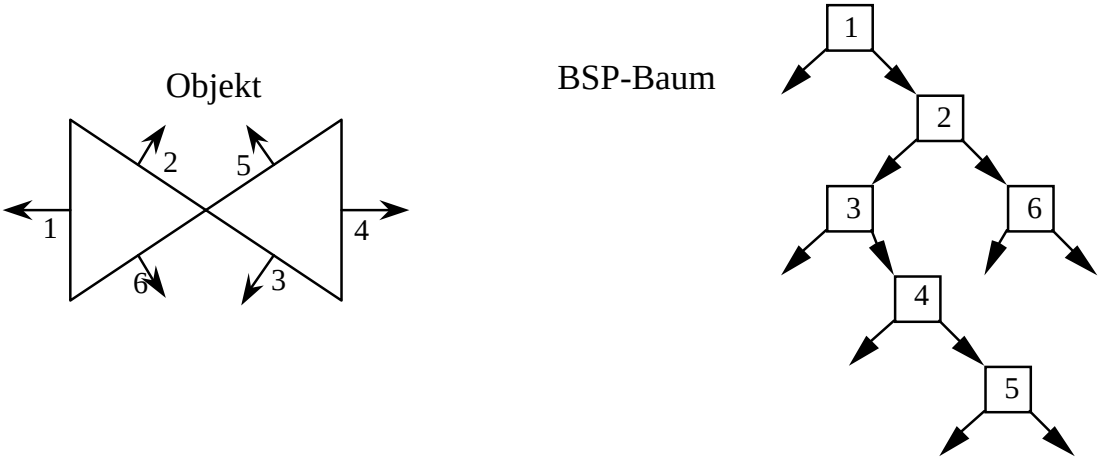


In dem binären Baum ist in jedem Knoten ein Polygon enthalten. Dessen Trägerebene (das ist die Ebene, in der das Polygon liegt) zerteilt den Raum in zwei Teile, in den Raum vor dem Polygon und in den Raum hinter dem Polygon. Die beiden Teilbäume dieses Knotens enthalten nur mehr Polygone, die entweder in dem einen oder im anderen Teilraum liegen. Falls nötig, müssen Polygone, die in beiden Teilräumen liegen, zerschnitten werden. So wird der Raum, in dem die Szene definiert ist, durch seine Polygone in viele Teilräume zerschnitten.



Zwei weitere Beispiele zu BSP-Bäumen:





Im BSP-Baum ist auch die Frage, ob ein Punkt innerhalb oder außerhalb eines Objektes liegt, einfach durch einen Abstieg in den Baum zu beantworten: je nach dem ob man im relevanten Blatt des Baumes nach links oder nach rechts weitergehen müßte, um den betrachteten Punkt aufzusuchen, liegt der Punkt innerhalb oder außerhalb des Objekts.

Aufbau eines BSP-Baumes:

1. Für konvexe Objekte ist der BSP-Baum trivial (lineare Liste).
2. Sonst wird die Flächenliste der Boundary Representation mit folgendem Algorithmus in einen BSP-Baum umgewandelt:
   i. Suche das Polygon aus der Liste, das die wenigsten anderen Polygone mit seiner Trägerebene schneidet (in der Praxis nimmt man das Beste aus drei oder fünf zufällig gewählten).
   ii. Teile die Flächenliste in zwei Teile: in Polygone, die vor bzw. hinter der Trägerebene liegen.
   iii. Das unter (i) gefundene Polygon wird Wurzel des BSP-Baumes. Mit den beiden unter (ii) erstellten Listen erzeuge den linken und rechten Teilbaum rekursiv.

Bei **Transformationen** müssen sowohl die Punkte in der Punkteliste, als auch die Ebenengleichung und der Normalvektor transformiert werden.

Bei **Kombinationen** hat man zwei Möglichkeiten: Entweder man wendet die Kombination auf die Boundary Representation an und erzeugt erst dann den neuen BSP-Baum, oder man wendet die Kombinationsoperatoren direkt auf den BSP-Baum an, worauf im folgenden noch näher eingegangen wird (schneller als eine Kombination der Boundary-Representations).

Bei der **Kombination** von BSP-Bäumen steht man vor dem Problem, das Objekt A (durch Polygone a im BSP-Baum A definiert) mit dem Objekt B mithilfe des Operators op zu kombinieren.

Gegeben seien zwei Objekte, repräsentiert durch die BSP-Bäume A und B, gesucht das Ergebnis C = A op B der Anwendung des Operators op auf die beiden Objekte.

Algorithmus zur Durchführung dieser Kombination:

- Man schneidet das Polygon a des Wurzelknotens A mit dem Objekt B. Dadurch erhält man zwei Teile

  a^{in} = Teil von a im Objekt B

  und a^{ex} = Teil von a außerhalb von B

des Polygons a, die innerhalb bzw. außerhalb des Objekts B liegen.

- Außerdem nimmt man die Trägerebene von a des Wurzelknotens A und spaltet damit das Objekt B in zwei Teile.

B^- = Teil von B hinter der Trägerebene von a

und B^+ = Teil von B vor der Trägerebene von a

Der Wurzelknoten C des Ergebnisses erhält die Trägerebene von a als Ebene und je nach Operator entweder a^{in} oder a^{ex} als Polygonmenge (bei Vereinigung und Differenz wird a^{ex} genommen, bei Schnitt a^{in}). Die Unterbäume von C werden rekursiv aus den Unterbäumen A_l und A_r von A und den Spaltprodukten B^- und B^+ von B berechnet:

$$C_l = A_l \text{ op } B^-$$

$$C_r = A_r \text{ op } B^+ .$$

Die Rekursion bricht ab, wenn einer der beiden Operanden einen homogenen Teilraum repräsentiert, also ein Blatt ist (voll oder leer) In diesem Fall wird das Operationsergebnis nach der folgenden Tabelle ermittelt:

Einfache Kombinationsregeln			
op	A	B	A op B
or	inhomogen	voll	voll
	inhomogen	leer	A
	voll	inhomogen	voll
	leer	inhomogen	B
and	inhomogen	voll	A
	inhomogen	leer	leer
	voll	inhomogen	B
	leer	inhomogen	leer
minus	inhomogen	voll	leer
	inhomogen	leer	A
	voll	inhomogen	¬B
	leer	inhomogen	leer

Rendering:

Die BSP-Bäume eignen sich besonders gut zur schnellen Darstellung mit richtiger Sichtbarkeit der Objekte, wobei folgender rekursiver Algorithmus Anwendung findet (ein Algorithmus, bei dem unsichtbare Polygone durch sichtbare übermalt werden heißt Painters Algorithmus):

Will man die Polygone von hinten nach vorne ausgeben, so muß man nur den Baum in der richtigen Reihenfolge abarbeiten. Zuerst werden alle Polygone im Teilraum, in dem sich der Augpunkt nicht befindet, ausgegeben, dann das Polygon im jeweiligen Knoten und dann alle Polygone, die auf der gleichen Seite der Teilungsebene liegen wie der Augpunkt. So wird

jeder Knoten des Baumes nur einmal besucht, die Polygone werden ohne Sortieraufwand in richtiger Sichtbarkeit ausgegeben.

Liegt der Augpunkt im positiven Halbraum von Polygon a:

1. Ausgabe von A^-
2. Zeichne a
3. Ausgabe von A^+

Sonst (Augpunkt im negativen Halbraum von Polygon a):

1. Ausgabe von A^+
2. Zeichne a
3. Ausgabe von A^-

Die Polygone werden dadurch von hinten nach vorne ausgegeben, wodurch weiter vorne liegende Polygone weiter hinten liegende überschreiben.

Backface Culling: Bei geschlossenen Objekten brauchen Polygone mit nach hinten schauendem Normalvektor nicht ausgegeben werden (das gilt auch für die Boundary-Representation):

$$(\text{Polygon.NV} \cdot \text{Blickrichtung}) > 0 \text{ bedeutet also nicht zeichnen}$$

Vorteile:

+ Leichte Transformierbarkeit.
+ Schnelle Darstellbarkeit mit richtiger Sichtbarkeit.
+ Universell (wie die Boundary-Representation).
+ Modellgenerierung durch Digitalisierung realer Objekte möglich

Nachteile:

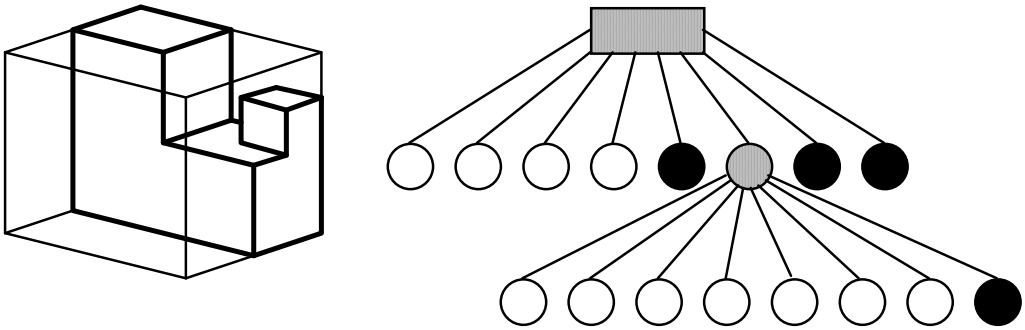
- Alle runden Objekte müssen approximiert werden.
- Nur konvexe Polygone.
- Hoher Speicheraufwand.

1.2.5 Octree

Octrees sind ein Volumsmodell (sie repräsentieren das Volumen direkt in der Datenstruktur). Der große Vorteil ist, daß die Kombinationen direkt am Modell rasch und schnell durchgeführt werden und nicht erst im Darstellungsprozeß berücksichtigt werden müssen. Dadurch wird der Darstellungsprozess einfacher und schneller. Einfache Octrees arbeiten mit dem Prinzip der Raumteilung ("Space Subdivision"): Der durch einen Octree-Knoten repräsentierte Raum wird dabei jeweils durch die drei Normalebenen durch seinen Mittelpunkt in 8 Teilräume unterteilt, die seine Nachfolger im Octree sind. Die Trennebenen sind dabei nicht objektabhängig.

Primitive:

Jedes Objekt wird durch einen zugehörigen Octree dargestellt, der aus vielen verschieden großen Würfeln besteht. Jedem Würfel kann man eine von drei Eigenschaften zuordnen: voll (auch schwarz oder S-Knoten), leer (auch weiß oder W-Knoten) oder inhomogen => der Würfel muß noch unterteilt werden (grau oder G-Knoten). Die Weiterunterteilung in acht Teilwürfel geschieht solange, bis eine ausreichende Approximation des wirklich darzustellenden Objekts erreicht ist. Allerdings gehen dabei die geometrischen Informationen über die Oberflächen des Objekts verloren.



G (W W W W S G (W W W W W W W S) S S)

Datenstruktur:

Ein Octree ist ein acht-wertiger Baum, in dem jeder Teilbaum einen Teilwürfel repräsentiert. Jeder Teilbaum kann die Information "voll", "leer" oder einen weiteren acht-wertigen Teilbaum enthalten, je nach dem wie sein zugehöriger Würfel definiert wurde.

Objekt = Verweis auf einen Knoten

Knoten = entweder Voll, Leer oder 8 Verweise auf Knoten.

Transformationen:

Sind sehr schwer zu implementieren. Nur einige wenige können mit vernünftigen Aufwand berechnet werden: Rotation um 90 Grad, Spiegelung an einer Trennebene, Skalierung mit Zweierpotenz, o.ä. Man muß die Elementarobjekte schon in der richtigen Lage und Größe erzeugen.

Kombinationen:

Sind leicht durchzuführen. Die Implementierung braucht nur boole'sche Operationen ausführen und ist daher extrem schnell.Will man zwei Octrees (im weiteren A und B genannt) mit dem Operator op kombinieren, so verwende man folgenden Algorithmus:

- Sind A und B homogen, verwende die Boole´schen Regeln.
- Sonst kombiniere (rekursiv):
  - den Oktant 1 von A und B,
  - den Oktant 2 von A und B,

    ...
  - und den Oktant 8 von A und B.

Rendering:

Für das Octreemodell gibt es eigene Darstellungsalgorithmen, die im wesentlichen den Baum in Abhängigkeit vom Blickpunkt abarbeiten und auf das Rasterbild abbilden. Problematisch sind die dabei entstehenden Ecken und Kanten die aus der Approximation des Objekts durch die Octreewürfel entstehen (Lego™-Oberfläche).

Da alle Objekte nur aus Würfeln bestehen, können diese beim **Rendering** von hinten nach vorne ausgegeben werden:

- Ist der Octree ein W-Knoten, tue nichts.
- Ist der Octree ein S-Knoten, projiziere den Würfel auf die Bildebene.
- Ist der Octree ein G-Knoten, berechne die Reihenfolge der Ausgabe der untergeordneten Knoten für den gewünschten Augpunkt (entfernte zuerst), und gib diese in der berechneten Reihenfolge aus (Rekursion).

Speicherplatz:

Objekte mit krummen oder nicht orthogonalen Oberflächen können nur mit großem Speicheraufwand (viele kleine Knoten) approximiert werden.

Vorteile:

+ Kombinationsoperatoren sind sehr leicht zu implementieren.
+ Einfache Darstellbarkeit.
+ Räumliches Suchen sehr schnell.
+ Modellgenerierung durch Digitalisierung realer Objekte möglich.

Nachteile:

- Hoher Speicheraufwand für approximierte Objekte.
- Beschränkte Transformierbarkeit.
- Unexakte Repräsentation allgemeiner Objekte.

1.2.6 Erweiterter Octree

Eine mögliche Erweiterung von Octrees besteht durch die Einführung von neuen Knotenarten:

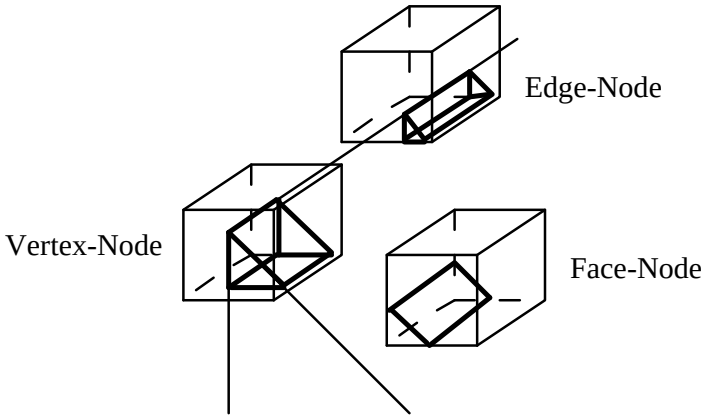
- Face Nodes für Flächen
- Edge Nodes für Kanten
- Vertex Nodes für Punkte

Dadurch wird sowohl Speicherplatz gespart als auch das Objekt exakter repräsentiert (glatte Oberflächen, Normalvektoren bestimmbar).

Der erweiterte Octree wird wie folgt aufgebaut:

- Erzeuge eine Liste von Flächen und eine Liste von Punkten (siehe Boundary Representation).
- Teile diese beiden Listen in 8 Punkte bzw. Flächenlisten durch Clipping an den Teilungsebenen

- Wenn eine dieser 8 Doppellisten leer ist: Schwarz/Weiß
  - Wenn nur noch ein Punkt enthalten ist: Vertex
  - Wenn nur noch zwei Flächen enthalten sind: Edge
  - Wenn nur noch eine Fläche enthalten ist: Face



Kombination von erweitertem Octree A und B:

- Ist Octree A oder Octree B homogen (Schwarz oder Weiß) werden die Boole´schen Regeln angewendet.
- Ist Octree A und Octree B inhomogen (Grau), so werden alle acht untergeordneten Bäume von A und B kombiniert (Rekursion).
- Ist A homogen (Edge, Vertex, Face) und B inhomogen (Grau) oder umgekehrt, werden die acht untergeordneten Bäume mit dem homogenen Knoten kombiniert (Rekursion).
- Sind A und B homogen (Edge, Vertex, Face), so werden die Ergebnisknoten geometrisch berechnet.

Rendering:

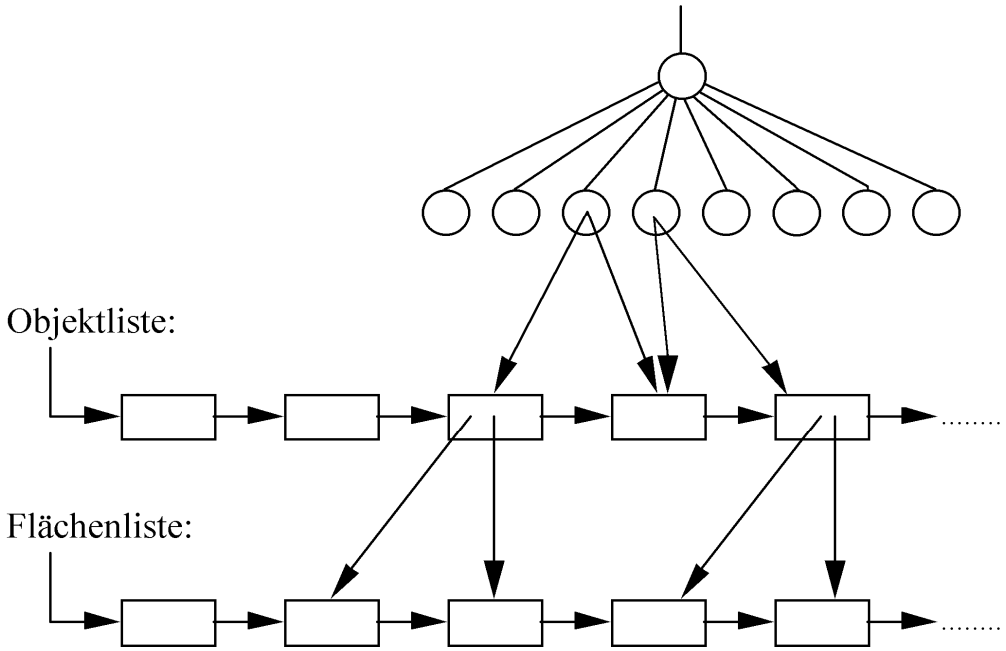
Rückumwandlung des Octrees in die Boundary Representation:

- Aus den Vertex-Knoten des Octrees wird wieder die Punkteliste erzeugt.
- Aus den Flächen-, Kanten und Punktknoten wird die Flächenliste berechnet.

1.2.7 Octrees als räumliche Directories

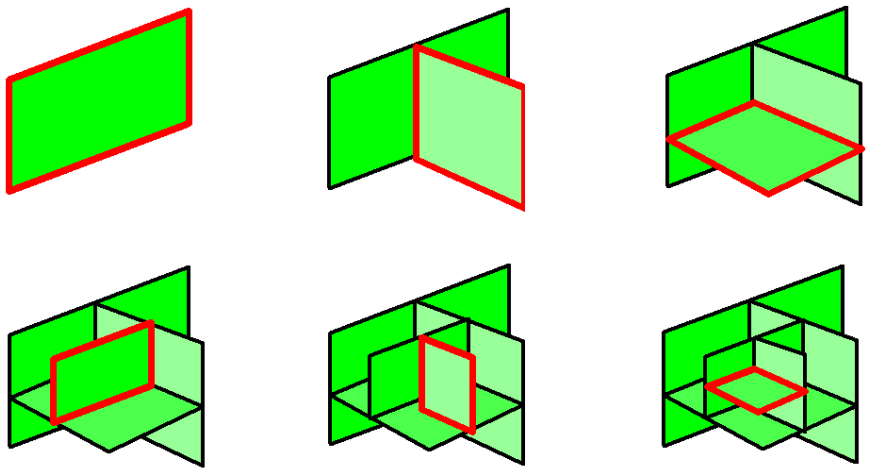
Der Octree wird vor allem auch als Suchstruktur für Objekte verwendet, die zu einer Szene zusammengefaßt sind. Die geometrischen Eigenschaften der Objekte bleiben dabei erhalten, und die Tiefe der Octree-Unterteilung muß nicht so groß sein. Für die darunterliegende Datenstruktur der Objekte sind die verschiedensten Repräsentationen denkbar. Es kann natürlich vorkommen, daß ein Objekt in mehreren Blättern des Octrees aufgenommen werden muß.

Durch diese Directories erreicht man sowohl ein schnelles Suchen als auch ein effizientes Darstellen der Objekte.


Objektliste:
Flächenliste:

1.2.8 Bintree

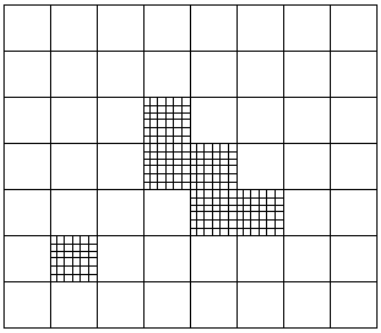
Der Bintree eignet sich ebenfalls als räumliche Suchstruktur für Objekte in der Szene. Im Gegensatz zum Octree wird ein Knoten dabei jeweils nur durch eine Normalebene in zwei Unterräume (Nachfolgeknoten) unterteilt. Diese Trennebene geht dabei nicht notwendigerweise durch den Mittelpunkt des Knotens, sondern kann auch so gewählt werden, daß sie den Raum möglichst optimal unterteilt (z.B. gleich viele Objekte in jedem Unterraum). Dadurch wird es ermöglicht daß ein Bintree für die selbe Szene weniger Knoten benötigt als ein Octree.



1.2.9 Grid

Als räumliche Suchstruktur für Objekte eignet sich auch eine regelmäige Unterteilung des Raums in Form eines 3D Grids. Der Vorteil im Gegensatz zu Bäumen ist hierbei daß man nur eine Ebene hat, in der man die einzelnen Zellen des Grids direkt adressieren kann. Aus diesem Grund ist es in einem Grid z.B. besonders einfach und effizient (konstanter Aufwand) die Nachbarzellen zu einer bestimmten Zelle zu finden. Der Nachteil eines Grids ist klarerweise das man bei der Wahl seiner Zellenanzahl einen Kompromiß eingehen muß. Wählt man zu wenige (große) Zellen, so liegen jeweils viele Objekte in einer einzelnen Zelle. Wählt man zu viele (kleine) Zellen, so wird der Speicheraufwand für das Grid zu groß und man muß zu viele Zellen besuchen um eine gegebene Strecke entlang zu gehen (z.B. bei Raycasting).

Um diese Nachteile zu veringern und die Vorteile im wesentlichen zu erhalten, kann man hierarchische Grids verwenden. Hierbei verwendet man Grids mit eher wenigen Zellen, und Zellen mit zu vielen Objekten unterteilt man weiter, indem man in solch einer Zelle ein Sub-Grid einfügt.



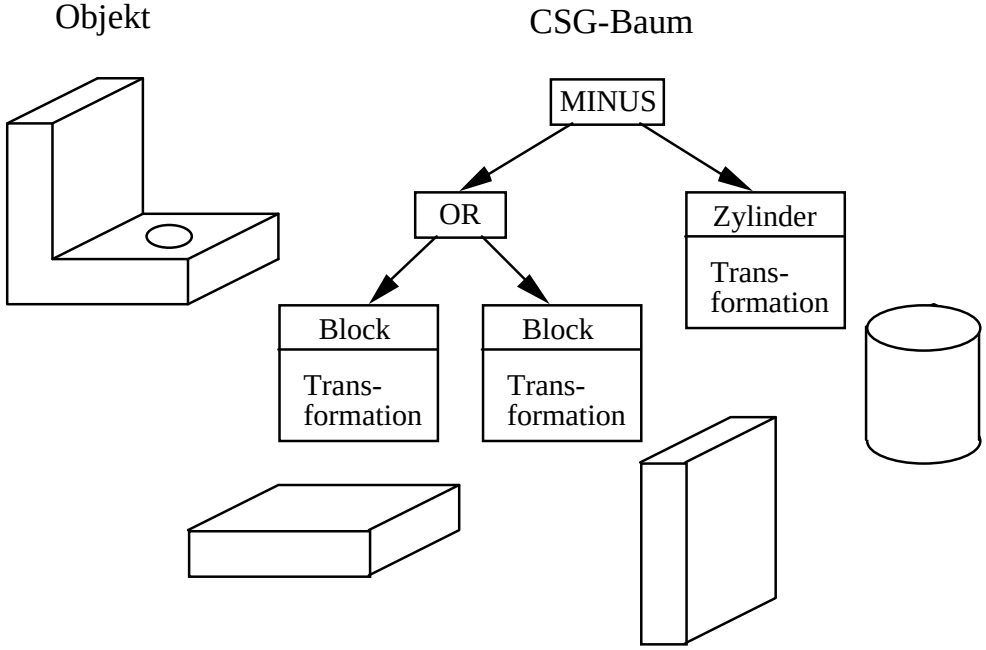
1.2.10 Contructive Solid Geometry (CSG)-Tree

CSG ist ein Volumenmodell, bei dem Objekte durch eine Hierarchie von Kombinationen von Elementarobjekten repräsentiert werden. Der Darstellungsalgorithmus verwendet den CSG-Baum, um das richtige Aussehen und die richtige Lage der kompletten Objekte zu berechnen. Der CSG-Baum stellt daher eine exakte Repräsentation der Objekte dar, im Gegensatz zu polygonalen Strukturen (z.B. Boundary-Representation), wo krumme Objekte durch eine Vielzahl von Polygonen approximiert werden müssen. Die Probleme, solche Objekte darzustellen, werden also auf den Renderingprozeß abgeschoben.

Es besteht dabei auch die Möglichkeit, vom CSG-Modell auf das Flächenmodell überzugehen, indem man die Elementarobjekte durch Polygone approximiert, diese Polygone dann transformiert und, wenn nötig, so verändert, wie die Kombination es vorschreibt; also schneidet oder Teile wegfallen läßt.

Datenstruktur:

Das Objekt wird in einem binären Baum - dem CSG-Baum - aufgehoben, dessen *Zwischenknoten* die Kombinationen repräsentieren, und dessen *Endknoten* die Elementarobjekte und die darauf angewendeten Transformationen beinhalten.



Objekt = Verweis auf einen Knoten
Knoten = entweder
Zwischenknoten: Kombination zweier Objekte, oder
Endknoten: Primitiv und Transformation.

Primitive:

Wenige Elementarobjekte (nur die Einheitsobjekte mit Seitenlängen, Höhen und Radien = 1) genügen, sie werden durch Transformationen und Kombination verändert. Als Elementarobjekte werden z.B. Würfel, Kugel, Kegel, Zylinder, Torus, Polyeder, Sweeps, prozedurale Objekte, Freiformflächen oder Terrains verwendet.

Kombination:

Um zwei Objekte zu **kombinieren,** erzeugt man einen neuen Kombinationsknoten und hängt die Operanden als linken und rechten Unterbaum ein.

Transformationen:

Bei jedem Elementarobjekt wird eine Transformationsmatrix aufakkumuliert, die angibt, wie das Elementarobjekt transformiert werden soll. Das gesamte Objekt wird **transformiert,** indem alle Matrizen des Baumes mit der Transformationsmatrix multipliziert werden.

Rendering:

Bei der Darstellung hat man zwei Möglichkeiten: entweder man wandelt den CSG-Baum in eine Boundary Representation um und wendet dann einen Hidden Surface Algorithmus an, oder man wählt die direkte Darstellung durch Ray-Tracing oder ein anderes an diese Datenstruktur angepaßtes Sichtbarkeitsverfahren.

Es gibt nur sehr wenige Verfahren, die CSG-Modelle direkt darstellen. Diese erzeugen aber am ehesten realistische Bilder, da bei der CSG-Darstellung alle geometrischen Informationen über die Objekte exakt erhalten bleiben. Im allgemeinen wird man aber das CSG-Modell in ein Flächenmodell konvertieren und dann ein Hidden-Surface Verfahren anwenden. Dadurch spart man Rechenzeit, verzichtet aber auf einen gewissen Grad an Realismus. Allerdings erreicht man bei Algorithmen, die an CSG angepaßt sind und so hochwertige Verfahren wie Ray-Tracing verwenden, schon ähnliche Geschwindigkeiten, wie wenn man den Umweg über ein Flächenmodell geht.

Speicherplatz :

Für die Kombinationsknoten wird kaum Speicherplatz verbraucht, da man nur den Operator aufheben muß. Bei der Verwendung von Einheitsobjekten als Elementarbilder brauchen nur die Transformationsmatrizen den Platz, egal ob das Elementarobjekt krumme Oberflächen hat oder nicht. Das CSG-Modell ist die speicherplatzsparendste und exakteste Darstellung.

Vorteile:

+ Sehr wenig Speicherplatz (besonders für krumme Objekte).

+ Kombinationen und Transformationen simpel.
+ Exakte Darstellbarkeit vieler Objekte (auch mit krummen Flächen).
+ Baumstruktur (Aufwand ld n bei Suchen im Baum).

Nachteile:

- Renderingverfahren sind langsam.
- Modellgenerierung durch Digitalisierung realer Objekte i.a. nicht möglich.

1.2.11 Sequentielle Repräsentation

Bei der sequentiellen Repräsentation wird auf jegliche Struktur verzichtet. Die Objekte liegen nacheinander in ungeordneter Reihenfolge vor. Es gibt daher keine Kombinationsoperatoren. Die Anzahl der Objekte ist nicht durch den Speicherplatz begrenzt.

Auch bei der Struktur der einzelnen Objekte unterliegt man keinen Beschränkungen. Denkbar sind Polygone, Elementarobjekte, Sweeps, prozedurale Objekte, aber auch ganze CSG-Bäume. Geeignet ist diese Repräsentation für realistische Bilder mit sehr vielen oder sehr komplizierten Objekten (z.B. Regentropfen), die willkürlich nacheinander dargestellt (z.B. durch einen Z-Buffer-Algorithmus) und wieder weggeworfen werden.

Vorteile:

+ Keine Größenbeschränkung durch den verfügbaren Hauptspeicher. Es können beliebig viele Objekte in einer Szene zusammengefaßt werden.
+ Exakte Repräsentation der Objekte möglich.

Nachteile:

- Keine oder nur beschränkte Interaktionsmöglichkeit.
- Keine Zusammenhänge zwischen den Objekten.