

Bauhaus Universität Weimar
Institut für Strukturmechanik

Diplomarbeit

Konzept und Implementierung einer Geometrie-Datenstruktur
in das Programmsystem Slang basierend auf der
„Boundary Representation Data Structure“

eingereicht von: Schneider, David
geb. am 09.05.1980
in Gera/Thüringen

Erstprüfer: Prof. Dr. techn. Christian Bucher
Prof. Dr.-Ing. habil. C. Könke

Zweitprüfer: Dipl.-Ing. S. Eckardt
Dipl.-Ing. T. Luther

Bearbeitungszeitraum: 20.03.2006-20.09.2006



Aufgabenstellung zur Diplomarbeit 01/2006

Student: David Schneider
Matr.-Nr.: 10022
Bearbeitungszeitraum: 20.03.2006 – 20.09.2006
Erstprüfer: Prof. Dr. techn. Christian Bucher
Prof. Dr.-Ing. habil. C. Könke
Zweitprüfer: Dipl.-Ing. T. Luther

Thema:

Konzept und Implementierung einer Geometrie-Datenstruktur in das Programmsystem SLang basierend auf der „Boundary Representation Data Structure“

Aufgabenstellung:

Eine wichtige Grundlage für die numerische Simulation physikalischer Prozesse ist die geometrische Modellierung des zugrundeliegenden realen dreidimensionalen Objektes. Dabei wird in rechnergestützten Technologien die Beschreibung der Geometrie und der Topologie vom sogenannten Körpermodellierer übernommen. Der Kern eines Körpermodells ist seine Datenstruktur. Die meisten modernen Datenstrukturen für die 3D-Modellierung von Körpern basieren auf dem Boundary Representation (BRep)-Datenmodell. Darin wird eine Trennung von geometrischen und topologischen Informationen vollzogen. Mit Hilfe von BRep-Datenstrukturen können komplexe Geometrien aus den Verknüpfungen ihrer Volumen, Flächen, Kanten und Knoten beschrieben werden.

Im Rahmen der Diplomarbeit soll der Kandidat eine solche Datenstruktur zur Körpermodellierung in das am ISM genutzte Programmsystem SLang implementieren. Dazu kann eine objektorientierte Programmierung in C++ erfolgen. Die Datenstruktur soll ausschließlich der effizienten Verwaltung geometrischer Informationen dienen. Es ist zu überprüfen, in wie weit die am ISM vorliegenden Algorithmen von Röhrig [4], die in das FE-Programmsystem FRANC3D implementiert wurden, nutzbar sind. Im Rahmen der Diplomarbeit sollen innerhalb der zu implementierenden Datenstruktur einfache geometrische Objekte angelegt werden, die eine Beschreibung der am Institut entwickelten Mesomodelle für Beton ermöglichen. Diese Beschreibung basiert in 3D auf Quadern und Ellipsoiden sowie in 2D auf Rechteckformen und Ellipsen. Die Visualisierung geometrischer Objekte soll unter Nutzung der in SLang bereits implementierten Datenstruktur erfolgen. Eine Implementierung von Booleschen Operationen zur Anwendung auf geometrische Objekte ist in der Diplomarbeit nicht vorgesehen. Als Grundlage für eine spätere Implementierung soll aber abschließend eine Literaturrecherche zur algorithmischen Formulierung Boolescher Operationen erfolgen.

Im Rahmen dieser Aufgabenstellung sind vom Kandidaten folgende Punkte zu bearbeiten:

- Literaturlauswertung zum Thema "Boundary Representation Data Structure".
- Erarbeitung eines Konzepts für die Einbindung einer geometrischen Datenstruktur in SLang basierend auf einem BRep-Datenmodell.
- Implementierung dieser geometrischen Datenstruktur in SLang.
- Implementierung einfacher Geometriebeschreibungen und Test der Geometrie-Datenstruktur am Beispiel des geometrischen Mesomodells von Beton.
- Anknüpfung der Visualisierung geometrischer Objekte an die Geometriedatenstruktur in SLang.
- Literaturrecherche zur algorithmischen Formulierung Boolescher Operationen.
- Schriftliche Dokumentation der Arbeit.
- Dokumentation der erstellten und implementierten Algorithmen auf CD-Rom

Literatur:

- [1] Jonathan Corney, Theodore Lim: "3D Modeling with ACIS", 2002
- [2] Martti Mäntylä: "An Introduction to Solid Modeling", 1988
- [3] Kevin J. Weiler: "Topological Structures for Geometric Modeling", Ph.D., 1986
- [4] Jackson Röhrig: "Topologische Datenstrukturen für Körper- und Netzmodellierung", Dissertation, Ruhr-Universität Bochum, 1998
- [5] Dokumentationen zu C und C⁺⁺ Programmierung

Inhaltsverzeichnis

1	Einleitung	6
2	Grundlagen	8
2.1	Anforderungen an die Datenstruktur	8
2.2	Topologische Grundlagen	9
2.3	Bekannte Geometriedatenmodelle	11
3	Radial-Edge Data Structure (RED)	17
3.1	Geometrische Beschreibungen	18
3.2	Topologische Beschreibungen	20
3.3	Operatoren	26
4	Boolesche Operationen	32
4.1	Operationen	33
4.2	Lageinformation	35
4.3	bounding boxes	39
5	Implementation in SLang	43
5.1	Datenstruktur	44
5.2	Kommandos und Befehle	47

<i>INHALTSVERZEICHNIS</i>	5
5.3 Beispiele	53
6 Zusammenfassung	59
Literaturverzeichnis	62
Abbildungsverzeichnis	66
Anhang	67
A.1 CD-ROM	67
A.2 Erklärung	68
A.3 Thesen	1

Kapitel 1

Einleitung

Die numerische Auswertung physikalischer Prozesse erfordert eine Modellierung der realen Welt. Grundlage bildet dabei eine realistische und konsistente Beschreibung der Geometrie. Das Programmsystem `SIang`, welches am ISM genutzt und weiterentwickelt wird, kombiniert stochastische Methoden mit der Finite Elemente Methode. Lineare und nichtlineare, statische und dynamische, deterministische und stochastische Probleme können gelöst werden [4]. Zur Zeit ist keine Datenstruktur zur Verwaltung von Geometrien in `SIang` implementiert. Ziel dieser Arbeit ist es, eine solche Datenstruktur in das bestehende Programmsystem `SIang` zu implementieren.

In den meisten modernen Systemen kommt die Boundary Representation Data Structure (BRep) zur Anwendung. Durch die topologische Verknüpfung der geometrischen Objekte (Knotenpunkte, Kanten, Flächen) wird dabei ein Modell beschrieben. Es ist möglich, Kanten und Flächen zu parametrisieren bzw. Ansätze zu wählen, welche nicht-linear sind. Folglich wird die BRep aufgrund der variablen Gestaltung der Verknüpfungen und der geometrischen Grundobjekte zur Darstellung komplexer Geometrien anwendbar. Durch die Kombination der BRep mit Definitionen von wichtigen Grundgeometrien (solids) entsteht eine hybride Struktur. Sie ist wesentlich effizienter und benutzerfreundlicher als reine BRep-Datenstrukturen.

`SIang` wird sowohl im wissenschaftlichen als auch im kommerziellen Bereich eingesetzt. Es ergeben sich daraus zwei Anforderungen, welche sich auf den ersten Blick ausschließen: Flexibilität und Effizienz. Durch konsequentes Nutzen objektorientierter Paradigmen, variabler Verzeigerung und Ausnutzen der durch die C-Standardlibrary gegebene-

nen Elemente werden diese Anforderungen erfüllt. Eine Beschränkung auf manifold-Systeme darf nicht erfolgen. Aus diesem Grund wird auf die in [14] und [10] beschriebene RED (Radial-Edge Data Structure) zurückgegriffen. Angehängte Daten wie z.B. Material und Informationen für die Vernetzung werden in Informationscontainern (\rightarrow EED¹) abgelegt.

Das Erstellen eines Modells erfolgt über die Make-Operatoren. Das Löschen einzelner Elemente oder des gesamten Modells erfolgt über die Kill-Operatoren. Sie garantieren die Konsistenz des Modells zu jedem Zeitpunkt [7, 14]. Die verschiedenen (Entwicklungs-) Stände können einzeln im Speicher abgelegt werden. Durch ein Interface zu einem Vernetzer wird es möglich, vom jeweiligen Modell vernetzte Strukturen zu erstellen. Diese können für die strukturmechanischen Auswertungen genutzt werden.

Die Funktionsfähigkeit wird am Beispiel eines Betonmodells nachgewiesen. Dazu werden ein Quader (lineare Beschreibungen) und verschiedene Ellipsoide (parametrische Beschreibungen) in der Datenstruktur angelegt. Durch Änderung der topologischen Verknüpfungen werden die Ellipsoide in das Volumen des Quaders eingefügt. Die entstandene Geometrie wird über ein Interface an den Vernetzer *gmsh* übergeben.

¹Extended Entity Data

Kapitel 2

Grundlagen

2.1 Anforderungen an die Datenstruktur

SIang wird ständig weiterentwickelt und verbessert. Eine Implementierung einer Datenstruktur in dieses Programmsystem setzt demnach eine Vielzahl von Abstimmungen voraus. Neben Fragen der Anbindung und Leistungsfähigkeit mussten auch Fragen der Effizienz formuliert und beantwortet werden. Innerhalb dieser Abstimmungen wurden mit Verantwortlichen und Entwicklern zunächst einige grundlegende Anforderungen formuliert. Diese sind:

- Das Programm soll flexibel in der Darstellung von Geometrien werden. Die Darstellung von nonmanifold-Systemen soll möglich sein. Aus diesem Grund wurde die Datenstruktur der RED (Radial-Edge Data Structure) gewählt.
- Attribute wie Materialparameter, Lasten oder Informationen für den Vernetzer werden an jedem Teilobjekt gespeichert. Dazu werden Informationscontainer an jedem geometrischen Objekt vorgesehen (\rightarrow EED).
- Zur Vernetzung der Geometrie ist ein Interface zu einem Vernetzer vorzusehen.
- Die Erstellung der Geometrie über Primitive soll ermöglicht werden. Eine Übersetzung in die Datenstruktur soll das Programm übernehmen.
- Effiziente Algorithmen zum Erstellen und Löschen der Struktur sollen bereitgestellt werden.

2.2 Topologische Grundlagen

Mannigfaltigkeit

Eine Mannigfaltigkeit ist ein topologischer Raum, der lokal einem gewöhnlichen Euklidischen Raum \mathbf{R}_n gleicht. Im Ganzen muss die Mannigfaltigkeit nicht einem \mathbf{R}_n entsprechen (nicht zu ihm homöomorph¹ sein) [16]. Räume mit Mannigfaltigkeit 2 werden auch als manifold bezeichnet. Sie sind homöomorph zu einer Scheibe. Für sie ergeben sich folgende Grundsätze:

1. An jede Kante stoßen genau 2 Flächen.
2. Um jede Ecke existiert nur ein einziger Ring von Flächen.
3. Flächen können sich nur an einer gemeinsamen Kante/ Ecke schneiden.
4. Die Euler Poincaré Gleichung muss erfüllt sein.

Räume, welche nicht homöomorph zu einer Scheibe sind, werden als nonmanifold bezeichnet. Dies sind z.B.:

- Körper mit gemeinsamen Kanten, Knotenpunkten oder Flächen (z.B. Finite Elemente),
- Räume mit Körpern unterschiedlicher Dimensionen oder
- zusammengesetzte Materialien.

Euler-Poincaré Gleichung

In einem Polyeder (Mannigfaltigkeit 2) gilt:

$$v - e + f = 2(s - g) + (l - f) \quad (2.2.1)$$

¹ (homöo griech. hómoio = das Gleiche, Gleichartige)(morph griech. morphé = Gestalt, Form)
Zwei Körper sind homöomorph, wenn sie im topologischen Sinn gleichartig sind. Sie lassen sich durch einen Homöomorphismus ineinander überführen.

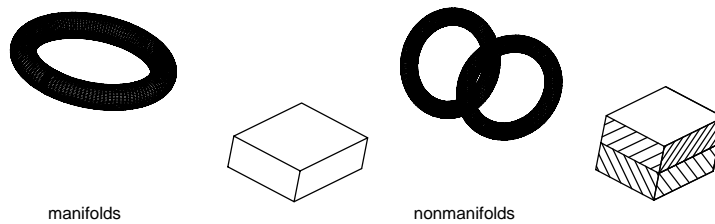


Abbildung 2.1: Beispiele für manifolds und nonmanifolds

Mit	v	...	Knotenanzahl
	e	...	Kantenanzahl
	f	...	Facetten (Flächen)-anzahl
	l	...	Schleifenanzahl
	s	...	Schalenanzahl
	g	...	Geschlecht (Anzahl der Löcher durch die Schalen)

Der Beweis wurde von L. Euler im Jahr 1751 veröffentlicht. Ausgangspunkt war das Königsberger Brückenproblem². Dabei kam es nicht auf die genaue Lage oder Größe der Brücken an, sondern darauf, wie sie über welche Gebiete verbunden sind. Die Lösung dieses Problems war der Beginn der Topologie.

Königsberger Brückenproblem: Können die Bürger von Königsberg alle sieben Brücken jeweils nur einmal in einem Zuge überqueren? Antwort: Nein

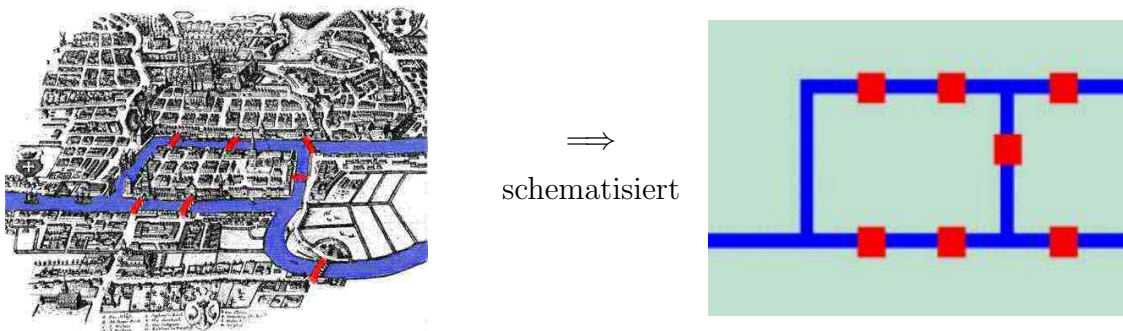


Abbildung 2.2: Königsberger Brücken [18], [19]

²L. Euler (1707-1783). 1736: In solutio problematis ad geometriam situs pertinentis (in Deutsch etwa: „Lösung eines Problems zur Geometrie der Lage“)-Lösung des Königsberger Brückenproblems Die Eulersche Polyederformel stellte H. Poincaré (1854-1912) im Jahre 1895 in einer Reihe von Arbeiten analysis situs auf eine völlig neue Basis. [17]

2.3 Bekannte Geometriedatenmodelle

CSG (Constructive Solid Geometry)

In der CSG werden geometrische Primitive durch boolesche Operationen miteinander verknüpft. Geometrische Primitive sind in der Datenstruktur fest definierte zwei- oder dreidimensionale geometrische Formen. In der Regel sind dies Quader, Kugel, Zylinder, Linie, Ellipse etc.. Die booleschen Grundoperationen sind Vereinigung (\cup , UND), Schnitt (\cap , ODER) und Differenz ($-$, MINUS). Durch das Speichern seiner Entstehungsgeschichte wird ein Körper definierbar. Es entsteht eine hierarchische Baumstruktur. Dabei werden die Elementarobjekte zu Blättern, die Mengenoperationen stellen die Knoten dar und das Resultat findet sich in der Wurzel wieder. [7]

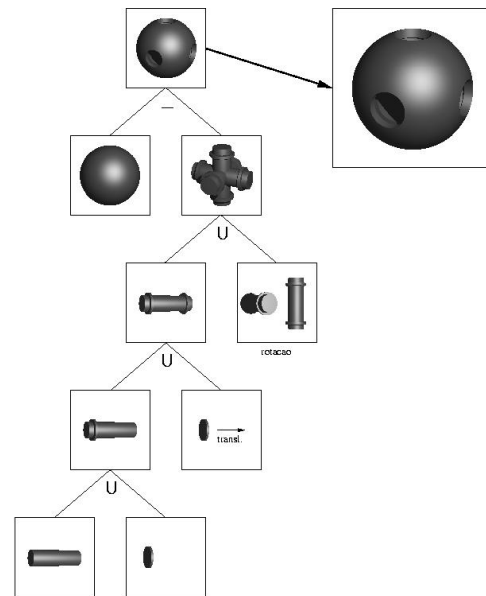


Abbildung 2.3: Beispiel der CSG [15]

Sweep-Representation

Eine Fläche wird durch Translation, Rotation oder Trajektion in die dritte Dimension extrahiert. Das Volumen, welches die Fläche entlang eines Pfades durchfährt, definiert den Sweep-Körper [5]. Als Definition der Fläche und des Pfades kann jede der hier genannten Geometriedatenstrukturen genutzt werden.

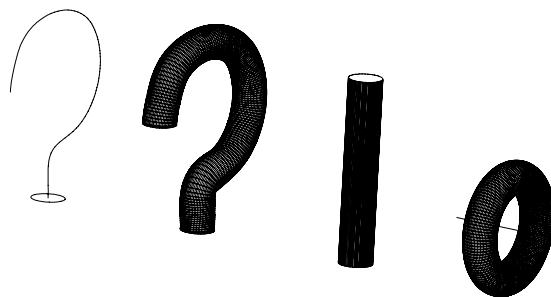


Abbildung 2.4: Sweep-Modell: Trajektion, Translation, Rotation

Octree

Ein Octree ist eine in der Informatik angewandte Baumstruktur mit einer Wurzel. Jeder Knoten hat entweder acht oder keinen Nachfolger. Sie können als Weiterentwicklung von Binärbäumen (1D) oder Quadtrees (2D) angesehen werden. Mit seiner Hilfe können alle erdenklichen Strukturen dargestellt werden. Nachteil der Verwendung ist, dass mit zunehmender Genauigkeitserwartung auch Speicherbedarf und Rechenleistung steigen (vgl. Geometriefehler in der FEM).

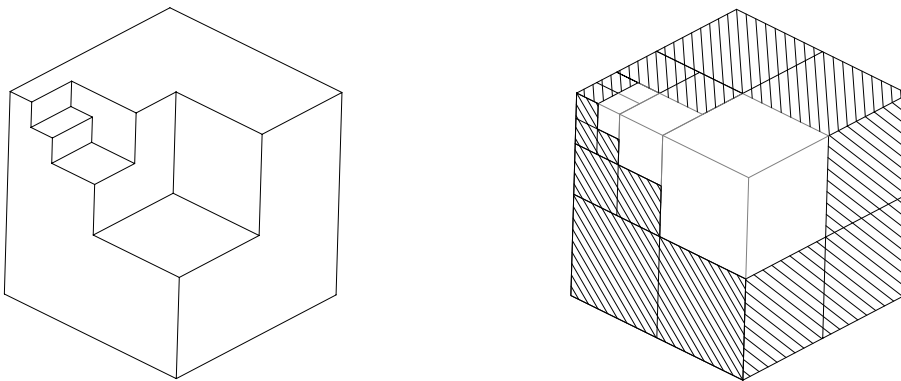


Abbildung 2.5: Darzustellender Körper (links) und Octreezerlegung des umgebenden Raumes (rechts)

Der den darzustellenden Körper umgebende Raum wird in 8 (i.d.R. gleichgroße) Würfel zerlegt. Die entstehenden Würfel werden darauf untersucht, ob sie voll, teilweise oder überhaupt nicht besetzt sind. Es wird jeweils nur der Pfad von teilweise besetzten Würfeln mit der gleichen Strategie weiterverfolgt. Dadurch ergibt sich eine Baumstruktur, der Oktalbaum (siehe Abb. 2.6) [7].

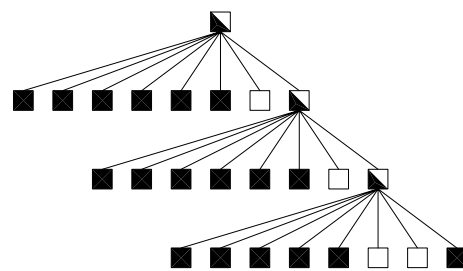


Abbildung 2.6: Oktalbaum

Boundary Representation (B-Rep)

In der BRep wird ein Körper über seine Begrenzungen beschrieben. Die umhüllenden Primitive sind dabei Knotenpunkte, Kanten und Flächen. Jedem dieser Elemente werden Eigenschaften zugeordnet, welche es geometrisch beschreiben. So kann man beispielsweise einer Kante einen linearen, aber auch einen parametrischen Verlauf höherer Ordnung zuweisen. Ähnliche Möglichkeiten bieten sich bei den Flächen, welche in Form von B-Splines o.ä. komplexe Verläufe darstellen können. (→ Geometrie)

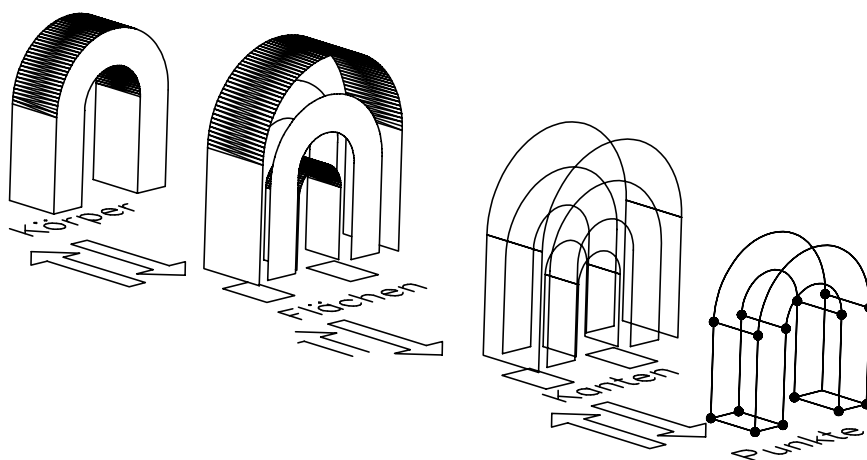


Abbildung 2.7: Grundprinzip der Boundary Representation Data Structure

Bild 2.7 zeigt den Zusammenhang der geometrischen Elemente (→ Topologie):

- Das Modell besitzt eine bestimmte Anzahl an Knotenpunkten. Jeder Knotenpunkt (VERTEX) wird in seiner Lage durch Koordinaten bestimmt.
- Jede Kante (EDGE) besitzt zwei Knotenpunkte. Die Form der Kante kann durch Parameterbeschreibung beliebig gekrümmt dargestellt werden.
- Jede Fläche (FACE) besitzt mindestens einen geschlossenen Ring von Kanten. Durch Stützpunkte oder Flächenfunktionen kann jede Form beschrieben werden.

- Die Oberfläche (SHELL) eines Körpers (REGION) wird beschrieben durch mindestens einen geschlossenen Ring von Flächen. Durch die Orientierung der Kanten und Flächen erreicht man die Zuordnung von äußerer und innerer Umgebung.
- Das Modell (MODEL) wird durch mindestens eine Oberfläche beschrieben.

Die Speicherstruktur der BRep geht von einer klaren Trennung der Topologie von der Geometrie aus. Änderungen in den Koordinaten eines Knotenpunkts haben somit keine Auswirkungen auf seine Verknüpfungen zu Nachbarpunkten. Werden Verknüpfungen zwischen Knotenpunkten verändert, behalten diese ihre geometrischen Informationen. Allein die Topologie ändert sich.

Es existieren verschiedene Verfahren, einen Körper mittels BRep zu beschreiben. Die für die Darstellung von manifolds wohl gebräuchlichste ist die **Winged-Edge Data Structure**. Die Theorie dafür stammt von Baumgart [1] und geht von Polyedern der Mannigfaltigkeit 2 aus. Eine Fläche wird von einem geschlossenen Ring an Kanten bestimmt. Die Richtung dieses Ringes, mit oder gegen den Uhrzeigersinn, beschreibt die Orientierung der Fläche. Demzufolge macht es einen Unterschied, ob eine Kante von $\mathbf{X} \rightarrow$ nach \mathbf{Y} oder von $\mathbf{Y} \rightarrow$ nach \mathbf{X} läuft.

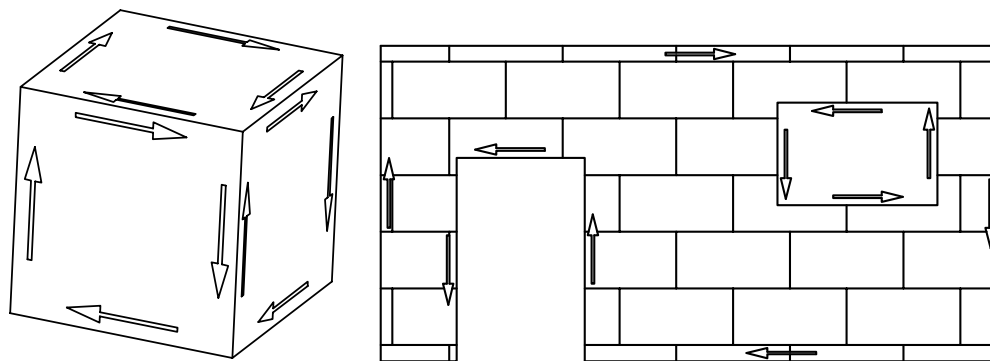


Abbildung 2.8: Prinzip der Winged-Edge Data Structure

Stoßen zwei Flächen an einer Kante zusammen, sind ihre Ringe gegensätzlich orientiert. Man erkennt den linken und rechten Flügel der Kante (winged-edge). In ersten BRep-Modellen war es nicht möglich, Löcher innerhalb einer Fläche darzustellen [7]. Mit

Hilfe von inneren und äußeren Schleifen (LOOPS) wurde dieses möglich (siehe Abb. 2.8). Innere und äußere Schleifen sind gegenläufig.

Es bleibt festzuhalten, dass sich mit Hilfe der *Winged-Edge Data Structure* nur Körper der Mannigfaltigkeit 2 darstellen lassen. Anwendungen, wie z.B. die Darstellung eines Quaders, finden sich u.a. in [5]. Für die Implementierung in *SING* sollen jedoch keine Einschränkungen hinsichtlich der Darstellbarkeit von Geometrien gemacht werden. Durch Beschränkungen auf Polyeder und oder manifold-Objekte erhält man a priori ungewollte Geometriefehler. Es muss demnach eine Datenstruktur verwendet werden, welche die Darstellung von krummlinig berandeten nonmanifold Objekten ermöglicht. Eine solche Datenstruktur wird von Weiler in [14] vorgestellt: Die *Radial-Edge Data Structure*. Sie wird im folgenden Kapitel (Kap. 3) näher erläutert.

Hybride Modelle

Jede der vorgestellten Geometriemodelltypen hat seine Vorzüge, aber jeder hat auch seine Nachteile. So ist eine Modellierung, ohne die Vorzüge der BRep zu nutzen, kaum vorstellbar. Die schwerfälligen Primitive der CSG machen es schwer, komplizierte Formen darzustellen. Es muss sich aber feststellen lassen, dass es für den Nutzer sehr aufwendig ist, jedes geometrische Objekt und jede topologische Verbindung zu definieren. Auch ist diese Art der Eingabe nur schwer frei von Fehlern zu halten.

Hybride Modelle verbinden verschiedene Geometriemodelltypen miteinander. Zum Beispiel werden in einem hybriden Modell Grundformen in CSG- oder Sweepmodellen angelegt. Die Endverarbeitung und Speicherung erfolgt dann aber in den flexibleren BRep Strukturen. Dadurch sinkt die Fehleranfälligkeit und steigt die Anwenderfreundlichkeit. Moderne Programmsysteme speichern CSG- und BRep-Daten parallel. Das Erstellen neuer CSG-Primitive mittels BRep-Beschreibung ist ebenso möglich wie die Speicherung der Entstehungsgeschichte.

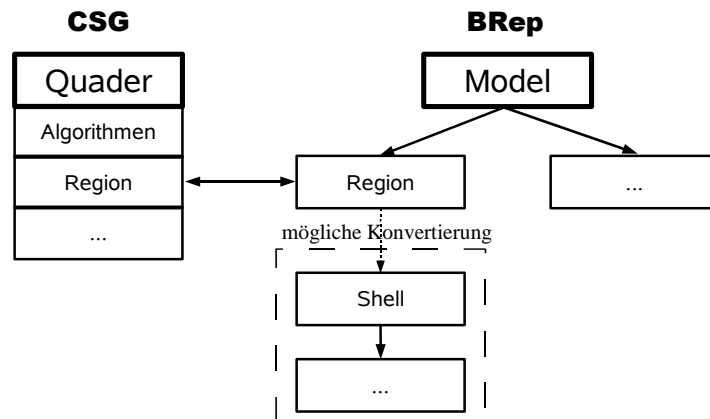


Abbildung 2.9: Beispiel für die Verknüpfung von CSG und BRep in einer hybriden Datenstruktur

Es ergeben sich u.a. folgende Vorteile:

- Abfragen können mit den Informationen über das CSG-Primitiv schneller und einfacher gestaltet werden.
- Die Eingabe von einfachen geometrischen Objekten wird erleichtert. Die Wahrscheinlichkeit von Eingabefehlern sinkt.
- Es können über die BRep jederzeit nicht-primitive Geometrien dargestellt werden.
- Der Speicherbedarf für CSG-Primitive ist geringer als eine äquivalente BRep-Beschreibung.

Kapitel 3

Radial-Edge Data Structure (RED)

Die Darstellung von nonmanifold-Objekten ist mit Hilfe der Winged-Edge Data Structure nicht möglich. Aus diesem Grund wurden verschiedene Verfahren entwickelt um auch dieses zu ermöglichen. Diese Arbeit wird sich damit beschäftigen, eine RED-Datenstruktur in das Programmsystem `Slang` zu implementieren.

Die RED (*Radial-Edge Data Structure*) wurde von Weiler [14] entwickelt und veröffentlicht. Mit ihrer Hilfe ist es möglich, krummlinig berandete nonmanifolds abzubilden.

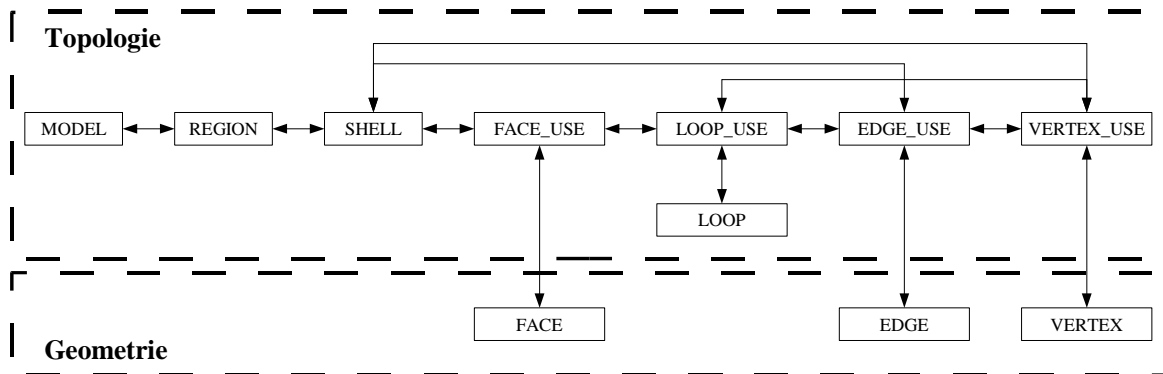


Abbildung 3.1: Trennung von Topologie und Geometrie

Um dies zu erreichen, trennte Weiler die Topologie noch strenger von der Geometrie als dies in vorhergegangenen Arbeiten [1, 7] geschah (siehe Abb. 3.1). Er trennte die geometrischen Objekte Knotenpunkt, Kante und Fläche von ihren topologischen Verknüpfungen. Die Topologieelemente erhielten den Zusatz *-use.

3.1 Geometrische Beschreibungen

In Boundary Modellen existieren drei Typen geometrischer Elemente. Diese sind Knotenpunkte (VERTICES), Kanten (EDGES) und Facetten (Flächen, FACES). Jedes Objekt kann verschieden definiert werden. Durch ihre topologische Verknüpfung entsteht das Modell. Dieser Abschnitt soll dazu dienen die verschiedenen Definitionen der Geometrieobjekte zu erläutern:

VERTEX

Ein Knotenpunkt oder auch VERTEX ist gleichzusetzen mit einem Punkt im dreidimensionalen Raum. Der Punkt ist das kleinste Objekt in der Geometrie. In der Regel hat der Knotenpunkt lediglich Informationen über seine Lage. Das heißt seine geometrische Beschreibung beschränkt sich auf die Koordinaten in 3D.

EDGE

Die Kante, im Weiteren auch als EDGE bezeichnet, ist eine ungeordnete Menge von zwei VERTICES. In der Regel geht man davon aus, dass diese ungleich sind. Im Spezialfall kann eine EDGE jedoch aus zwei Knotenpunkten bestehen, die sich gleich sind. Diese Sonderfälle sind geschlossene Kurven z.B. geschlossene Ellipsen- bzw. Kreisbögen. Die Kante kann, im Gegensatz zum VERTEX, verschieden definiert werden. Der einfachste Fall ist dabei die Linie. Sie stellt den linearen Verlauf zwischen den beiden Punkten dar. Möglich ist aber auch eine andere Charakteristik zu wählen. Diese erfordert neben den beiden Knotenpunkten weitere Informationen. Eine Kante definiert sich demnach über ihre beiden VERTICES und der Information über den Verlauf zwischen diesen. Typische Darstellungsformen von Kanten sind:

- gerade Linien
- Ellipsen-/ Kreisbogen
- Bezier- bzw. Spline-Kurven
- explizite oder implizite Funktionsbeschreibungen

FACE

Flächen (FACEs, Facetten) werden durch mindestens einen geschlossenen äußeren Ring (LOOP, Schleife) von Kanten gebildet. Der Anfangspunkt der ersten EDGE muss der Endpunkt der letzten EDGE sein. Endpunkte einer EDGE sind gleich mit den Anfangspunkten ihres Nachfolgers. Folglich werden, um einen Ring zu schließen, die zugehörigen EDGEs gerichtet.

Mit inneren Schleifen werden Löcher innerhalb der Fläche ausgedrückt (siehe Abs. 2.3). Ihre Orientierung erfolgt entgegengesetzt zur Richtung der äußeren LOOP.

Flächen können wie Kanten verschiedene Geometriebeschreibungen ausdrücken. Sie enthalten die Information über den Kurvenverlauf zwischen den EDGEs. Zu beachten ist, dass die Definition der Fläche in jedem Fall auch die der zugehörigen Kanten erfüllen muss. Typische Darstellungsformen von Flächen sind:

- Ebenen,
- Ansatzfunktionen (mit z.B. den verbundenen VERTICES als Stützstellen),
- Bezier- bzw. Spline-Kurven,
- NURBS (Non-Uniform Rational B-Splines),
- explizite oder implizite Funktionsbeschreibungen.

Ein Volumen (REGION) wird dann definiert, wenn eine Menge von Flächen in sich geschlossen ist. Das heißt, jede Kante der Flächen ist mit einer Kante der in der Menge enthaltenen Flächen verbunden. Gleiches gilt folglich auch für die VERTICES. Schließt eine Fläche sich selbst (Blase), dann ist die Definition der LOOP über einen geschlossenen Ring von Kanten nicht sinnvoll. In diesem Fall erhält die LOOP als Referenzpunkt lediglich einen VERTEX.

3.2 Topologische Beschreibungen

Die Topologie stellt die Verbindung zwischen den geometrischen Objekten dar. Damit übernimmt sie die Organisation der Datenstruktur. Sie stellt diese Struktur erst auf. Für gewöhnlich verbindet sie die Elemente höherer mit denen niederer Dimension und umgekehrt.

Ein Modell ist dann eindeutig deklariert, wenn die Beziehungen aller geometrischen Elemente zueinander eindeutig sind. Das heißt zum Beispiel, ein VERTEX kennt die mit ihm verbundenen Elemente und die durch diese Elemente mit ihm verbundenen VERTICES. Für die RED ergeben sich danach folgende Abhängigkeiten:

	V(ERTEX)	E(DGE)	L(OOP)	F(ACE)	S(HELL)	R(EGION)
V	$V\{V\}$	$V\{E\}$	$V\{L\}$	$V\{F\}$	$V\{S\}$	$V\{R\}$
E	$E\{V\}^2$	$E\{\langle E \rangle\}^2$	$E\{\langle L \rangle\}$	$E\{\langle F \rangle\}$	$E\{\langle S \rangle\}$	$E\{\langle R \rangle\}$
L	$L\{\langle V \rangle\}^2$	$L\{E\}^2$	$L\{\langle\langle L \rangle\rangle\}^2$	$L\{F\}^1$	$L\{S\}^2$	$L\{R\}^2$
F	$F\{\{\langle V \rangle\}\}^2$	$F\{\{\langle E \rangle\}\}^2$	$F\{L\}$	$F\{\{\langle\langle F \rangle\rangle\}\}^2$	$F\{S\}^2$	$F\{R\}^2$
S	$S\{V\}$	$S\{E\}$	$S\{L\}$	$S\{F\}$	$S\{S\}$	$S\{R\}^1$
R	$R\{V\}$	$R\{E\}$	$R\{L\}$	$R\{F\}$	$R\{S\}$	$R\{R\}$

Abbildung 3.2: topologische Beziehungen der non-manifold Elemente [14]

Hauptdiagonale

z.B. ein VERTEX kennt alle mit ihm durch Kanten, Flächen usw. verbundenen Knotenpunkte

Erste Nebendiagonale

z.B. eine Kante kennt ihre VERTICES und die Flächen zu der sie gehört

Weitere Elemente

Zumindest während der Erstellung eines Körpers treten wireframe-Strukturen¹ auf. Die Knotenpunkte, Kanten oder auch Flächen werden dabei einer SHELL zugeordnet. Um ihre Abhängigkeiten erfassen zu können, müssen Elemente berücksichtigt werden, welche nicht auf der ersten Nebendiagonale liegen.

Zudem wird eine Verbindung zwischen VERTEX und LOOP benötigt. Dieser Fall tritt, wie bereits beschrieben, bei sich selbst schließenden Flächen auf.

¹Drahtgittermodell; In einer wireframe-Struktur existieren lediglich Knotenpunkte und Kanten.

Alle diese Abhängigkeiten müssen in einer Datenstruktur definiert werden bzw. abrufbar sein. Es ist nicht notwendig, jedes Element der Matrix in Abbildung 3.2 abzuspeichern. Man beschränkt sich auf Verbindungen zu Objekten, welche eine Ordnung höher und eine Ordnung tiefer liegen. Eine Abfrage über zwei Hierarchiestufen muss demzufolge über einen Zwischenschritt erfolgen. Die Elemente der Hauptdiagonale werden ebenfalls nicht mit abgespeichert. Auch in diesem Fall ist der Zwischenschritt notwendig. Die doppelte Verknüpfung geschieht, um Auf- und Abwärtsabfragen vornehmen zu können. Der Speicherbedarf hält sich mit dieser Lösung in Grenzen.

Der Einsatz von mehr Speicherkapazität bedeutet nicht zwingend einen geringeren Rechenaufwand. Denn dieser erhöht sich durch die erhöhten Kosten, welche das Erstellen und Löschen der Struktur mit sich bringt. Mit zusätzlichen Verknüpfungen sollte vorsichtig umgegangen werden. Sie sollten nur im Zusammenhang mit umfangreichen Testbeispielen und Benchmarktests erfolgen.

VERTEX_USE

VERTEX_USEs sind die topologischen Elemente des Knotenpunkts. Sie erzeugen seine Verknüpfung mit hierarchisch höherliegenden Objekten. Ein VERTEX kann topologisch mit einer Kante, einer LOOP oder einer SHELL verbunden sein.

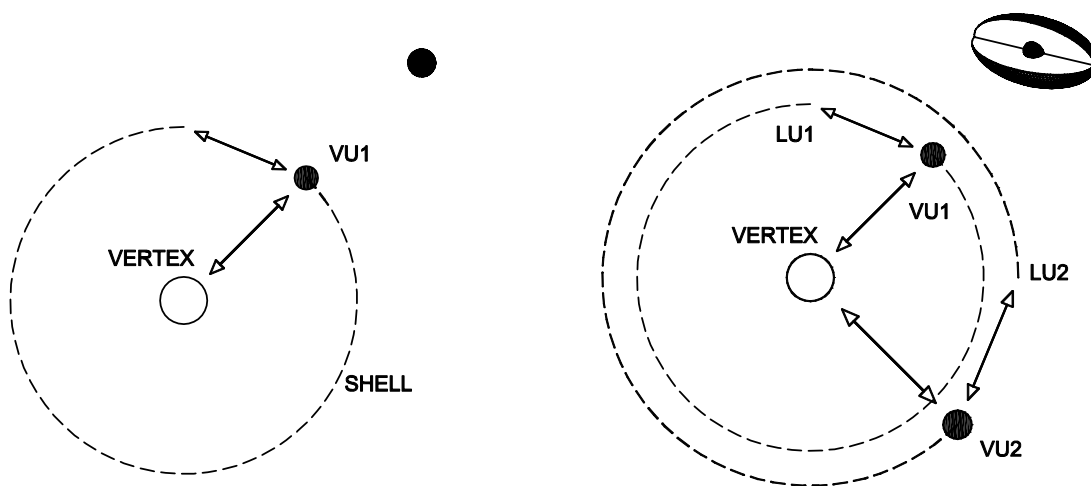


Abbildung 3.3: Verbindung VERTEX_USE - SHELL

Abbildung 3.4: Verbindung VERTEX_USE - LOOP_USE

Liegt ein VERTEX frei im Raum und hat keine Verbindung zu anderen geometrischen Objekten, wird ihm eine Oberfläche (SHELL) zugewiesen. Im Fall des Anschlusses an eine Kante wird der VERTEX von deren topologischem Element, der EDGE_USE, genutzt. Soll im MODEL eine in sich geschlossene Fläche dargestellt werden, dient der Knotenpunkt als Referenz der, die Blase beschreibenden, LOOP.

Er wird dann mit ihrem topologischen Element, der LOOP_USE verbunden. Die Abbildung zeigt den Anschluß von 2 Kanten an einen VERTEX. Der VERTEX wird im topologischen Sinn der RED zweimal genutzt. Um diese Verbindung realisieren zu können, erhält er genau diese Anzahl an VERTEX_USEs. Diese stellen die Verbindung zur EDGE_USE der verbundenen EDGE bereit.

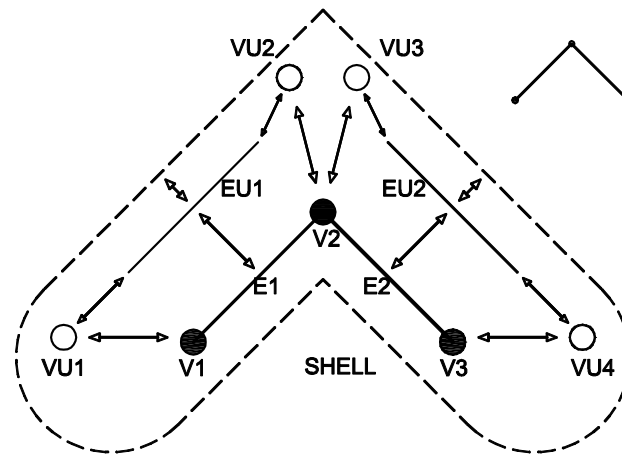


Abbildung 3.5: Verbindung von zwei Kanten an einem Punkt

EDGE_USE

Das topologische Element der EDGE ist die EDGE_USE. Jede EDGE_USE hat zwei Verknüpfungen zu den VERTEX_USEs der verbundenen Knotenpunkte. Die hierarchisch höherliegenden Elemente, welche mit ihr verbunden werden, können LOOP_USE oder SHELL sein.

Ein geschlossener Ring von Kanten, welcher eine Fläche nach innen oder nach außen begrenzt, ergibt eine LOOP. Ihre LOOP_USEs werden mit den EDGE_USEs der den Ring bildenden Kanten verbunden. Ist die Kante Teil einer wireframe-Struktur, wird ihre EDGE_USE einer SHELL zugeordnet. Die Anzahl der topologischen Verbindungen einer Kante und die der zugehörigen EDGE_USEs muss zu jedem Zeitpunkt identisch sein. Eine „freie“ EDGE_USE darf nicht existieren. Gleiches gilt für alle anderen Topologieelemente eines Modells.

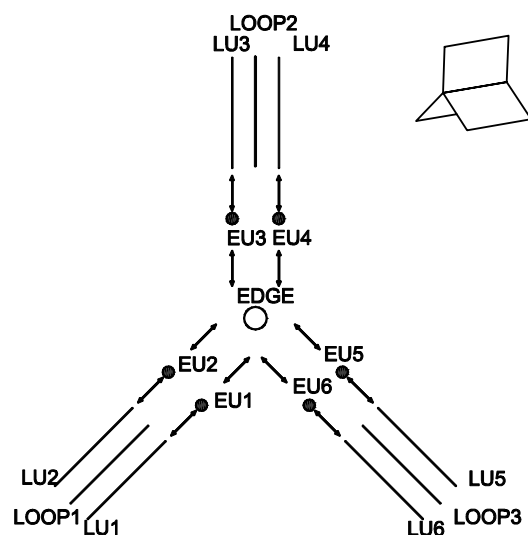


Abbildung 3.6: Verbindung von 3 Flächen an einer Kante nach Weiler [14]

Anmerkung: Die ursprüngliche Abbildung (3.6) von Weiler zeigt die Verbindung von EDGE_USEs zu FACE_USEs. Diese Art der Verbindung kann nach dem Konzept der RED aber nicht auftreten (siehe auch Abb. 3.1). Es wird die mögliche Verbindung von LOOP_USE zu EDGE_USE dargestellt.

LOOP/ LOOP_USE

Eine LOOP kann auf zwei verschiedene Arten definiert werden, zum Einen als Ring von Kanten, zum Anderen als Beschreibung einer geschlossenen Fläche.

Die LOOP_USE hat in der Hierarchie aufwärts eine Verbindung zu einer FACE_USE. Abwärts ergeben sich 2 Möglichkeiten:

1. Die LOOP wird durch einen Ring von Kanten beschrieben. In diesem Fall definiert sich die LOOP_USE durch eine geordnete Menge EDGE_USEs. Die EDGE_USEs einer LOOP_USE sind so gerichtet, dass der Endknoten der einen EDGE_USE dem Anfangsknoten seines Nachfolgers entspricht. Die LOOP_USE muss geschlossen sein, d. h. der Endknoten der letzten EDGE_USE ist gleich dem Anfangsknoten der Ersten. Die sich gegenüberliegenden LOOP_USEs sind entgegengesetzt orientiert.

2. Die LOOP ist in sich geschlossen (siehe Abb. 3.4.) Die abwärtige Verknüpfung der LOOP_USE erfolgt dann zu einer VERTEX_USE des Referenzknotens der Blase.

Die Abbildung zeigt die Verbindungen von 3 LOOPS an einer Kante, ein typisches Beispiel für nonmanifold Geometrien. Die 3 verbundenen LOOPS haben je zwei LOOP_USEs. Daraus ergibt sich, dass die EDGE 6 mal genutzt wird.

Die Orientierung der 2 EDGE_USEs einer Kante, welche mit der selben LOOP verbunden sind, muss gegenläufig sein. Das heißt die erste VERTEX_USE einer EDGE_USE zeigt auf den gleichen VERTEX wie die zweite VERTEX_USE seines Pendants.

FACE_USE

Grundsätzlich besitzt jede Fläche eine Ober- und eine Unterseite. Jede dieser Seiten kann ein Volumen nach innen bzw. nach außen begrenzen. Sie können demnach topologisch genutzt werden. Aus diesem Grund besitzt eine Fläche zwei FACE_USEs. Eine FACE_USE gehört zu einer Oberfläche, sie ist also topologisch aufwärts mit einer SHELL verbunden. Die abwärtsige Verbindung einer FACE_USE besteht zu einer LOOP_USE, deren LOOP die Fläche nach außen begrenzt und einer Menge LOOP_USEs, deren LOOPS die Fläche in ihrem Inneren begrenzt. Die inneren LOOPS bilden demzufolge die Löcher in einer FACE.

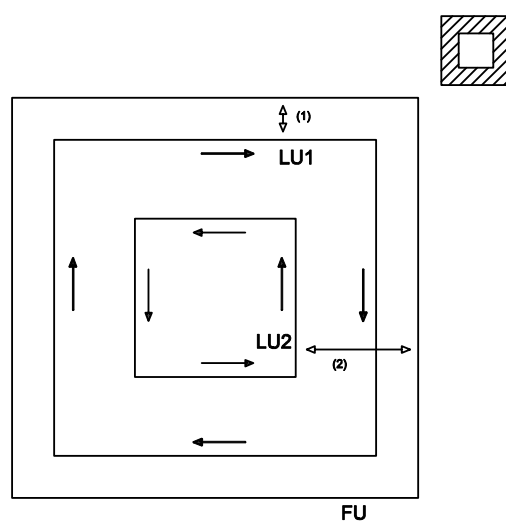


Abbildung 3.7: Beschreibung eines Loches in einer Fläche

SHELL

Die SHELL an sich hat keine eigene Geometriebeschreibung. Sie fungiert aus diesem Grund allein als topologisches Element. Die SHELL ist aufwärts an eine REGION und abwärts an Flächen, Kanten und oder einen Knotenpunkt gebunden. Besteht die Verbindung zu einer VERTEX_USE, so beschreibt die SHELL die Oberfläche des VERTEX. Offensichtlich ist diese Art der Verbindung so nicht notwendig. Da Knotenpunkte jedoch meist nur Zwischenstationen beim Erstellen eines Gesamtmodells sind, empfiehlt sich diese Verknüpfung im Hinblick auf die weitere Nutzung der SHELL (siehe 3.3). Beschreibt die SHELL die Oberfläche einer wireframe-Struktur, dann wird sie mit

den EDGE_USEs der zugehörigen Kanten verbunden. Sollen auch Flächen zur SHELL gehören, welche kein Volumen einschließen, dann werden beide FACE_USEs der Fläche der SHELL zugeordnet.

Bilden die Flächen einer SHELL ein abgeschlossenes Volumen, dann werden die inneren FACE_USEs der Flächen einer INNER-SHELL zugeordnet. Die äußeren FACE_USEs bilden eine OUTER-SHELL. Das geschlossene innere Volumen wird zu einer REGION, die äußere SHELL liegt im umgebenden Volumen.

Das äußerste Volumen ist die OUTREGION. Sie wird mit dem Anlegen des Modells erstellt. Somit umgibt sie alle Körper, wireframe-Strukturen und Knotenpunkte. SHELLs, welche wireframe-Strukturen oder Knotenpunkte beschreiben, liegen in der OUTREGION. Diese wird üblicherweise auch als REGION 0 („NULL“) bezeichnet.

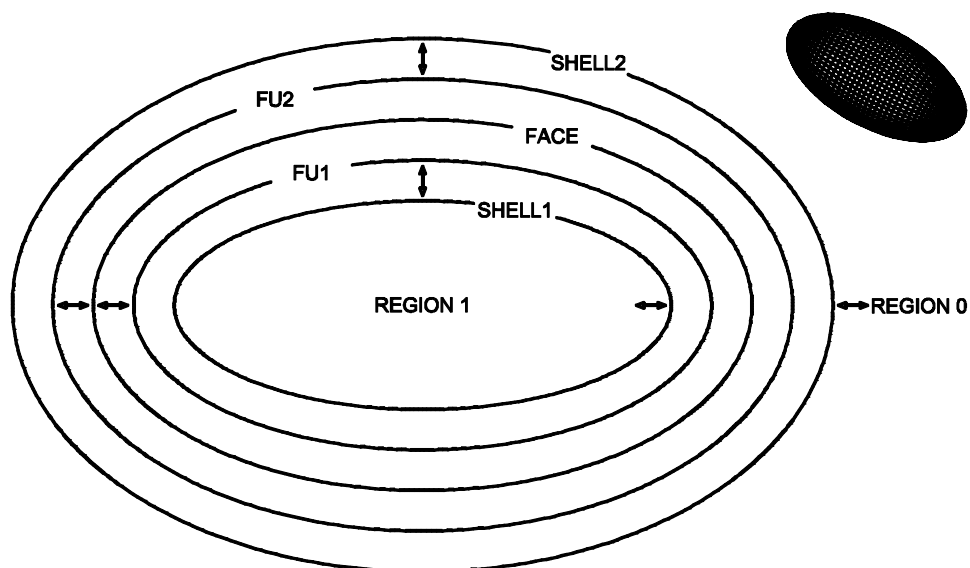


Abbildung 3.8: Topologische Verbindungen (aufwärts) einer in sich geschlossenen Fläche (vgl. Abb. 3.4 und 3.7)

Durch diese Art der Trennung von Topologie und Geometrie wird es möglich, nonmanifold-Geometrien darzustellen.

3.3 Operatoren

Für Polyeder der Mannigfaltigkeit 2 gilt die Euler-Poincaré Gleichung 2.2.1. Sie muss demzufolge auch für jeden Konstruktionsschritt gelten. Wird ein Element hinzugefügt oder gelöscht, dann muss sich auch die Anzahl der anderen Elemente ändern. Außerdem müssen topologische Verbindungen gelöscht oder erstellt werden.

Auf dieser Basis entwickelte Baumgart [1] die Euler-Operatoren. Sie garantieren die Einhaltung der Gleichung. Die Operatoren bilden infolgedessen die kleinsten Elemente aller Operationen auf der Datenstruktur. Jede Manipulation des Modells kann und muss mit ihrer Hilfe erfolgen. Dabei werden sämtliche topologischen Verknüpfungen gesetzt oder gelöst. Damit wird erreicht, dass die Datenstruktur zu jedem Zeitpunkt eindeutig ist. Topologisch unmögliche Verknüpfungen werden vermieden. Die Fehlerwahrscheinlichkeit wird verringert.

In nonmanifold-Strukturen gilt die Polyeder-Gleichung nicht mehr. Dem Konzept der kleinsten Operationen folgend, passte Weiler die Operatoren auf die RED an. Im Folgenden sollen die Operatoren vorgestellt werden, welche die Organisation der implementierten Datenstruktur übernehmen.

Die aufgeführten Operatoren unterscheiden sich von den in [14] genannten. Das hat zwei Gründe:

1. Operatoren wurden zusammengefasst. Damit wurde erreicht, dass das Programm nicht mehrfach die gleichen Entscheidungen treffen muss.
2. Die Operatoren wurden so angepasst, dass die Übergabeparameter der daraus folgenden Methoden der möglichen Eingabe der `SLANG`-Kommandos entspricht. Damit wird der programmierte Code leichter nachvollziehbar, und man erspart sich überflüssige Abfragen.

Die Funktionalität ändert sich dahingegen nicht. Es bleibt auch mit diesen Operatoren möglich, alle Arten von Geometrien zu erschaffen bzw. zu löschen. Das Modell bleibt zu jedem Zeitpunkt konsistent.

Make-Operatoren

Die Make-Operatoren dienen dem Erstellen des Modells. Sie beginnen mit „m_“ danach folgen die vom Operator anzulegenden Elemente: „r“-REGION, „s“-SHELL, „f“-FACE, „l“-LOOP, „e“-EDGE und „v“-VERTEX. Beim Erstellen einer Struktur kann ein Element überflüssig werden, dann wird dieses entfernt. Der Operator erhält vor dem zu löschenden Objekt ein „k“ für kill.

- **m_r**

Die zu konstruierende Geometrie benötigt einen Raum, in welchem sie beschrieben werden kann. Dazu wird mit Anlegen des MODELS die OUTREGION erstellt. Alle Elemente der Geometrie befinden sich mittelbar oder unmittelbar innerhalb der REGION 0.

- **m_sv**

Wird ein VERTEX erstellt, so hat er zuerst keinerlei Verbindung zu anderen geometrischen Objekten. Um ihn dennoch innerhalb des MODELS topologisch zu erfassen, erhält er eine SHELL. Diese SHELL ist mit der OUTREGION verbunden.

- **m_eks**

Die EDGE hat topologische Verbindungen zu zwei VERTICES. In einer wireframe-Struktur hat sie jedoch lediglich eine Verbindung zu einer SHELL. Die überflüssige SHELL wird gelöscht. Die Verbindungen von EDGE zu SHELL und VERTICES zu EDGE werden gesetzt. Die SHELL bleibt Teil der OUTREGION.

- **m_fl**

Ein Ring von Kanten, welcher eine Fläche einschließt, wird als neue LOOP kreiert. Die EDGE_USEs der Kanten werden gerichtet und an eine LOOP_USE gehängt. Die LOOP erstellt eine zweite LOOP_USE, welche zur Ersten gegenläufig ist. Jede der zwei LOOP_USEs wird mit einer FACE_USE verbunden. Beide FACE_USEs werden an die SHELL der EDGE_USEs gekoppelt.

- **m_l**

Soll eine INNER-LOOP erschaffen werden, verbindet der Operator die zugehörigen EDGES nach vorab beschriebenen Schema und erstellt zwei LOOP_USEs. Die LOOP_USEs werden an die zugehörigen FACE_USEs angeschlossen.

- **m_rs**

Begrenzen eine oder mehrere Flächen ein Volumen, so werden die inneren FACE_USEs zu einer neuen inneren Oberfläche (SHELL) zusammengezogen. Diese Oberfläche wird mit einem neuen Volumen (REGION) verbunden. Die äußeren FACE_USEs bleiben mit der an die OUTREGION gekoppelten SHELL verbunden.

- **m_firs**

Dieser Operator kombiniert die Operatoren `m_fl` und `m_rs`. Durch das Erstellen einer neuen Fläche (wie `m_fl`) wird ein Volumen geschlossen. Das führt zum Anlegen einer neuen inneren SHELL und eines neuen Volumens (siehe `m_rs`). Die Anwendung dieses Operators empfiehlt sich speziell beim Anlegen von Flächen, welche in sich geschlossen sind.

Kill-Operatoren

Die Kill-Operatoren dienen dem Löschen einzelner Elemente des MODELS. Dabei zerstören sie alle topologischen Verknüpfungen und beseitigen höherliegende Elemente, welche sich durch das niedere, gelöschte Element definiert haben. Grundsätzlich können die Kill-Operatoren jedes geometrische Objekt löschen. Neben Knotenpunkten, Kanten und Flächen müssen aber auch Volumen und LOOPS beseitigt werden können.

- **k_v**

Der Operator **k_v** löscht einen VERTEX inclusive aller Objekte die mit ihm verbunden sind. Es werden dazu Suboperatoren eingeführt. Sie werden von **k_v** aufgerufen.

- **k_vs**
- **k_ve**
- **k_ves**
- **k_vl**
- **k_velfrs**

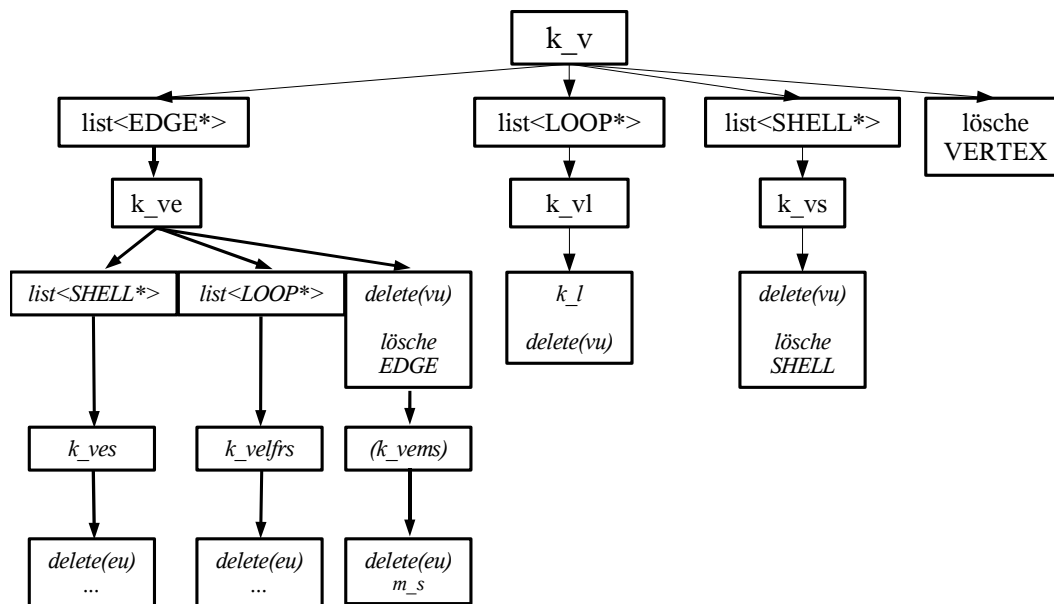


Abbildung 3.9: Entscheidungsbaum für **k_v**

- **k_e**

Eine Kante wird mit allen Elementen, welche durch sie mittelbar oder unmittelbar definiert werden, eliminiert. Dazu kommen folgende Unteroperatoren zum Einsatz:

- **k_e**
- **k_ems**
- **k_elfrs**

- **k_l**

Eine LOOP wird entfernt. Begrenzt die LOOP eine Fläche nach außen, werden auch diese und die durch sie beschriebenen Elemente gelöscht. Das geschieht durch:

- **k_lfrs**
- **k_lfms**

- **k_f**

Dieser Operator dient dem Löschen einer Fläche. Wird durch das Löschen der FACE eine geschlossene REGION geöffnet, so wird diese REGION gelöscht. Alle zur Fläche gehörenden LOOPS werden ebenfalls entfernt.

- **k_fl**
- **k_frs**

- **k_r**

Auflösen eines Volumens. Die untergeordneten SHELLs werden mit der OUTREGION verbunden.

- **k_m**

Löschen des gesamten MODELS.

Sonstige-Operatoren

Make- und Kill-Operatoren dienen allein dem Erzeugen und Löschen der Struktur. Mit ihnen kann man grundsätzlich alle Geometrien erzeugen. Doch nicht nur die Benutzerfreundlichkeit, sondern auch logische Vorgänge beim Erstellen eines MODELS erfordern weitere Werkzeuge. Diese sind z.B.:

- **Glue**
Zwei Elemente mit gleicher Beschreibung werden zu einer zusammengefasst. Dazu wird ein Objekt gelöscht und das verbleibende erhält jede seiner topologischen Verbindungen.
- **Split**
Ein Knotenpunkt, eine Kante oder eine Fläche wird in zwei identische Objekte geteilt. Die topologischen Verknüpfungen verbleiben zunächst an dem Grundelement. Das neu geschaffene Objekt wird in einem ersten Schritt mit dem Ursprungselement topologisch verknüpft.
- **Insert**
Es kann notwendig sein, ein Volumen in ein anderes einzufügen. Der Insert-Operator dient diesem Zweck. Dabei wird die äußere Oberfläche des inneren Volumens mit der REGION des umgebenden Körpers verbunden.
- **Move**
Ein Objekt wird im Raum verschoben oder gedreht. Dazu werden alle verbundenen Knotenpunkte bewegt. Wird durch die Bewegung ein hierarchisch höherliegendes Element verzerrt, muss seine geometrische Beschreibung geändert werden.

Kapitel 4

Boolesche Operationen

Teil dieser Arbeit ist eine Literaturrecherche zum Thema boolesche Operationen. Die RED ist ein boundary Modell. Dadurch ergeben sich spezielle Anforderungen. Kapitel 4 erläutert die Grundlagen der Verfahren im Hinblick auf die Nutzung in der RED. Die Beschreibung der Glue- und Split- Operatoren, als Basis der auftretenden Vorgänge in boundary Modells, soll als Einstieg für die Weiterbearbeitung der Implementierung gelten.

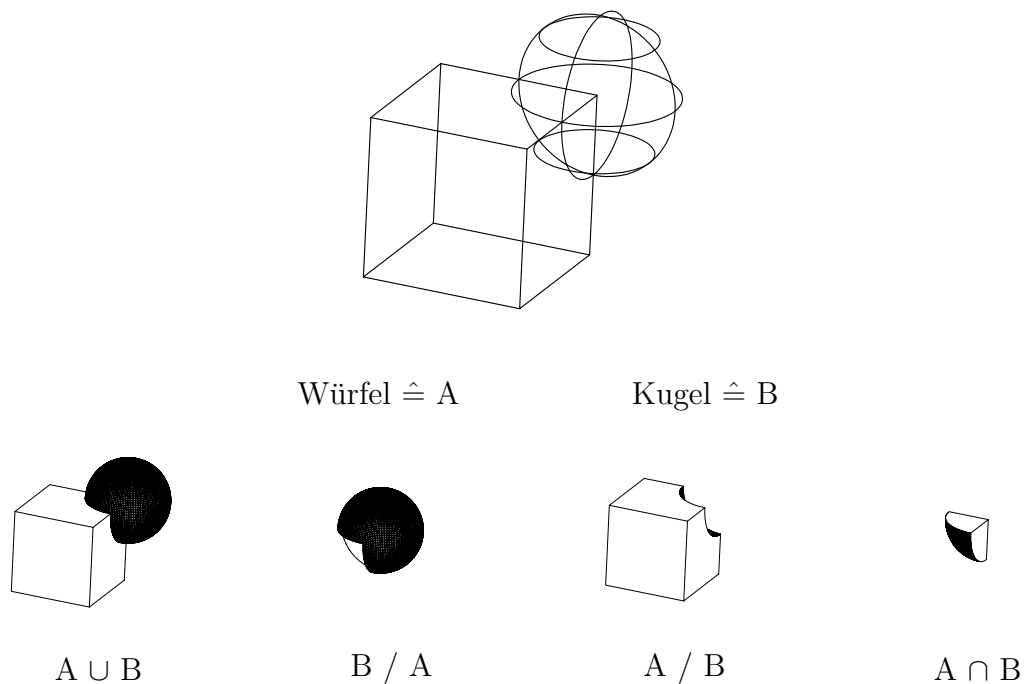


Abbildung 4.1: Boolesche Operationen

4.1 Operationen

In der RED ist es möglich, jedes erdenkliche Modell mit den in 3.3 Make-Operatoren anzulegen. Das Erstellen komplexer Geometrien stellt aber eine große Fehlerquelle dar. Aus diesem Grund werden auch in boundary Modellen Mengenoperationen eingeführt. Sie basieren auf den Methoden der booleschen Algebra und werden daher Boolesche Operationen genannt. Mit ihrer Hilfe können schwierige Beschreibungen durch eine geringe Anzahl dieser Operationen erzeugt werden. Grundsätzlich beschränkt man sich hier auf Verfahren, welche Körper zueinander in logische Verbindungen bringen. Eine Anwendung auf topologisch untergeordnete Elemente geschieht in der Regel nur als Teil einer Körperoperation.

Die Nutzung der Mengenoperationen in boundary Modellen ist nicht problemlos. Die Menge der möglichen geometrischen Definitionen ergibt, da jede Beschreibung mit jeder anderen getestet werden können muss, quadratisch die Menge der Abfragealgorithmen, welche entwickelt und implementiert werden müssen. Diese Algorithmen enthalten, aufgrund der komplexen Geometriedefinitionen, eine Unmenge an Fallunterscheidungen und diffizilen Teilalgorithmen. Die Robustheit eines Modells ist dadurch nur schwer zu garantieren [7]. Durch Schnittalgorithmen hervorgerufene numerische Fehler stellen ein weiteres Problem für die Anwendung der booleschen Operationen auf die RED dar.

Die Grundlage aller booleschen Anwendungen bilden drei Operationen. Alle weiteren Verfahren können durch sie erklärt werden. Beispiele für ihren Einsatz sind in Abbildung 4.1 zu sehen. Die Operationen sind Vereinigung \cup , Intersection \cap und Differenz $/$.

- Vereinigung \cup

Die Körper werden miteinander verschmolzen. Eines der Ausgangsvolumen wird dadurch überflüssig. Gleiches gilt für eine der inneren Oberflächen. Schneiden sich die Körper, so muss ein Volumen an der Oberfläche des anderen abgeschnitten und die Flächen mit INNER-LOOPS „geöffnet“ werden. Die inneren FACE_USES werden einer SHELL zugeordnet. Die andere innere SHELL wird, wie auch ihre verbundene REGION, gelöscht.

- Intersection \cap

Als Ergebnis der Intersection steht das gemeinsame Volumen beider Körper. Auch hier ist die Information über die Lage der beiden Körper ausschlaggebend für den Algorithmus. Beim Schnitt der beiden Körper verbleiben die Teilflächen, welche im anderen Volumen liegen. Sie werden zu einem neuen Volumen zusammengefasst.

- Differenz /

Die Differenz eines Körpers A von einem Körper B ergibt das Volumen von B, welches nicht gleichzeitig Teil von A ist. Der geschaffene Körper (B/A) erhält zu seiner Oberfläche die Teilflächen von A, welche innerhalb von B liegen. Im Gegensatz zu den beiden anderen Operationen ist die Differenz nicht assoziativ bzw. kommutativ.

Die Ergebnisse boolescher Operationen sollen auch wieder Körper sein. Um dass sicherzustellen werden die Operationen regularisiert. In der Regel werden dazu die Normalenvektoren der Oberflächen benötigt. Da die Elemente in boundary Modellen konvexe und konkave Beschreibungen in einem haben können, gestalten sich die Algorithmen, wie sie z.B. für die CSG angewandt werden, problematisch. Die Probleme mit numerischen Fehlern sind bei booleschen Operationen nur über Toleranzgrenzen effizient zu behandeln.

Durch die Kombination mehrerer boolescher Operationen kann eine komplexe Geometrie einfach erstellt werden. Durch die Nachteile, welche die RED im Bezug auf die Mengenoperationen mit sich bringt, muss zwischen Mehraufwand für das Entwickeln von Algorithmen oder Mehraufwand beim Konstruieren entschieden werden. Ein Kompromiss ist, die Operationen in einem Hybridsystem nur für die CSG-Körper vorzunehmen.

Man erkennt, dass die Lage zweier Körper maßgeblich den Algorithmus der booleschen Operation bestimmt. Der nächste Abschnitt wird verschiedene Lagetypen und die daraus folgenden Aktionen nennen und erläutern.

4.2 Lageinformation

Zwei Körper können zueinander unterschiedlich liegen. Die Lage der Körper gibt dabei an, inwieweit sie sich schneiden oder berühren. Diese Information bestimmt grundlegend den Algorithmus der booleschen Operationen. Sie wird dazu in vier Klassen eingeteilt. Ein Körper kann komplett außerhalb oder innerhalb eines anderen Volumens liegen, es berühren oder es schneiden. Es soll geklärt werden, welche Verfahren notwendig sind, um die Mengenoperationen zwischen den beiden Volumina auszuführen.

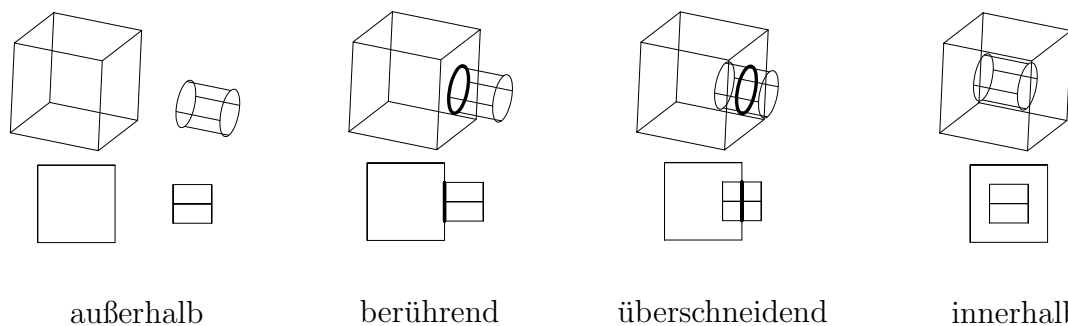


Abbildung 4.2: Lage eines Körpers zu einem anderen (Würfel: A; Zylinder: B)

außerhalb

Operation	Ergebnis	Beschreibung
$A \cup B$	$A; B$	A und B verbleiben eigenständige Volumina; keine Aktion
A / B	A	Nur A bleibt erhalten. B wird gelöscht.
B / A	B	Nur B bleibt erhalten. A wird gelöscht.
$A \cap B$	\emptyset	Beide Körper werden entfernt.

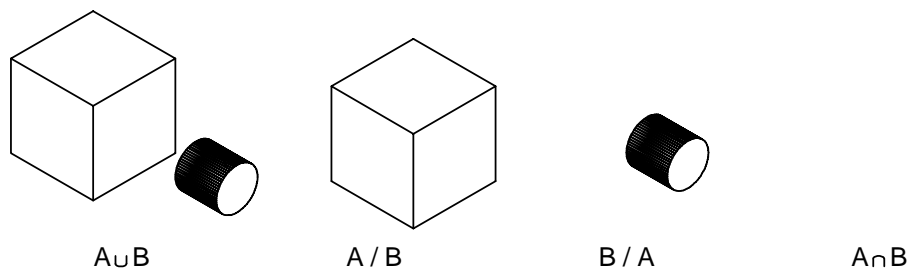


Abbildung 4.3: Ergebnis der booleschen Operationen (B außerhalb A)

berührend

Operation	Ergebnis	Beschreibung
$A \cup B$	$A + B$	Die Körper werden zusammengefasst. Von den sich berührenden Flächen wird die innere gelöscht. Die andere erhält die Definition der gelöschten Fläche als INNER-LOOP. Das Gesamtvolumen ergibt sich aus allen verbleibenden Flächen.
A / B	A	Nur A bleibt erhalten. B wird gelöscht.
B / A	B	Nur B bleibt erhalten. A wird gelöscht.
$A \cap B$	\emptyset	Beide Körper werden entfernt.

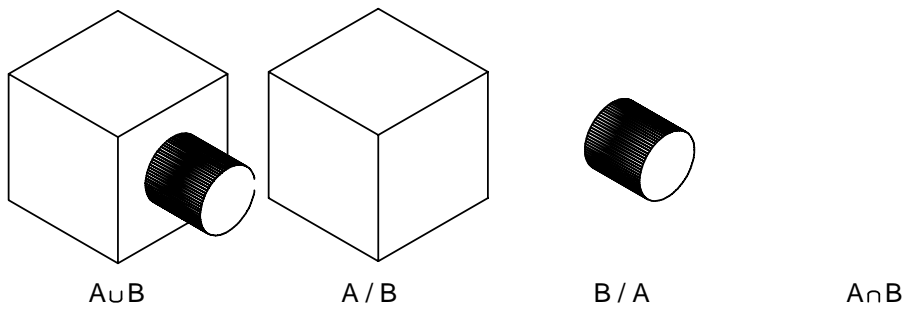


Abbildung 4.4: Ergebnis der booleschen Operationen (B berührt A)

überschneidend

Schneiden sich zwei Körper, so müssen Schnittfiguren gefunden werden. Als ein Schnittkörper wird $cutted(B)$ eingeführt. Er ist der Teil von B, welcher nicht in A liegt. Sein Gegenstück ist $-cutted(B)$, der Teil von B, welcher in A liegt.

Operation	Ergebnis	Beschreibung
$A \cup B$	$A + cutted(B)$	Zur SHELL von A werden die FACES von B addiert, welche außerhalb von A liegen. Die geschnittene Fläche erhält eine innere LOOP. Die schneidenden Flächen werden auf die Schnittkanten verkürzt.
A / B	$A + -cutted(B)$	Das Volumen A erhält zusätzlich die Flächen von B, welche in A teils oder voll enthalten sind. Die geschnittene Fläche bekommt eine INNER-LOOP. Die schneidenden Flächen werden auf die Schnittkanten verkürzt.
B / A	$cutted(B)$	Das Volumen B erhält zusätzlich die Flächen von A, welche in B teils oder voll enthalten sind. Die geschnittene Fläche bekommt eine INNER-LOOP. Die schneidenden Flächen werden auf die Schnittkanten verkürzt.
$A \cap B$	$-cutted(B)$	Die geschnittenen Flächen werden auf die Schnittkanten verkürzt. Alle Flächen die in beiden Volumen teils oder voll enthalten sind, werden zu einer Oberfläche zusammengefasst.

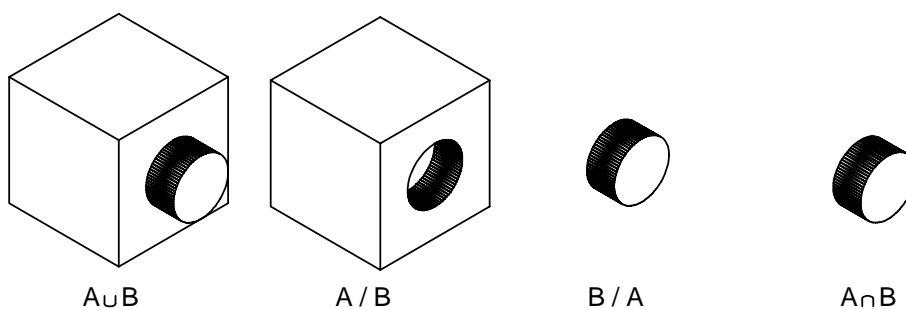


Abbildung 4.5: Ergebnis der booleschen Operationen (B schneidet A)

innerhalb

Operation	Ergebnis	Beschreibung
$A \cup B$	A	Nur A bleibt erhalten. B wird gelöscht.
A / B	$A - B$	A erhält zusätzlich zu seiner Beschreibung die äußere SHELL von B. Die innere SHELL von B wird an die OUTREGION geknüpft.
B / A	\emptyset	Beide Körper werden entfernt.
$A \cap B$	B	Nur B bleibt erhalten. A wird gelöscht.

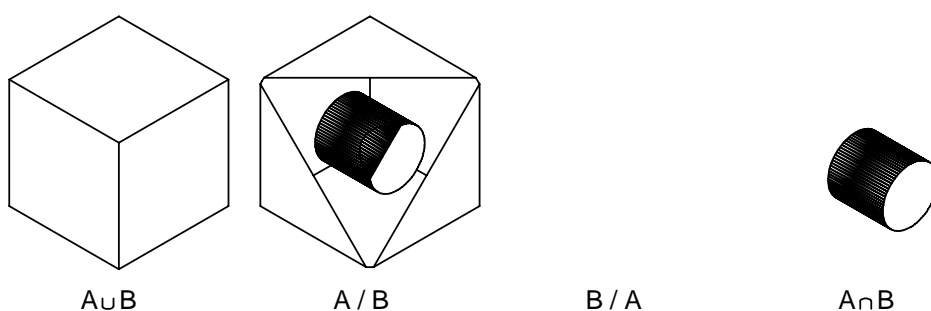


Abbildung 4.6: Ergebnis der booleschen Operationen (B innerhalb A)

Ein Sonderfall liegt vor, wenn die boolesche Operation auf zwei Körper angewandt werden soll, welche nicht im selben Volumen liegen. In diesem Fall kann eine Überschneidungsabfrage ergeben, dass sich die Körper an den Grenzen ihrer umgebenden Volumina berühren. Durch geeignete Abfragen sind die Probleme, welche solch ein Szenario mit sich bringt, auszuschließen.

Logischerweise benötigt man Überschneidungsalgorithmen nur, wenn die Körper sich schneiden. Die anderen Fälle implizieren lediglich topologisches Umordnen der Datenstruktur. Die Verfahren können dann wesentlich vereinfacht werden. Es stellt sich jedoch die Frage, wie möglichst effizient die Abfrage über die Lage von Körpern zueinander gestaltet werden kann. Auf die konkreten Überschneidungsabfragen soll hier nicht eingegangen werden. Es wird aber ein Werkzeug vorgestellt, welches Abfragen beschleunigen kann.

4.3 bounding boxes

Die für Überschneidungsabfragen einfachste Geometrie ist ein Quader. Mit seiner Information über Orthogonalität und planare Flächen können die Algorithmen sehr einfach gehalten werden. Das führt folgerichtig zu schnelleren Prozessen. Aus diesem Grund erhält in einem ersten Schritt der Überschneidungsabfragen jedes Volumen eine Box, in der es vollständig enthalten ist. Diese Box definiert sich über die Maxima des Volumens. Sie begrenzt es.

Der Aufruf eines konkreten Überschneidungsalgorithmus ist nur dann notwendig, wenn sich die bounding boxes zweier Körper schneiden. Schneiden sie sich nicht, so haben die untersuchten Körper keine gemeinsamen Punkte. Aufgrund von Sonderfällen muss die Definition des Schnitts (s.o. „überschneidend“) um die Fälle „berührend“ und „innerhalb“ erweitert werden (vgl. Abb. 4.8).

Mit Hilfe der genannten Definition für bounding boxes kann man zwei Typen entwickeln.

Typ I

Die bounding box liegt parallel zum Koordinatensystem. Dadurch kann sie durch zwei Punkte beschrieben werden.

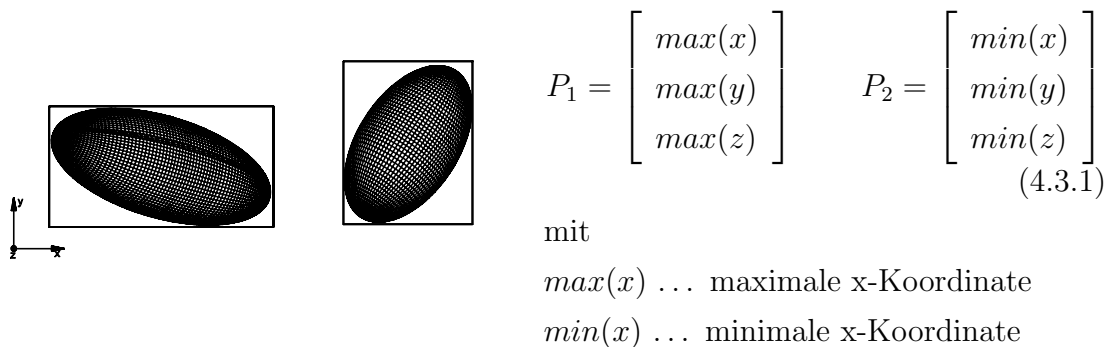


Abbildung 4.7: Bounding Box Typ I

Ein Punkt liegt innerhalb der Box, wenn gilt:

$$\begin{aligned} \max(x) &\geq P_x \geq \min(x) \\ \max(y) &\geq P_y \geq \min(y) \\ \max(z) &\geq P_z \geq \min(z) \end{aligned} \quad (4.3.2)$$

Es ist ersichtlich, dass das Erstellen dieses bounding box Typs sehr einfach ist. Das Gleiche gilt für den Abfragealgorithmus. Festzustellen ist jedoch, dass bei ungünstiger Drehung des Körpers die bounding box unverhältnismäßig groß wird. Aus diesem Grund kann der zweite Typ angewandt werden.

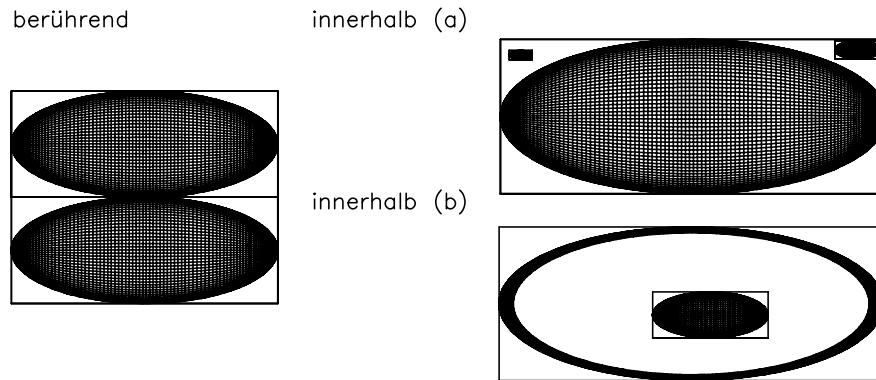


Abbildung 4.8: Sonderfälle beim Einsatz von bounding boxes

Typ II

Der bounding box Typ II berücksichtigt die Drehung des Körpers. Dabei wird ein gedrehter Quader um die äußeren Kanten des Körpers gelegt. Die Box berührt den Körper in seinen Maximalausdehnungen. Für die Definition des Körpers sind mindestens 4 Werte notwendig.

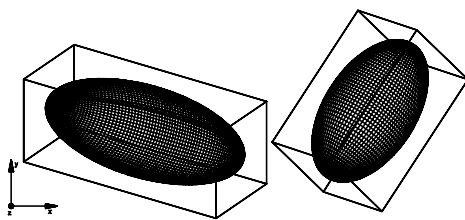


Abbildung 4.9: Bounding Box Typ II

Es gibt verschiedene Möglichkeiten diese Werte zu wählen. Zu empfehlen sind 2 Punkte, für die Definition einer box vom Typ I um den ungedrehten Körper und die 3 Winkel seiner Drehung im Raum. Offensichtlich ist die Handhabung dieses Typs wesentlich komplexer. Seine Anwendung empfiehlt sich daher nur bei der Abfrage von Körpern, deren Definition bekannt ist (Primitive).

Fazit: Für komplexe Strukturen, welche über die Operatoren der RED erstellt wurden, empfiehlt sich lediglich die Anwendung des ersten Typs. Die Erstellung der Begrenzung vom Typ II kann durch die Vielfältigkeit der Geometriebeschreibungen aufwändiger werden als die Überschneidungsabfrage an sich. Die im Hybrid-Modell vorhandenen (CSG-) Primitive können eine bounding box des zweiten Typs erhalten. Ihre Definition kann von vornherein bekannt sein. Sie kann in Einzelfällen einfacher sein als die vom Typ I. Ein Beispiel dafür ist ein Ellipsoid, welches über seine Mittelpunktkoordinate, seine Ausdehnungen und die Drehung im Raum beschrieben wird.

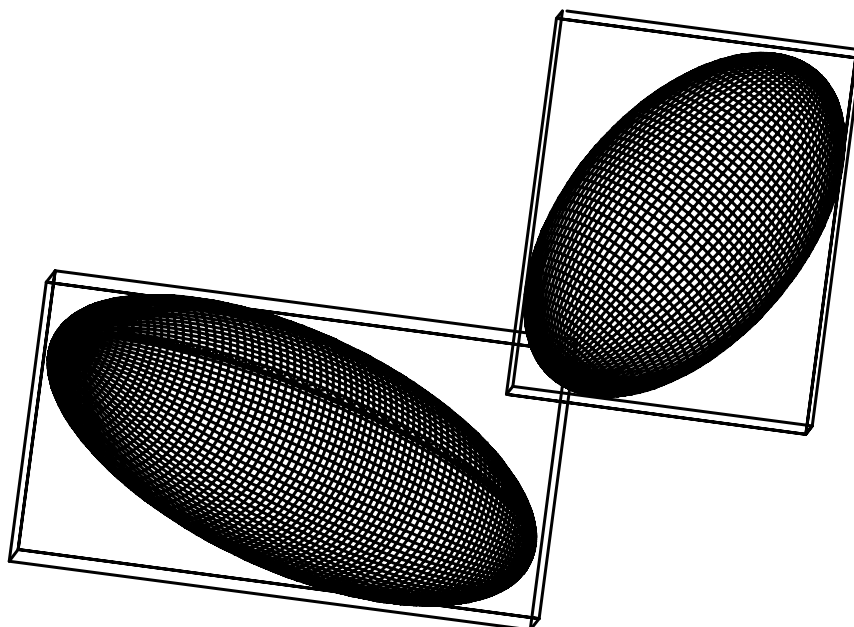
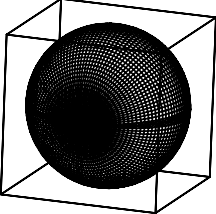
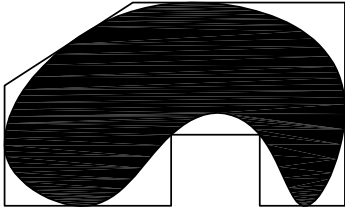


Abbildung 4.10: Schneidende Bounding Boxes

In boundary-Modellen ist zumindest für die Flächen ebenfalls eine bounding box vorzusehen. Man erreicht damit, dass tatsächlich nur die Flächen per Überschneidungsalgorithmus getestet werden, deren bounding boxes sich schneiden. In vielflächigen Körpern ergibt sich somit eine wesentlich höhere Effizienz. Es ist wiederum auch leichter möglich, aus den bounding boxes der Flächen (oder untergeordneten Elementen) eine bounding box für den Körper zu erstellen.

Sonderformen

Je genauer eine bounding box den Körper beschreibt, desto größer ist die Wahrscheinlichkeit, sich nicht schneidende Volumina schon vor dem konkreten Überschneidungsalgorithmus auszusortieren. Je einfacher die Beschreibung, Erstellung und Abfrage der bounding box, desto schneller kann mit ihr gearbeitet werden. Aus diesen Gründen sollen zwei Sonderformen, deren Form nicht mehr dem Quader gleicht, genannt werden.

Kugel	Polyeder
<p><u>Vorteil:</u> einfachste Überschneidungsabfrage über den Betrag des Abstandsvektors</p> <p><u>Nachteil:</u> Mittelpunkt der Kugel ist schwer zu finden Unterschied zwischen Körper und Kugel meist zu groß</p> 	<p><u>Vorteil:</u> in boundary Modellen: leicht aus bounding boxes der Flächen zu erstellen speziell bei konkaven, nach innen gewölbten, Flächen bessere Anpassung an den Körper</p> <p><u>Nachteil:</u> komplexe Abfrage, Erstellung</p> 

Kapitel 5

Implementation in SLang

SLang ist fast ausschließlich in der Programmiersprache C geschrieben. Aus diesem Grund wird die zu implementierende Datenstruktur und alle aufgesetzten Algorithmen und Abfragen in C++, der Weiterentwicklung von C, programmiert. Diese Programmiersprache ermöglicht objektorientiertes Programmieren (ooP). Dadurch können die in der Datenstruktur darzustellenden Elemente als Objekte (Klassen) programmiert werden.

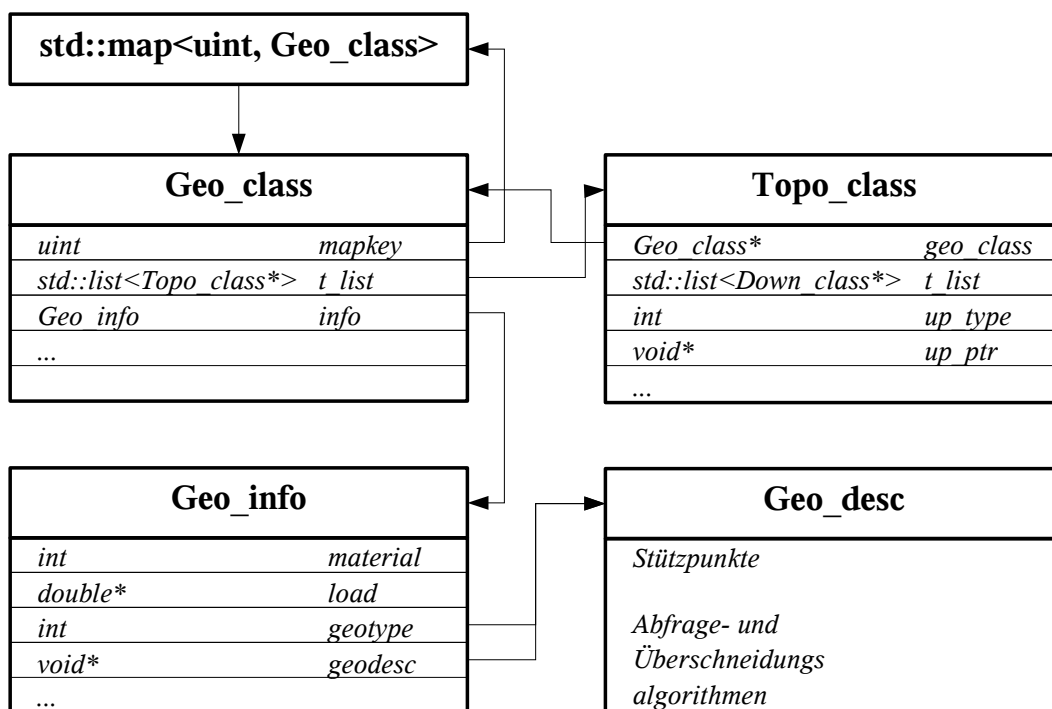


Abbildung 5.1: Allgemeiner Aufbau und Verknüpfung der Objekte

5.1 Datenstruktur

Beim Betrachten der Abbildung 3.1 kann man feststellen, dass die Verknüpfungen der meisten Geometrieelemente sich ebenso ähnlich sind, wie die zwischen den Topologieelementen. Die Klassenbeschreibungen der geometrischen bzw. topologischen Elemente werden demnach untereinander große Gemeinsamkeiten in ihrer Struktur haben. Abbildung 5.1 zeigt den allgemeinen Aufbau der Objekte für die Tiefe 1. Durch die auf- und abwärtigen Verknüpfungen der topologischen Elemente (\geq Tiefe 2) entsteht die RED. Nachfolgend soll der Aufbau der allgemeinen Elemente erläutert werden.

Jedes topologische Objekt erhält Verzeigerungen zur zugehörigen Geometrieinstanz und den hierarchisch eine Stufe höher und eine Stufe tiefer liegenden topologischen Objekten. Die abwärtigen Verknüpfungen werden in einer doppelt verketteten Liste der C-Standardlibrary abgespeichert. Dieser Datentyp kann dynamisch wachsen. Beim Löschen einzelner Elemente bleibt die Verzeigerung konsistent. Einen weiteren Vorteil für dieses Datentyps stellt die Möglichkeit dar, durch die doppelte Verkettung Nachbarschaftsbeziehungen abzuspeichern bzw. die Verbindungen zu ordnen.

Das Geometrieobjekt erhält die Verknüpfung zu seinen topologischen Objekten. Da ein geometrisches Objekt mehrere topologische Verbindungen haben kann, werden diese in einer Standard-*list* abgelegt. Um direkt auf ein geometrisches Objekt zugreifen zu können, z.B. beim Erstellen oder Löschen, werden die geometrischen Objekte in einer Standard-*map* abgelegt. Eine *map* speichert Objekte und eindeutige Schlüssel als Paar. Der Zugriff auf ein Objekt erfolgt über seinen Schlüssel (*mapkey*). Zur Vereinfachung einiger Abfragen wird dieser Schlüssel am Geometrieobjekt zusätzlich abgespeichert. Es entsteht eine doppelt verknüpfte *map*.

Angehängte Daten, wie Material, Vernetzerinformationen und geometrische Beschreibungen werden in einer *_INFO*-Klasse zusammengefasst. Diese Kapselung von Attributen garantiert die Übersichtlichkeit und erleichtert die Weiterentwicklung. Jedes Geometrieobjekt bekommt einen solchen „Informationscontainer“. Darin befindet sich ein *void*-Pointer auf die konkrete Geometriebeschreibung. Der Typ der darin gespeicherten Beschreibung, ist in dem Attribut *geotype* gespeichert. Mit dieser Hilfe kann der *void*-Pointer gecastet werden. Die Geometriebeschreibung einer REGION ist ein CSG-solid.

Die **Beschreibungsklassen** enthalten u.a. Stützpunkte, parametrische Beschreibungen, Abfrage- und Überschneidungsalgorithmen. Mit ihnen kann jede Geometrieform implementiert werden. Durch Mehrinformation über den Typ und die Kapselung in Beschreibungsklassen können Algorithmen, z.B. die Schnittpunktbestimmung, wesentlich effizienter gestaltet werden als mit statischen Beschreibungen direkt innerhalb der Geometrieklasse. Als Beispiel wurden Ellipsoidsurface (FACE), Ellipsenbogen (EDGE), Quader (CSG-solid/REGION) und ELLIPSOID (CSG-solid/REGION) implementiert. Sie sollen als Hilfestellung für Weiterentwicklungen und zur Verifikation des Konzeptes dienen.

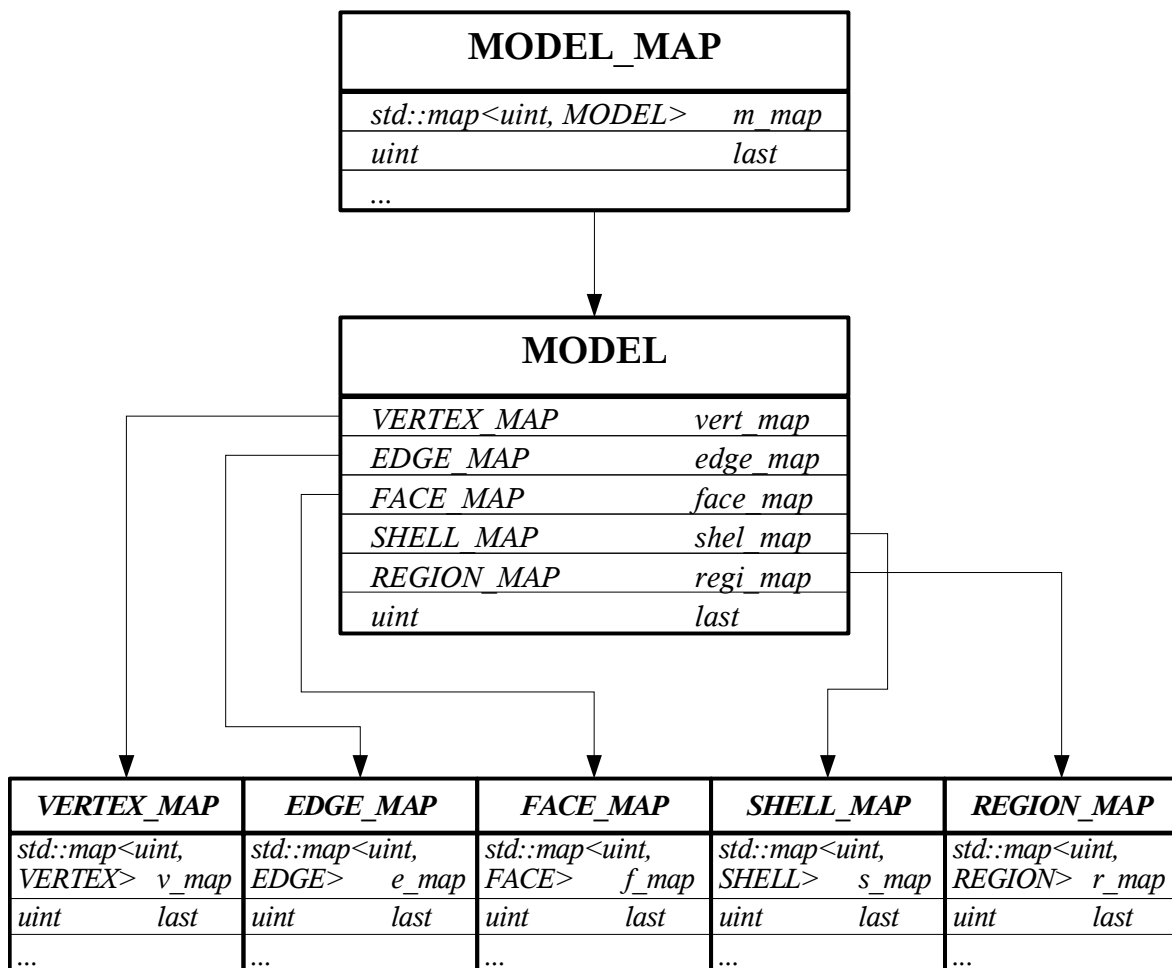


Abbildung 5.2: Speicherstruktur verschiedener Modelle und der zugehörigen geometrischen Elemente

Im **MODEL** enthalten sind die *maps* der zugehörigen Geometrieelemente. Das Model erhält so die Information über alle Elemente und kann sie verwalten. Die Möglichkeit, mehrere Entwicklungsstände in der Datenstruktur abzuspeichern, wird dadurch erreicht, dass die zugehörigen Modelle in einer **MODEL-map** integriert werden (Abb.5.2).

Die genannten Operatoren und Operationen werden auf der Ebene des Modells aufgerufen und ausgeführt. Die zuvor genannten Operatoren sind Methoden des Modells. Anlegen, Löschen und Modifizieren einzelner Elemente übernimmt das Modellobjekt.

5.2 Kommandos und Befehle

Slang basiert auf einem Kommando-Interpreter. Die in einem input-file enthaltenen Befehle werden eingelesen und ausgeführt. Im Allgemeinen besteht jedes Slang Kommando aus vier Teilen. Sie werden durch Kommata (,) getrennt. Jeder Teil enthält eine Liste von Argumenten. Diese Argumente werden durch ein Leerzeichen () von einander getrennt. Das Kommando endet mit einem slash (/) [4]. Der Aufbau der Slang Kommandos gliedert sich wie folgt:

1. ACTION , *besteht aus zwei Worten, GROUP und IDENTIFIER.*
2. ATTRIBUTES , *sind optional. Werden sie nicht angegeben, werden vordefinierte Werte verwendet.*
3. INPUT , *sind die vom Kommando benötigten Parameter und Werte.*
4. OUTPUT /

Die Kommandos, welche die Datenstruktur betreffen, werden in der Kommandogruppe „RED“ zusammengefasst. Folgende Kommandos wurden implementiert:

- **red create** *Erstellen eines Objektes,*
- **red delete** *Löschen eines Objektes,*
- **red info** *Ausgabe der gesamten Struktur in Textform,*
- **red modify** *Ändern eines Objektes,*
- **red boolean** *Ausführen boolescher Operationen,*
- **red export** *Interface zu anderen Geometriedatenstrukturen oder Vernetzern.*

Im Folgenden werden die Funktionsweise dieser Kommandos und ihre Übergabeparameter näher erläutert:

red create**Attributes:**

`auto`¹/ `man`, `model`/ `box`/ `ellipsoid`/ `elliptic_arc`/ `vertex`/ `edge`/ `face`/ `region`,

bei `auto`: `noreplace`,

bei `man`: `noreplace`/ `replace`,

Input:

bei `man`: *integer* mapkey

bei `box`: 6**double* box-parameter ($x_1, y_1, z_1, x_2, y_2, z_2$)

bei `ellipsoid`: 9**double* Ellipsoid-parameter (a, b, c -Ausdehnungen, c_x, c_y, c_z -Mittelpunkt-
koordinate, θ, ψ, ϕ -Verdrehung)

bei `elliptic_arc`: 2**integer* VERTEX-mapkeys, 3**double* Koordinaten des Zentrums,
3**double* Koordinaten des zweiten Viertelpunktes

bei `vertex`: 3**double* Koordinaten (x, y, z)

bei `edge`: 2**integer* VERTEX-mapkeys (x, y, z)

bei `face`: *integer* LOOP-size (n), n **integer* EDGE-mapkeys

bei `region`: *integer* size (n), n **integer* FACE-mapkeys

Output: –**Beispiel:**

```
red create, , , /
```

*erstellt automatisch ein MODEL und setzt dieses als aktuelles MODEL. Ist noch kein MODEL vorhanden, wird für das neue MODEL mapkey 1 vergeben./

```
red create, box man, 2 0. 0. 0. 10. 10. 10. , /
```

*erstellt einen Quader. Die zugehörige REGION erhält den mapkey 2 und den Quader als geometrische Beschreibung. Die Eckpunkte des Quaders sind (0, 0, 0) und (10, 10, 10)./

```
red create, vertex replace man, 9 -5. 5. 5., /
```

*legt VERTEX 9 mit den Koordinaten (-5, 5, 5) an. Ist der Knoten schon vorhanden, so wird er ersetzt./

¹Unterstrichene Werte sind voreingestellt. Wird kein anderes Attribut angegeben wird die Voreinstellung verwendet.

```
red create, edge man, 13 8 9, /
```

*verbindet die VERTICES 8 und 9 zu Kante 13./

```
red create, elliptic_arc man, 14 9 1 0. 5. 5. 0. 0. 0., /
```

*erzeugt Kante 14 zwischen den VERTICES 9 und 1 mit der geometrischen Beschreibung eines Ellipsenbogens. Das Zentrum des Ellipsenbogens liegt bei (0, 5, 5).

Zwei Viertelpunkte der Ellipse befinden sich am VERTEX 9 und bei den Koordinaten (0., 0., 0.)./

```
red create, face man, 7 3 13 16 12, /
```

*bildet aus den Kanten 3, 13, 16 und 12 die Fläche 7./

```
red create, region man, 7 5 4 7 8 9 10, /
```

*Bilden die 5 Flächen 4, 7, 8, 9 und 10 ein geschlossenes Volumen, so wird die REGION 7

angelegt. /

```
red create, ellipsoid man, 3 1. 2. 1. 2. 3. 2. 0. 1. 1. , /
```

*Die REGION 3 wird erstellt. Ein Ellipsoid (CSG-solid) mit den Abmessungen 1.0, 2.0 und 1.0, einem Mittelpunkt mit den Koordinaten 2, 3, 2 und den Verdrehungen 0°, 1° und 1° wird als geometrische Beschreibung mit REGION 3 verknüpft. /

Kurzbeschreibung: Das Kommando **red create** nutzt zum Erstellen der Objekte die Make-Operatoren. Wird ein Element **manuell** erstellt und die Option **replace** ist ausgewählt, so wird das Vorgängerelement mit allen seinen Verbindungen gelöscht. Dazu kommen die Kill-Operatoren zum Einsatz. Bevor geometrische Elemente erstellt werden können, muss ein MODEL vorhanden sein. Daher muss der Befehl **red create, , , /** (oder äquivalent) vor Beginn der Geometrieerstellung aufgerufen werden.

red delete

Attributes: model/ vertex/ edge/ face/ region

Input:

integer mapkey

Output: –

Beispiel:

```
red delete, vertex, 9, /
```

*VERTEX 9 wird inklusive seiner Verbindungen und den durch ihn definierten Objekten gelöscht./

Kurzbeschreibung: Das Kommando **red delete** ruft die Kill-Operatoren auf.

red info

Attributes: total

Input: –

Output: –

Beispiel:

```
red info, total, , /
```

*Die Datenstruktur wird im Terminal ausgegeben./

Kurzbeschreibung: Dieses Kommando soll als Hilfestellung für den Nutzer verstanden werden. Topologische Beziehungen und geometrische Nummerierungen (mapkeys der Elemente) werden hier auf ein Blick dargestellt.

red modify

Attributes: set/ length / glue,

bei **set:** model

bei **length:** region

bei **length:** face, edge, vertex

Input:*integer* mapkeybei **length**: *double* spezifische Länge bei **glue**: *integer* zweiter mapkey, *double* Toleranz**Output:** –**Beispiel:**`red modify, region length, 2 2.5, /`**setzt die spezifische Länge für alle mit der REGION 2 verbundenen VERTICES auf 2.5./*`red modify, model set, 2, /`**das MODEL mit dem mapkey 2 wird aktuelles MODEL./*`red modify, glue face, 3 10 3.0, /`**die Flächen 3 und 10 werden zu einer Fläche verschmolzen. Die zu vereinigenden VERTICES der**Flächen dürfen nicht mehr als 3.0 auseinanderliegen. Die verbleibende Fläche hat die Position und Definition von Fläche 3. Fläche 10 wird gelöscht./***Kurzbeschreibung:** Innerhalb des Kommandos **red modify** werden Befehle ausgeführt, welche Daten innerhalb der Struktur verändern.**red boolean****Attributes:** insert, region, single/list**Input:***integer* mapkey der äußeren REGIONbei **single**: *integer* mapkey der inneren REGIONbei **list**: *integer* Anzahl der inneren Volumina (*n*), *n*integer* mapkeys der inneren Volumina**Output:** –

Beispiel:

```
red boolean, insert list, 4 2 3 4 5 6, /
```

*Die 4 Volumina 3, 4, 5 und 6 werden in das Volumen 2 eingefügt./

Kurzbeschreibung: Zum Ausführen boolescher Operationen ist das Kommando `red boolean` vorgesehen. Zur Zeit ist lediglich der Insert-Operator implementiert.

red export

Attributes: `gmsh`, `unmeshed`/ `meshed`, `noexecute`/ `execute`

Input: –

Output: –

Beispiel:

```
red export, gmsh meshed execute, , /
```

*Die Datenstruktur wird in ein `gmsh`-input-file geschrieben. Das Programm `gmsh` vernetzt die Geometrie und speichert die vernetzte Struktur ab. Mit diesem Netz wird `gmsh` geöffnet./

Kurzbeschreibung: Das Kommando `red export` soll hauptsächlich die Interaktion mit den verschiedensten Vernetzern ermöglichen. Zur Zeit ist der Export in ein `gmsh`-input-file möglich.

5.3 Beispiele

Die Datenstruktur wird am Beispiel eines Betonmodells verifiziert. Dazu wird ein Quader mit den Abmaßen 10x10x10 angelegt. In diesen Quader werden Ellipsoide eingefügt. Dazu sind die Make- und Insert- Operatoren notwendig. Die zum Löschen, Modifizieren und Exportieren notwendigen Kommandos werden in weiterführenden Beispielen erprobt. Anhand dieser Beispiele wird die Funktionsweise der Befehle erläutert.

Zu Beginn muss das MODEL angelegt werden.

```
red create, , , /
```

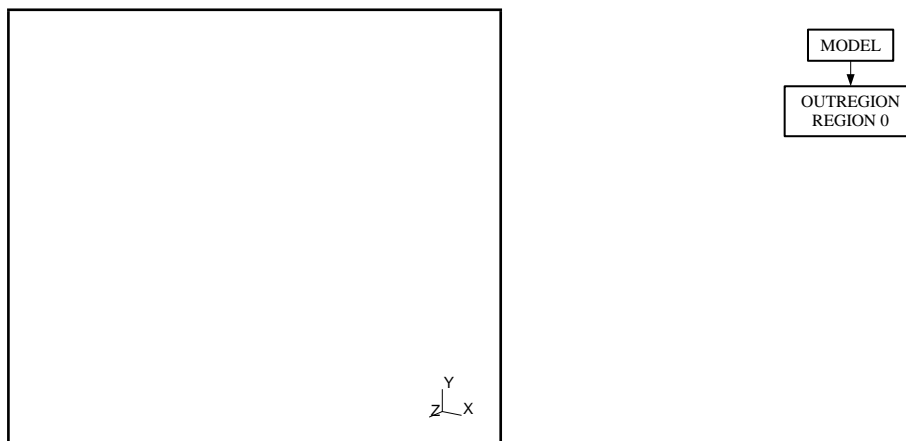


Abbildung 5.3: red create: Erstellen eines MODELS

Ist noch kein MODEL angelegt, erhält das neue MODEL den mapkey 1 innerhalb der MODEL.MAP. Mit dem Anlegen eines MODELS wird automatisch auch die OUTREGION angelegt. Das neu erstellte MODEL wird innerhalb der Datenstruktur als aktuelles MODEL gesetzt. Alle nachfolgenden Kommandos arbeiten auf dem aktuellen MODEL.

Die kleinsten Elemente innerhalb einer Geometrie sind die Knotenpunkte. Nach dem Erstellen des MODELS können sie kreiert werden. Zunächst werden die acht Eckpunkte des Quaders erstellt. Dazu wird der Operator *m_sv* genutzt.

```
red create, vertex man, 1 0 0 0, /
red create, vertex man, 2 10 0 0, /
red create, vertex man, 3 10 10 0, /
red create, vertex man, 4 0 10 0, /
red create, vertex man, 5 0 0 10, /
```

```

red create, vertex man, 6 10 0 10, /
red create, vertex man, 7 10 10 10, /
red create, vertex man, 8 0 10 10, /

```

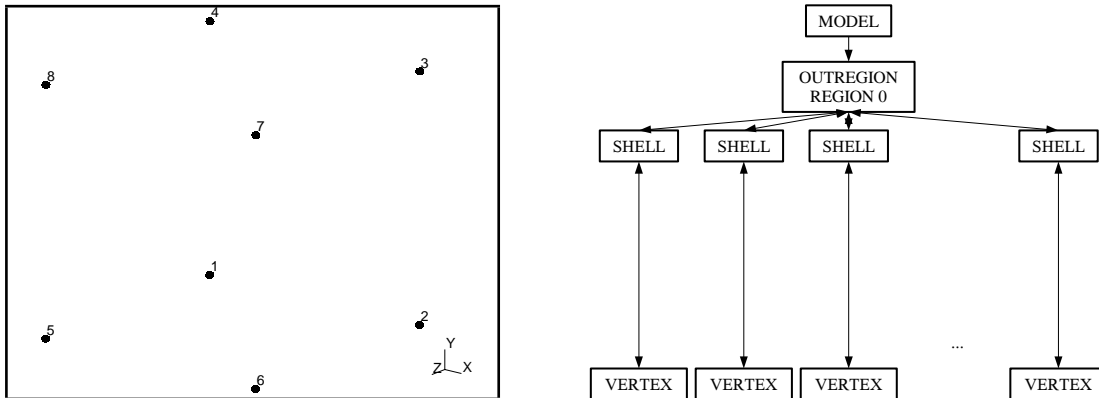


Abbildung 5.4: red create: Anlegen der Knotenpunkte

Danach werden die 8 VERTICES durch 12 Kanten verbunden. Im Hintergrund ruft das Kommando dazu jeweils den *m_eks* Operator auf.

```

red create, edge man, 1 1 2, /
red create, edge man, 2 2 3, /
red create, edge man, 3 3 4, /
red create, edge man, 4 4 1, /
red create, edge man, 5 1 5, /
red create, edge man, 6 5 6, /
red create, edge man, 7 6 7, /
red create, edge man, 8 7 8, /
red create, edge man, 9 8 5, /
red create, edge man, 10 6 2, /
red create, edge man, 11 7 3, /
red create, edge man, 12 8 4, /

```

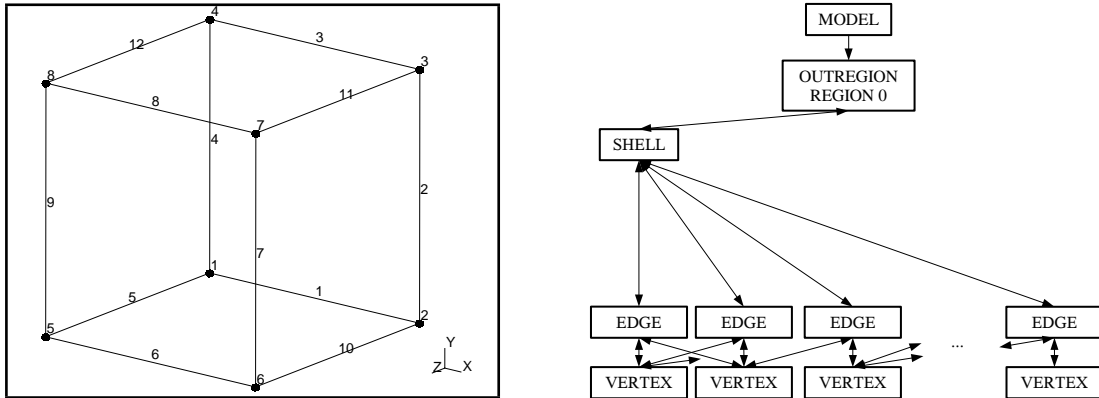


Abbildung 5.5: red create: Verbinden der Knotenpunkte mit Kanten

Jeweils 4 Kanten bilden einen geschlossenen Ring. Dieser Ring beschreibt die Grenzen einer Fläche. Mit den Befehlen

```

red create, face man, 1 4 1 5 10 6, /
red create, face man, 2 4 3 11 8 12, /
red create, face man, 3 4 2 7 10 11, /
red create, face man, 4 4 4 5 9 12, /
red create, face man, 5 4 6 7 8 9, /
red create, face man, 6 4 1 2 3 4, /
    
```

werden die 6 Flächen des Quaders erzeugt. Der Operator *m_fl* kontrolliert dabei, ob die angegebenen Kanten tatsächlich einen geschlossenen Ring ergeben.

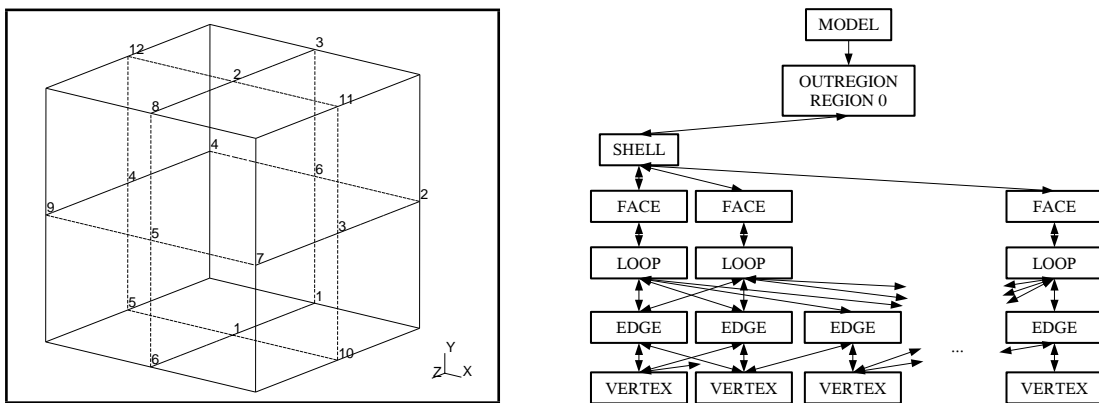


Abbildung 5.6: red create: Erzeugen der 6 Flächen

Durch die 6 erstellten Flächen wird das Volumen des Quaders eingeschlossen. Mit Hilfe des Befehls

```
red create, region man, 2 6 1 2 3 4 5 6, /
```

wird der Operator `m_r` aufgerufen. Dieser überprüft die Geschlossenheit des Volumens und legt, bei positivem Ergebnis, eine neue REGION an.

Durch Aufrufen des Kommandos

```
red modify, region length, 2 2.5, /
```

wird für alle Knotenpunkte, welche mit dem Quader (REGION 2) verbunden sind, die spezifische Länge auf 2.5 gesetzt. Dieser Wert ist ein Maß für den Abstand der FE-Knoten und somit Grundlage für die Vernetzung. Diese wird mit

```
red export, gmsh meshed execute, , /
```

aufgerufen.

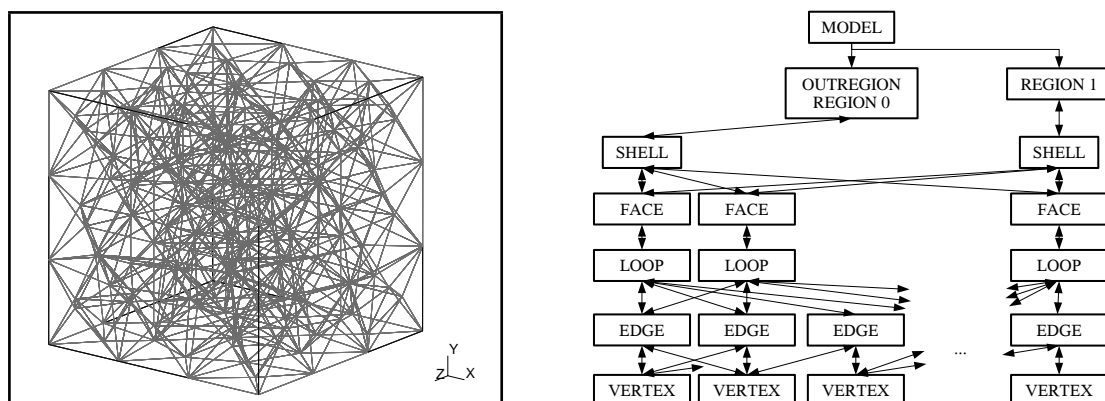


Abbildung 5.7: red create: Erzeugen des Volumens und Vernetzung

Das Anlegen des Quaders zeigt, dass die Konstruktion einfacher geometrischer Primitive sehr aufwendig sein kann. Aus diesem Grund kann der Quader auch als CSG-solid erstellt werden. Der Befehl lautet:

```
red create, box man,  
2 0. 0. 0. 10. 10. 10. , /
```

Als geometrisches Primitiv wird nun ein Ellipsoid angelegt. Die spezifische Länge dieses solids wird mit 0.5 festgelegt.

```
red create, ellipsoid man,
      3 1. 2. 1. 2. 3. 2. 0. 1. 1. , /
red modify, region length, 3 .5, /
```

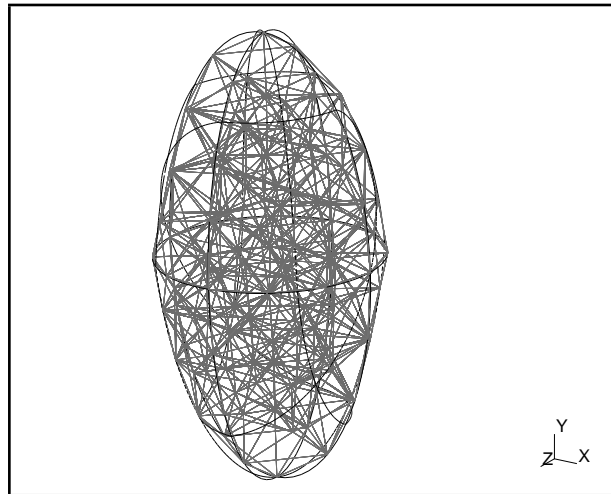


Abbildung 5.8: red create: Ellipsoid

Durch Aufruf des Insert-Operators wird der Ellipsoid (REGION 3) in den Quader (REGION 2) eingefügt.

```
red boolean, insert single, 2 3, /
```

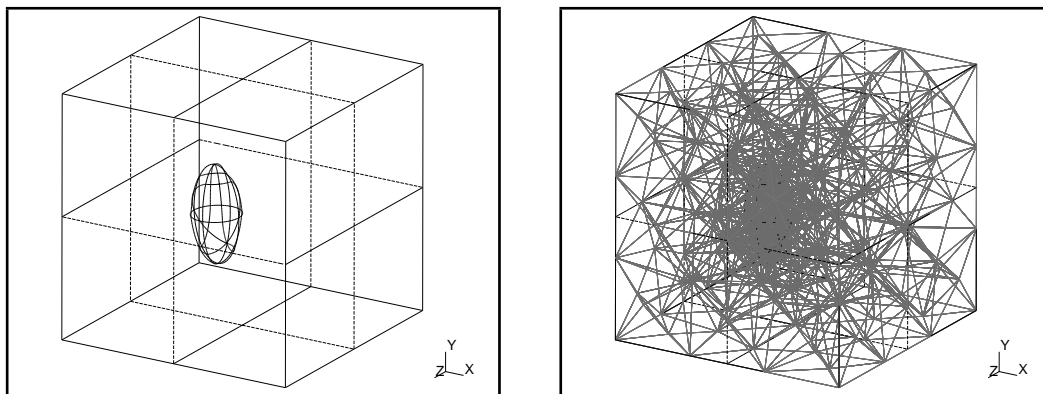


Abbildung 5.9: red boolean: Ellipsoid im Quader unvernetzt, vernetzt

In nonmanifold-Strukturen ist es möglich, dass Körper gemeinsame Knotenpunkte, Kanten oder Flächen haben. Existieren zwei Elemente mit gleicher geometrischer Beschreibung in unmittelbarer Nähe, so kann man diese zu einem Objekt zusammenfassen. Im folgenden Beispiel sind sich die Flächen zweier Quader

```
red create, box man,
    1 0. 0. 0. 10. 10. 10. , /
red create, box man,
    2 12. 0. 0. 20. 10. 10. , /
```

sehr nah. Durch den Befehl

```
red modify, glue face, 3 10 3.0 , /
```

werden die Flächen 3 (Quader 1) und 10 (Quader 2) mit vereint. Dazu wird geprüft, ob der Abstand der Flächen innerhalb der Toleranzgrenze von 3.0 liegt. Ist dies der Fall, so werden alle zur äußeren LOOP gehörigen Kanten und Knotenpunkte zusammengeführt.

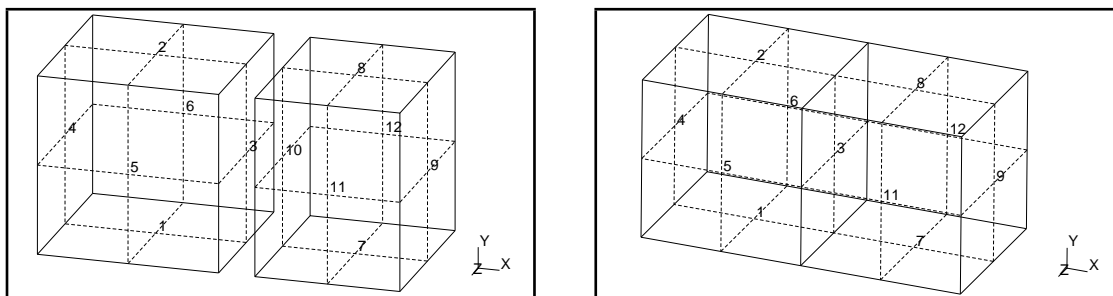


Abbildung 5.10: red modify: Verschmelzen zweier Flächen

Durch das Löschen des VERTEX 6, mit dem beide Volumina verbunden sind, werden auch die Körperbeschreibungen gelöscht.

```
red delete, vertex , 6 , /
```

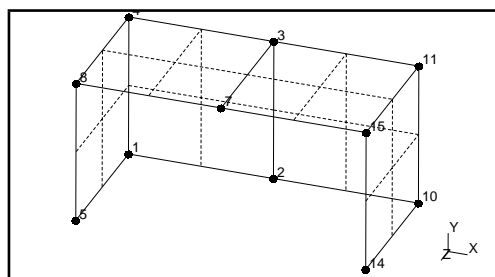


Abbildung 5.11: red delete: Löschen eines VERTEX und all seiner Verbindungen

Kapitel 6

Zusammenfassung

Die numerische Auswertung physikalischer Prozesse erfordert eine Modellierung der realen Welt. Grundlage bildet dabei eine realistische und konsistente Beschreibung der Geometrie. Im Programmsystem *Slang* existierte zu Beginn dieser Arbeit keine Datenstruktur zur Verwaltung von Geometrien. Ziel dieser Arbeit war es, eine solche Datenstruktur in das bestehende Programmsystem *Slang* zu implementieren.

Ziel innerhalb des Konzeptes war es, keinerlei Einschränkungen hinsichtlich der Darstellbarkeit zu machen. Aus diesem Grund wurde als Grundstruktur eine BRep (***B**oundary **R**epresentation **D**ata **S**tructure*) gewählt. Speziell für die spätere Nutzung der Geometriedaten ist es wichtig, nonmanifold Geometrien darstellen zu können. Die von Weiler [14] entwickelte RED (***R**adial-**E**dge **D**ata **S**tructure*) ist eine BRep-Struktur, welche die Darstellung von nonmanifold Geometrien ermöglicht. Die RED wurde als Basisstruktur implementiert.

Die Erzeugung einfacher geometrischer Primitive in BRep-Strukturen kann sehr umfangreich werden. Aufgrund vielfältiger Möglichkeiten der Darstellbarkeit von Geometrien gestalten sich (Überschneidungs-)Abfragen für BRep Geometrien kompliziert und fehleranfällig. Aus diesen Gründen wurde die Struktur als **hybrides Modell** angelegt. Dabei ist es möglich, CSG-Primitive zu erstellen. Diese werden als Volumen in der RED gespeichert. Die REGION der RED erlangt dadurch die geometrische Beschreibung eines CSG-solids. Auf dieser Basis können Abfragen auf Grundlage der geometrischen Information erfolgen. Um die Weiterbearbeitung eines Primitives zu ermöglichen, kann es in eine BRep-Struktur konvertiert werden.

Die Implementierung erfolgte in der Programmiersprache C++. Sie macht es möglich, objektorientiert zu programmieren. Zudem werden in der C-Standardlibrary Datentypen mit dynamischer Größe bereitgestellt. Um die **geometrischen Objekte**, z.B. beim Erstellen oder Löschen, direkt ansprechen zu können, wurden sie in *maps* der Standardlibrary abgelegt. Der Zugriff auf die Elemente einer *map* geschieht über einen Schlüssel, den *mapkey*. Um eine doppelte Verknüpfung der *map* zu realisieren, wurde der *mapkey* auch am zugehörigen geometrischen Objekt abgespeichert.

Angehängte Daten werden in **Info-Klassen** gekapselt. Jedes geometrische Objekt erhält eine Instanz dieser Informationscontainer. Darin sind z.B. Daten für den Vernetzer, Material oder die konkrete geometrische Beschreibung enthalten. Grundsätzlich kann jede Geometrieformulierung, auch von Kanten oder Flächen, als Primitiv definiert werden. Diese Definition wird, der objektorientierten Programmierung folgend, in Beschreibungsklassen gespeichert. Methoden dieser Klassen sind z.B. Abfragealgorithmen, Attribute sind Stützpunkte oder parametrische Beschreibungen. Der Informationscontainer hält eine Verzeigerung zur geometrischen Beschreibungsklasse.

Zeiger auf die **topologischen Elemente** sind in Standard-*lists* gespeichert. In diese doppelt verketteten Listen sind Verknüpfungen zu topologisch abwärtigen Elementen. Sie werden ebenfalls als Attribut des zugehörigen Geometrieobjektes genutzt. Zusätzlich erhält jedes Topologieobjekt Verzeigerungen zum hierarchisch höherliegenden Topologieobjekt und zum Geometrieobjekt.

Am **MODEL** werden die *maps* der zugehörigen Geometrieelemente abgespeichert. Das MODEL erhält so die Information über alle Elemente und kann sie verwalten. Die Möglichkeit, mehrere Entwicklungsstände in der Datenstruktur abzuspeichern, wird dadurch erreicht, dass die zugehörigen Modelle in einer *MODEL-map* integriert werden.

Die vorgestellten **Operatoren** sind Basis aller Operationen auf der Datenstruktur. Als Methoden des MODELS organisieren sie jede Modifikation der Datenstruktur. Es wird sichergestellt, dass sämtliche topologischen Verknüpfungen gesetzt oder gelöst, und geometrische Objekte angelegt oder gelöscht werden. Dadurch ist die Datenstruktur zu jedem Zeitpunkt eindeutig. Topologisch unmögliche Verknüpfungen werden vermieden. Die Fehlerwahrscheinlichkeit wird verringert.

Mit Hilfe der Make-Operatoren ist es möglich, jedes erdenkliche Modell zu erstellen. Da das Erstellen komplexer Geometrien jedoch eine große Fehlerquelle darstellt, wur-

de untersucht, inwieweit **Mengenoperationen** auf BRep-Strukturen zu realisieren sind. Dabei beschränkt man sich auf Verfahren, welche Körper zueinander in logische Verbindungen bringen. Es wurden die booleschen Grundoperationen Vereinigung \cup , Intersection \cap und Differenz $/$ erläutert.

Wesentlicher Teil logischer Operationen ist die **Überschneidungsinformation**. Dazu wurden vier Möglichkeiten der Lage zweier Körper zueinander (*außerhalb, berührend, schneidend und innerhalb*) erläutert. Weiterhin wurde aufgezeigt, dass durch die Nutzung von *bounding boxes* die notwendigen Abfragealgorithmen effizienter gestaltet werden können. Es bleibt jedoch festzuhalten, dass durch die Menge an notwendigen Abfragealgorithmen, ihre Komplexität und Fehleranfälligkeit, eine robuste Umsetzung boolescher Operationen auf RED-Strukturen nur schwer zu realisieren ist. Aus diesem Grund wird vorgeschlagen, diese Mengenoperationen zunächst nur für die CSG-Primitive der Hybridstruktur zu implementieren.

Die `Slang`-Kommandos zur Nutzung der Datenstruktur wurden erläutert. Als Beispiel zur Verifikation diente ein Betonmodell. Dazu wurde mittels Make-Operatoren ein Quader und ein Ellipsoid innerhalb der Datenstruktur abgebildet. Danach wurde das Volumen des Ellipsoiden mittels Insert-Operator topologisch in das Volumen des Quaders eingefügt. Durch ein Interface zum Vernetzer *gms* ist es möglich, die entstandene Geometrie vernetzen zu lassen. Des Weiteren wurde in Beispielen die Funktionalität des Glue- und des Kill-Operators gezeigt.

Aus den gewonnenen Erkenntnissen dieser Arbeit werden folgende Möglichkeiten für die Weiterentwicklung der vorliegenden Datenstruktur vorgeschlagen:

- Interface zu verschiedenen Vernetzern (RED \rightarrow Vernetzer \rightarrow SLang // z.B. Ansys)
- Implementierung verschiedener Geometrieformulierungen (B-Spline, Nurbs, Funktionsbeschreibungen usw.)
- Überschneidungsalgorithmen (boolesche Operationen)
- Postprocessing (FE-Verschiebungsfigur \rightarrow RED)
- Erweiterung der Funktionalität der INFO-Klassen (Material- und Lastinformationen)
- Interface zu Eingabeformaten wie z.B. *.dxf

Literaturverzeichnis

- [1] Baumgart, B. G.: *Winged Edge Polyhedron Representation*; Stanford Artificial Intelligence Laboratory; Stanford University; 1972
- [2] Geuzaine, C.; Remacle, J.-F.: *Gmsh Reference Manual*; 2006
- [3] Gueorguieva, S.; Marcheix, D.: *Non-Manifold Boundary Representation for Solid Modelling*; Proc. of the International Computer Symposium; 1994
- [4] Bucher, C. et. al. *SLang* The Structural Language. Institute of Structural Mechanics, Bauhaus-University of Weimar, Germany; version 5.09; 2005
- [5] Jenter, J. *Entwicklung eines Boundary Representation Modells am Beispiel eines Quaders unter Verwendung von C++*; Diplomarbeit; TU Kaiserslautern; 2004
- [6] Lee, S. H.; Lee, K.: *Partial entity structure: a compact non-manifold boundary representation based on partial topological entities*; SMA '01, Proceedings of the sixth ACM symposium on Solid modeling and applications; Ann Arbor, MI, USA; p. 159 - 170; ISBN:1-58113-366-9; 2001
- [7] Mäntylä, M.: *An Introduction to Solid Modeling*; Computer Science Press; College Park, MD, USA; 1988
- [8] Muuss, M.: *Understanding the Preparation and Analysis of Solid Models*; Techniques for Computer Graphics; ed: Rogers & Earnshaw; Springer Verlag, New York; p. 109-172; ISBN 0-387-96492-4; 1987
- [9] Poutrain K.; Contensin, M.: *Dual Brep-CSG Collision Detection for General Polyhedra*; Ninth Pacific Conference on Computer Graphics and Applications (PG'01); p. 124-133; 2001
- [10] Roehrig, J.: *Topologische Datenstrukturen für Körper- und Netzmodellierung*; Dissertation; Ruhr-Universität Bochum; 1998

- [11] Rossignac, J.; Cardoze, D.: *Matchmaker: manifold breps for non-manifold r-sets; SMA '99, Proceedings of the Fifth Symposium on Solid Modeling and Applications; Ann Arbor, MI, USA; p. 31-41; June 1999*
- [12] Spitzer, A. W.; Weiss, W.: *Mit Hybridsystemen alle Möglichkeiten offenhalten; CAD/CAM Report Nr. 7; 1999*
- [13] Sun, W.; Hu, X.: *Reasoning Boolean operation based modeling for heterogeneous objects; Computer-Aided Design 34(6); p. 481-488; 2002*
- [14] Weiler, K.J.: *Topological structures for geometric Modeling; PhD Thesis; Rensselaer Polytechnic Institute; 1986*

Internetquellen (Stand 09/2006):

- [15] www.ime.usp.br/~lye/
- [16] www.wikipedia.org
- [17] www.matematik.de
- [18] <http://www-gap.dcs.st-and.ac.uk/~history/>
- [19] <http://www.matheprisma.uni-wuppertal.de/>

Abbildungsverzeichnis

2.1	Beispiele für manifolds und nonmanifolds	10
2.2	Königsberger Brücken [18], [19]	10
2.3	Beispiel der CSG [15]	11
2.4	Sweep-Modell: Trajektion, Translation, Rotation	11
2.5	Darzustellender Körper (links) und Octreezerlegung des umgebenden Raumes (rechts)	12
2.6	Oktalbaum	12
2.7	Grundprinzip der Boundary Representation Data Structure	13
2.8	Prinzip der Winged-Edge Data Structure	14
2.9	Beispiel für die Verknüpfung von CSG und BRep in einer hybriden Da- tenstruktur	16
3.1	Trennung von Topologie und Geometrie	17
3.2	topologische Beziehungen der non-manifold Elemente [14]	20
3.3	Verbindung VERTEX_USE - SHELL	21
3.4	Verbindung VERTEX_USE - LOOP_USE	21
3.5	Verbindung von zwei Kanten an einem Punkt	22

3.6	Verbindung von 3 Flächen an einer Kante nach Weiler [14]	23
3.7	Beschreibung eines Loches in einer Fläche	24
3.8	Topologische Verbindungen (aufwärts) einer in sich geschlossenen Fläche (vgl. Abb. 3.4 und 3.7)	25
3.9	Entscheidungsbaum für k_v	29
4.1	Boolesche Operationen	32
4.2	Lage eines Körpers zu einem anderen (Würfel: A; Zylinder: B)	35
4.3	Ergebnis der booleschen Operationen (B außerhalb A)	35
4.4	Ergebnis der booleschen Operationen (B berührt A)	36
4.5	Ergebnis der booleschen Operationen (B schneidet A)	37
4.6	Ergebnis der booleschen Operationen (B innerhalb A)	38
4.7	Bounding Box Typ I	39
4.8	Sonderfälle beim Einsatz von bounding boxes	40
4.9	Bounding Box Typ II	40
4.10	Schneidende Bounding Boxes	41
5.1	Allgemeiner Aufbau und Verknüpfung der Objekte	43
5.2	Speicherstruktur verschiedenener Modelle und der zugehörigen geometrischen Elemente	45
5.3	red create: Erstellen eines MODELS	53
5.4	red create: Anlegen der Knotenpunkte	54
5.5	red create: Verbinden der Knotenpunkte mit Kanten	55

5.6	red create: Erzeugen der 6 Flächen	55
5.7	red create: Erzeugen des Volumens und Vernetzung	56
5.8	red create: Ellipsoid	57
5.9	red boolean: Ellipsoid im Quader unvernetzt, vernetzt	57
5.10	red modify: Verschmelzen zweier Flächen	58
5.11	red delete: Löschen eines VERTEX und all seiner Verbindungen	58

Anhang

A.1 CD-ROM

LaTeX-Dateien

- *.tex-File
- zugehörige Bilderdateien
- diese Arbeit im pdf-Format

Programmierung

- Header-Dateien
- *.cpp-Dateien
- Testbeispiele
- doxygen-Dokumentation

A.2 Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Weimar, 20.09.2006

David Schneider

A.3 Thesen

- Zu Beginn dieser Arbeit existierte im Programmsystem *Slang* keine Datenstruktur zur Verwaltung von Geometriebeschreibungen. Es wurde ein Konzept für eine Geometriedatenstruktur erarbeitet. Aufgrund dieses Konzeptes erfolgte die Implementierung dieser Struktur in das Programmsystem *Slang*.
- Durch Kombination der Vorteile von CSG (**Constructive Solid Geometry**) und BRep (**Boundary Representation Data Structure**) können effiziente Algorithmen auf der Struktur ausgeführt werden. Einschränkungen in der Darstellbarkeit, welche die CSG mit sich bringt, müssen nicht gemacht werden.
- Die RED (**Radial-Edge Data Structure**) ist ein BRep-Struktur, welche geometrische und topologische Informationen klar voneinander trennt. Sie ermöglicht es, nonmanifold-Geometrien darstellen zu können.
- Es ist notwendig, nonmanifold-Geometrien darstellen zu können. Aus diesem Grund wurde für die BRep-Struktur des Hybrid-Datenmodells die RED gewählt.
- Die implementierte Datenstruktur ist ein Hybrid von CSG- und BRep-Darstellungsformen.
- Die **Geometrieelemente** werden in *maps* der C-Standardlibrary gespeichert.
- Jedes geometrische Objekt erhält eine Instanz einer **Info-Klasse**. In den Info-Klassen sind u.a. die Verzeigerung zur geometrischen Beschreibung, Material- oder Vernetzerinformationen gespeichert.
- Die **geometrischen Beschreibungen** sind in Klassen gekapselt. Sie übernehmen z.B. Such- oder Abfragealgorithmen. Als Beispiel wurden die geometrischen Beschreibungen für einen Ellipsenbogen, eine Ellipsoidoberfläche, ein Ellipsoid und einen Quader implementiert.
- Zeiger auf die zugehörigen **topologischen Elemente** werden i.d.R. in einer doppelt verketteten Liste der C-Standardlibrary (*list*) am geometrischen Objekt gespeichert.

- Jedes Topologieelement erhält eine Verzeigerung zum hierarchisch höherliegenden Topologieobjekt, eine Verzeigerung zum zugehörigen geometrischem Objekt und eine *list* der Verzeigerungen zu den hierarchisch tieferliegenden Topologieelementen.
- Die Modell-Klasse (**MODEL**) enthält die *maps* aller im Modell enthaltenen Geometrieobjekte. Das MODEL hat somit direkten Zugriff auf alle Element und kann sie verwalten.
- Mehrere Entwicklungsstände eines Modells werden in einer *map* gespeichert.
- Es wurden **Operatoren** vorgestellt, welche die Basis aller Operationen auf der Datenstruktur sind. Sie organisieren jede Modifikation der Datenstruktur. Da sie Zugriff auf alle geometrischen und topologischen Elemente haben müssen, wurden sie als Member der Modell-Klasse implementiert.
- Die Operatoren *make* (Erstellen), *kill* (Löschen), *insert* (logisches Einfügen eines Volumens in ein anderes) und *glue* (Vereinigen von zwei geometrischen Elementen) wurden implementiert.
- Da das Erstellen komplexer Geometrien eine große Fehlerquelle darstellt, wurde untersucht inwieweit **boolesche Operationen** auf der Datenstruktur zu realisieren sind. Es wurden die booleschen Grundoperationen Vereinigung \cup , Intersektion \cap und Differenz / erläutert.
- Die **Überschneidungsinformation** ist die Grundlage für boolesche Operationen. Dazu wurden vier Möglichkeiten der Lage zweier Körper zueinander (*außerhalb*, *berührend*, *schneidend* und *innerhalb*) erläutert.
- Zur effizienteren Nutzung der notwendigen Überschneidungsalgorithmen werden **bounding boxes** eingesetzt. Typische Formen von bounding boxes sind z.B. ein zum Koordinatensystem paralleler Quader, ein zum Koordinatensystem nicht paralleler Quader, eine Kugel oder ein Polyeder.
- Eine robuste, fehlerfreie Umsetzung auf BRep-Strukturen ist nur schwer zu realisieren. Aus diesem Grund wird vorgeschlagen, diese Mengenoperation zunächst nur für die CSG-Primitive der Hybridstruktur zu implementieren.

- Die *Slang* -Kommandos, welche die Datenstruktur betreffen, werden in der Kommandogruppe „RED“ zusammengefasst. Es wurden die Kommandos **red create**, **red delete**, **red info**, **red modify**, **red boolean** und **red export** implementiert.
- Die Funktionsfähigkeit der *make*- und *insert*- Operatoren wurde anhand eines Betonmodells nachgewiesen. Die *glue*- und *kill*-Operatoren wurden in weiteren Beispielen verifiziert.
- Durch ein Interface zum Vernetzer *gmsh* ist es möglich, die entstandene Geometrie vernetzen zu lassen.

Es werden folgende Möglichkeiten für die Weiterentwicklung der vorliegenden Datenstruktur vorgeschlagen:

1. Interface zu Eingabeformaten (z.B. *.dxf)
2. Implementierung verschiedener Geometrieformulierungen (B-Spline, Nurbs, Funktionsbeschreibungen usw.)
3. Interface zu verschiedenen Vernetzern (RED \rightarrow Vernetzer \rightarrow *Slang*)
4. Erweiterung der Funktionalität der INFO-Klassen (Material- und Lastinformationen)
5. Überschneidungsalgorithmen (boolesche Operationen)
6. Postprocessing (FE-Verschiebungsfigur \rightarrow RED)