

Erfahrungsbericht zur Entwicklung und Anwendung der Methode der Volumenmodellierung

H. Bröker¹, V. Nübel¹, M. Bernreuther², B. Firmenich³

Februar 2002

1. Einleitung (B. Firmenich)

Professoren und Mitarbeiter der Bauinformatik-Lehrstühle der Technischen Universität München, der Universität Stuttgart und der Bauhaus-Universität Weimar trafen sich am 24. und 25.11.2000 in München zu einem Informationsaustausch über ihre Forschungsaktivitäten. Ziel war es, gegenseitig von den Erkenntnissen und Ergebnissen der anderen zu profitieren.

Als eine gemeinsame Forschungsaktivität aller drei Lehrstühle wurde die *Methode der Volumenmodellierung* identifiziert. Der vorliegende Aufsatz entstand in Kooperation der drei beteiligten Lehrstühle, die sich in der Vergangenheit bereits intensiv mit der Entwicklung und der Anwendung der Methode der Volumenmodellierung im Bauwesen beschäftigt haben. Ziel ist es, die bereits vorliegenden Erfahrungen der beteiligten Lehrstühle auf diesem Gebiet zu dokumentieren.

Im Planungsprozess dient die Geometrie des Bauwerks einer Vielzahl der Beteiligten als Grundlage für die Bearbeitung der eigenen Aufgabenstellungen. Für das dabei erstellte geometrische Modell des Bauwerks sind zwei vollständig unterschiedliche Lösungsansätze bekannt.

Beim traditionellen *zeichnungsorientierten Ansatz* sind die Bauwerksgeometrie (und weitere Bauwerksinformationen) in Zeichnungen dargestellt. Technische Zeichnungen sind zwar das klassische, bewährte und unverzichtbare Medium zur Darstellung und Vermittlung von Ingenieurlösungen. Sie repräsentieren jedoch immer nur einen Teilaspekt des Gesamtzusammenhangs als zweidimensionale Sicht auf ein dreidimensionales Bauwerk. CAE-Software, die ausschließlich den zeichnungsorientierten Ansatz verfolgt, hat daher insbesondere folgende Schwachpunkte:

- Das System kann keine Hilfestellung zur Konsistenzerhaltung der Planungsunterlagen geben, da ihm der Gesamtzusammenhang nicht bekannt ist.
- Zeichnungen im Bauwesen sind nur in begrenztem Umfang auszuwerten, da die korrekte physikalische Repräsentation der Objekte sich oft von der normengerechten symbolischen Darstellung einer Zeichnung unterscheidet.
- Aus der Menge der Zeichnungen eines Bauwerks ist die Bauwerksgeometrie nicht vollständig abzuleiten.

¹ Technische Universität München, Bauinformatik

² Universität Stuttgart, Informationsverarbeitung im konstruktiven Ingenieurbau

³ Bauhaus-Universität Weimar, Informatik im Bauwesen

Im Gegensatz zum zeichnungsorientierten Ansatz ist das Ziel des *modellorientierten Ansatzes* die vollständige dreidimensionale Repräsentation der Geometrie eines Bauwerks. Die mathematisch eindeutige und vollständige geometrische Repräsentation der Objekte des Modells kann nur auf der Basis eines leistungsfähigen Volumenmodellierers gelöst werden. Alle Methoden des Modellierers sichern die Gültigkeit ihrer Ergebnisse zu. Damit ist die Voraussetzung für eine konsistente geometrische Modellierung geschaffen.

Der vorliegende Aufsatz ist wie folgt gegliedert. In *Kapitel 2* wird die Methode der Volumenmodellierung allgemein beschrieben. Die für das Bauwesen wesentlichen Typen von Volumenmodellierern (BRep-Modellierer, CSG-Modellierer und räumliche Zellmodellierer) werden bezüglich ihrer Datenstrukturen klassifiziert, beschrieben und hinsichtlich der Anwendbarkeit gegeneinander abgegrenzt. Wesentliche Operationen dieser Modellierer werden vorgestellt. *Kapitel 3* befasst sich mit der Programmierschnittstelle eines konkreten BRep-Volumenmodellierers. Von einem Volumenmodellierer fordert man stets die Konsistenz der Daten. Basis für die topologische Konsistenz eines Polyeders ist der Eulersche Polyedersatz bzw. dessen Erweiterung durch Poincare. In *Kapitel 4* wird ein Volumenmodellierer beschrieben, der die Erzeugung und Manipulation von Volumenmodellen durch sogenannte Euler-Operatoren unterstützt und damit jederzeit die topologische Konsistenz dieser Modelle gewährleistet. *Kapitel 5* behandelt einen kommerziellen Volumenmodellierer in einer kommerziellen CAD-Umgebung, wobei der Schwerpunkt auf der Anwendung und der Programmierung liegt. Der beschriebene Modellierer ist sehr leistungsfähig, da die Modelle durch Freiformflächen begrenzt sein können. Damit kann die komplexe Geometrie aktueller Bauprojekte, die längst nicht mehr ausschließlich als Polyeder zu abstrahieren ist, im Rechner vollständig abgebildet werden.

2. Methode der Volumenmodellierung (B. Firmenich)

Dieses Kapitel basiert teilweise auf „Joachim-F. Grätz; Handbuch der 3D-CAD-Technik“ sowie dem „Handbuch zum *unicad-Programmier-Paket*; Hochtief“.

3D-Modelle: Der Begriff 3D-Modell sagt sehr wenig über die Leistungsfähigkeit des zugehörigen Modellierers aus. Der Grund dafür ist, dass hier mit 3D lediglich eine Eigenschaft des geometrischen Raums gemeint ist, in dem die Modelle angeordnet sind. Alle 3D-Modelle liegen demzufolge im dreidimensionalen Raum. Geometrische Modelle beschreiben stets eine Punktmenge einer gewissen geometrischen Dimension. Zur Unterscheidung der geometrischen Dimension dieser Punktmenge gliedert man 3D-Modelle in die Klassen *Drahtmodell*, *Flächenmodell* und *Volumenmodell*. Nachfolgend werden alle drei Modellierertypen kurz beschrieben.

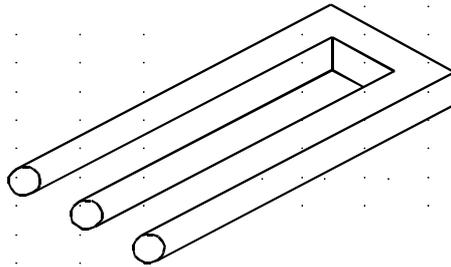


Bild 2.1: Modellierung eines „Nonsense“-Objekts als Drahtmodell

Drahtmodelle oder Kantenmodelle sind die einfachste Form der 3D-Modelle. Es werden eindimensionale Punktmenge – also Kanten – beschrieben. Ein Beispiel für eine geeignete Anwendung eines Drahtmodells sind Rohrleitungsisometrien, da es bei dieser Anwendung nur auf die Repräsentation der eindimensionalen Rohrleitungsachse ankommt. Soll ein Drahtmodell für die Repräsentation von zwei- oder dreidimensionalen Objekten verwendet werden, so kann der Modellierer keine Hilfestellung geben, da ihm ein Zusammenhang zwischen den *modellierten* Kanten und den *gedachten* Flächen oder Körpern nicht bekannt ist. So können beispielsweise mehrdeutige (gedachte) Objekte analog Bild 2.1 erzeugt werden.

Flächenmodelle beschreiben zweidimensionale Punktmenge (Flächen). In der Regel besteht der geometrische Bereich eines Flächenmodellierers aus aufwändigen Flächenformen. Beispiele für geeignete Anwendungen sind die Modellierung der Außenhaut von Fahrzeugen oder die Modellierung einer Geländeoberfläche. Soll ein Flächenmodell jedoch für die Repräsentation von dreidimensionalen Objekten verwendet werden, so kann der Modellierer nur begrenzt Hilfestellung geben, da ihm ein Zusammenhang zwischen den Flächen des Modells und dem *gedachten* Körper nicht bekannt ist. Mit einem Flächenmodellierer erzeugte „Körper“ können ungültig sein (Bild 2.2). Auswertungen des „Körpers“ wie beispielsweise die Ermittlung des Volumens sind nicht möglich. Allerdings kann ein solcher „Körper“ – sofern er vom Nutzer konsistent modelliert wurde – als Flächenmodell korrekt dargestellt werden.

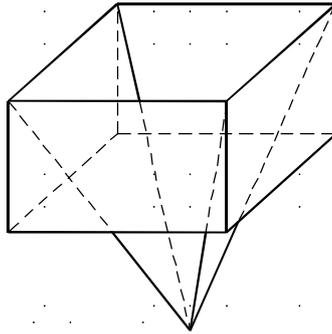


Bild 2.2: Modellierung eines ungültigen „Körpers“ mit einem Flächenmodellierer

Volumenmodelle beschreiben dreidimensionale Punktmenge (Körper) im dreidimensionalen Raum. Ein Beispiel für eine geeignete Anwendung ist die Modellierung der Baustruktur eines Bauwerks. Von einem Volumenmodell erwartet man, dass die modellierten Körper stets konsistent sind und dass alle darauf angewendeten Operationen korrekte Ergebnisse liefern. So können beispielsweise die in Bild 2.1 und 2.2 dargestellten Objekte nicht mit einem Volumenmodellierer erzeugt werden.

2.1 Methoden von Volumenmodellierern

Volumenmodellierung ist die Methode, die die geometrische Form von Körpern mathematisch genau beschreibt. Auf die Volumenmodelle angewendete Operationen liefern korrekte Ergebnisse.

Die Methoden eines Volumenmodellierers lassen sich gliedern in

- Methoden zur Generierung von Körpern
- Methoden zur Manipulation der Form eines Körpers
- Methoden zur Darstellung von Körpern
- Methoden zur Auswertung von Körpern

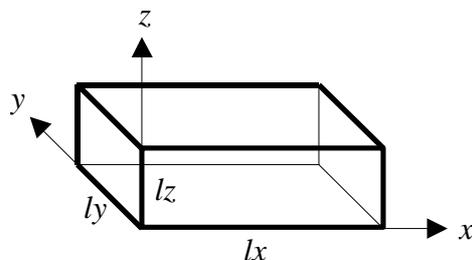


Bild 2.3: Parametrisierung eines Quaders

Generierung durch Grundkörper: Ein Volumenmodellierer verfügt in der Regel über eine Bibliothek mit parametrisierbaren allgemeinen Grundkörpern wie Quader, Zylinder oder Kegel. Es können auch spezielle Grundkörper für einen bestimmten Anwendungsbereich zur Verfügung gestellt werden. So kann eine Grundkörperbibliothek für die Tragwerksplanung beispielsweise häufig verwendete Körper wie Konsolen oder Maschinenfundamente beinhalten. Der Anwender wählt einen Grundkörper aus der Bibliothek aus, gibt die entsprechenden Parameter – beispielsweise Länge, Breite und Höhe für den in Bild 2.3 dargestellten Quader – ein und platziert den Körper im dreidimensionalen Raum.

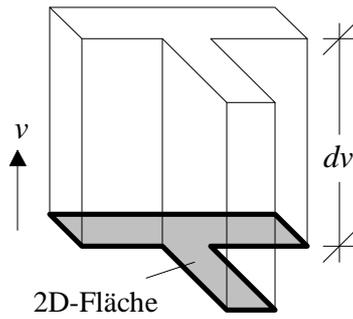


Bild 2.4: Generierung eines Körpers durch *Translatorisches Sweeping einer ebenen Fläche*

Generierung durch Sweeping ebener Flächen: Ein Körper wird aus einer zweidimensionalen Fläche, die entlang eines einzugebenden Vektors v um den einzugebenden Betrag dv translatorisch bewegt wird, generiert (Bild 2.4). Mit dieser Generierungstechnik können Körper sehr einfach aus zweidimensionalen Flächen erzeugt werden. Beispielsweise können so aus den geschnittenen vertikalen Bauteilen eines zweidimensionalen Grundrisses dreidimensionale Körper generiert werden.

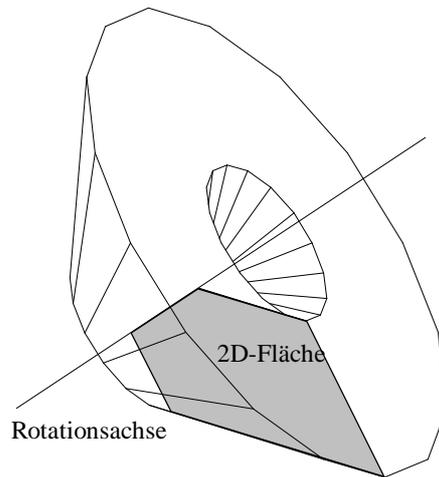


Bild 2.5: Generierung eines Körpers durch *Rotatorisches Sweeping einer ebenen Fläche*

Ebenfalls bekannt ist das rotatorische Sweeping (Bild 2.5). Bei dieser Generierungstechnik werden Körper durch Rotation ebener Flächen um eine Rotationsachse definiert.

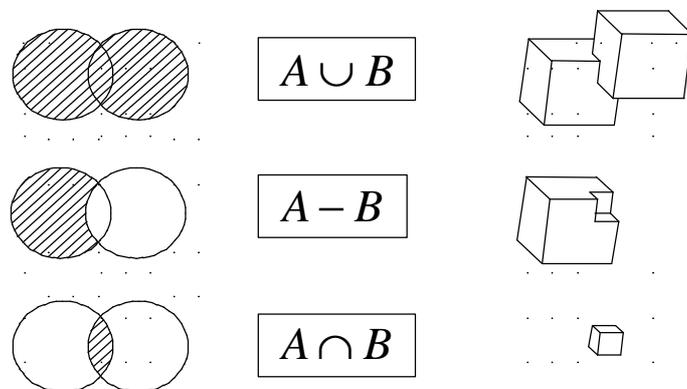


Bild 2.6: Anwendung mengentheoretischer Operationen

Anwendung mengentheoretischer Operationen: Aus zwei beliebig komplexen Körpern kann mit Hilfe der mengentheoretischen Operationen *Vereinigung*, *Differenz* und *Durchschnitt* ein resultierender Körper berechnet werden (Bild 2.6). Diese Operationen erlauben die Erzeugung komplexer Körper mit einem Minimum an Eingabeaufwand. Allein die Verknüpfung zweier Quader liefert abhängig von Mengenoperator, relativer Lage zueinander und absoluten Abmessungen der Körper eine Vielzahl von Ergebniskörpern unterschiedlicher Form. Die Umsetzung erfordert jedoch die Entwicklung äußerst komplexer und numerisch robuster Algorithmen, die bei einer Anwendung auf komplexe Körper auch eine lange Rechenzeit haben können. Eine mögliche Anwendung mengentheoretischer Operationen im Bauwesen ist beispielsweise die Subtraktion der Öffnungen von einer Wand oder die Vereinigung einer Stütze mit einer Konsole.

Weitere Operationen: Es sind eine Menge weiterer Methoden zur Manipulation von Volumenmodellen bekannt. Darunter fällt beispielsweise die Koordinatentransformation, die unterschiedliche Auswirkungen auf die Form der Körper hat: Während nämlich bei einer Translation oder Rotation die Körper ihre Form behalten ändert sich diese bei einer Skalierung oder Spiegelung.

Darüber hinaus sind lokale Manipulationen einzelner Flächen, Kanten oder Vertices bekannt. Beispiele dafür sind das Fasen einer Kante oder das Verschieben eines Vertex eines Körpers. Lokale Manipulationen sind immer starken Restriktionen unterworfen. Sie erfordern vom Nutzer ein gutes Verständnis der Zusammenhänge.

Darstellung: Die Darstellung dreidimensionaler Strukturen ist ein sehr wichtiger Aspekt der Volumenmodellierung. Erst eine korrekte Visualisierung ermöglicht das Erfassen und Beurteilen der rechnerinternen Volumenmodelle. Die bekannten Visualisierungsmethoden erstrecken sich auf ein weites Feld: Es beginnt in der einfachsten Form mit der Darstellung der Modelle als Drahtmodell, setzt sich fort mit Hidden-Line- und Hidden-Surface-Darstellungen und endet bei enorm aufwendigen Verfahren zur Erzielung fotorealistischer Bilder.

Man unterscheidet zwischen bildschirm- und geometrieorientierten Visualisierungsmethoden. Bildschirmorientierte Verfahren liefern Daten im jeweils installierten Grafiksystem des Computers. Prinzipiell kann jedes Visualisierungsverfahren bildschirmorientiert implementiert werden – aufwändige fotorealistische Methoden sind ausschließlich schirmorientiert zu implementieren. Oftmals werden die Berechnungen von leistungsfähigen Grafikprozessoren unterstützt. Der Wert der bildschirmorientierten Verfahren bei der Darstellung dreidimensionaler Geometrie in technischen Zeichnungen ist sehr fragwürdig. Eine Speicherung der Visualisierungsdaten als zweidimensionale Geometrie oder die Ausgabe der Vektoren auf Plotter ist praktisch nicht möglich. Darüber hinaus können die Bilddaten nicht den Modelldaten zugeordnet werden. Oft ist eine strichlierte Darstellung der unsichtbaren Kanten nicht möglich.

Bei den geometrieorientierten Verfahren wird ein Körper in der Regel mit der Hidden-Line-Methode (Eliminierung verdeckter Kanten) dargestellt. Die Hidden-Line-Methode klassifiziert alle hinter einer oder mehreren Flächen liegenden Kanten oder Kanteile als verdeckt und alle übrigen Kanten als sichtbar. Die berechneten Bilddaten können mit einer hohen Genauigkeit gespeichert werden. Unsichtbare Kanten können bei Bedarf strichliert dargestellt werden.

Modellauswertung: Volumenmodelle erlauben vielfältige Auswertungen. So kann ein Volumenmodellierer Algorithmen zur Ermittlung von Volumen, Oberfläche, Schwerpunkt oder Massenträgheitsmoment eines Körpers zur Verfügung stellen. Andere Methoden führen geometrische Lagetests zwischen einem Körper und anderen geometrischen Objekten durch. So kann beispielsweise durch den Lagetest „Körper – Linie“ festgestellt werden, ob ein Bewehrungsstab vollständig innerhalb eines Bauteils liegt.

2.2 Datenstrukturen von Volumenmodellen

Volumenmodellierer können bezüglich ihrer Datenstrukturen klassifiziert werden. Es wird unterschieden zwischen ausgewerteten (BRep- und Zellmodellierer) und nicht-ausgewerteten Modellen (CSG-Modellierer). Die Eigenschaften dieser Modellierer werden nachfolgend beschrieben.

2.2.1 Ausgewertete Modelle

Die nachfolgend beschriebenen Volumenmodellierer heißen ausgewertet, da die Operationen ad-hoc auf die Volumenmodelle ausgeführt werden. Die angewendeten Operationen werden nicht in der Datenstruktur gespeichert.

BRep-Modellierer (Boundary Representation Modellierer) haben die gemeinsame Eigenschaft, dass ein Körper durch eine Repräsentation seiner orientierten Oberfläche im Rechner abgebildet wird. Es sind verschiedene Datenstrukturen von BRep-Modellierern bekannt. Generell ist zu unterscheiden zwischen *geometrischen* und *topologischen* Daten.

Zu den geometrischen Daten zählen Flächen, Kanten und Punkte. Diese Daten sind jedoch genauso in einem Flächenmodell vorhanden. Erst die topologischen Daten eines BRep-Modells verknüpfen die geometrischen Daten dergestalt, dass der Körper über die orientierte Oberfläche eindeutig definiert wird.

Die Leistungsfähigkeit eines Volumenmodellierers lässt sich in erster Linie durch die unterstützten Oberflächentypen beurteilen. Dieser Punkt ist auch für den Einsatz im Bauwesen entscheidend, da heutzutage Bauteile mit gekrümmter Oberfläche keine Besonderheit mehr darstellen. Wenn ein Modellierer die real vorhandenen Flächenformen nicht verarbeiten kann, so müssen diese durch die unterstützten Flächentypen des Modellierers angenähert werden. Dieser Schritt ist jedoch immer mit einem Verlust von Genauigkeit verbunden (Facettierung) und erfordert in der Regel entsprechende lokale Lösungen in der Anwendungssoftware.

Räumliche Zellmodellierer: Das Datenmodell dieser Volumenmodellierer besteht aus einer Menge räumlicher Zellen (Voxel), die im dreidimensionalen Raum angeordnet sind und die einen Körper annähern. Jede Zelle hat die Form eines Würfels. Da alle Zellen eine einheitliche Größe haben, lassen sie sich in einem räumlichen Gitter anordnen. Jede Zelle hat die Eigenschaft ihres Materials gespeichert – wobei Zellen außerhalb eines Körpers die Materialeigenschaft „Luft“ haben. Mit Hilfe dieses Datenmodells kann auf sehr einfache Art und Weise ein Körper beschrieben werden. Die bereits beschriebenen Operationen eines Volumenmodellierers können auf der Basis gleich großer Zellen eines Gitters sehr einfach umgesetzt werden. Von Nachteil ist die große Datenmenge sowie die fehlende Definition der geometrischen Typen Fläche, Kante und Punkt.

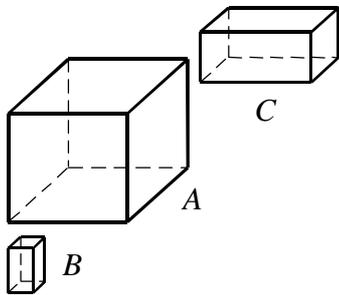
Das Problem der Datenmenge lässt sich effizient auf der Basis der Datenstruktur eines Octree lösen. Ein Octree besteht aus einer hierarchischen Zellmodellstruktur, die mit Hilfe eines Subdivisionsmechanismus‘ den Speicherplatz effizient nutzt. Die Octree-Repräsentation beruht auf der rekursiven Zellteilung eines Würfels. Liegt der Würfel vollständig außerhalb oder vollständig innerhalb des anzunähernden Körpers, so wird keine weitere Unterteilung vorgenommen. Kollidiert der Würfel mit der Oberfläche des Körpers, so lässt sich der Würfel in acht gleich große Subwürfel unterteilen. Für jeden der acht Subwürfel wird das beschriebene Verfahren so lange fortgesetzt, bis die vorgegebene Genauigkeit der Annäherung an den Körper erreicht ist.

2.2.2 Nicht-ausgewertete Modelle

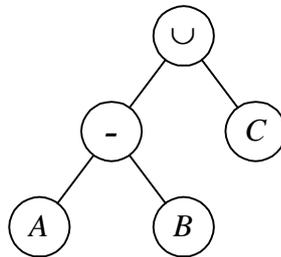
CSG-Modellierer: Ein vollständig anderer Ansatz, Körper mit einem Rechner zu repräsentieren, wird von Volumenmodellierern des Typs Constructive Solid Geometry verfolgt. CSG-Modellierer, die oft auch als operatives Volumenmodell bezeichnet werden, legen das rechnerinterne Modell durch die definierenden Volumengrundkörper sowie die auf diese Körper wirkenden Basisoperationen fest. Damit ist die Entstehungsgeschichte eines Körpers Bestandteil der Datenstruktur.

Für die Datenhaltung wird eine baumartige Datenstruktur gewählt. Die Blätter des Baumes sind Volumenprimitive, die inneren Knoten repräsentieren Operatoren, die auf die „Kinder“ des Knotens (Operanden) angewendet werden. Der oberste Knoten (Wurzel) repräsentiert den modellierten Körper. Die Datenstruktur eines CSG-Baums entspricht einem arithmetischen Ausdruck. Beispielsweise entspricht der in Bild 2.7 dargestellte CSG-Baum dem arithmetischen Ausdruck $(A - B) \cup C$.

Grundkörper:



CSG-Baum:



Evaluierung:

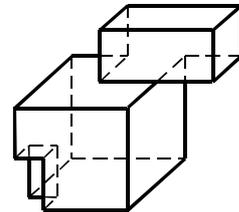


Bild 2.7: Datenstruktur eines CSG-Modellierers

CSG-Datenstrukturen sind in der Regel sehr kompakt, da zur Erzeugung eines komplexen Körpers oft nur wenige parametrisierte Grundkörper und die darauf operierenden Methoden erforderlich sind. Allerdings ist eine Auswertung des resultierenden Körpers allein auf der CSG-Datenstruktur nur in wenigen Fällen möglich. Die vollständige Evaluierung eines CSG-Modells ist erst durch die Überführung in ein ausgewertetes Volumenmodell (BRep- oder Zellmodell) möglich.

Die Evaluierung eines großen CSG-Modells kann sehr aufwändig sein, da bei der geringsten Änderung der resultierende Körper vollständig neu zu berechnen ist. Jeder Knoten eines CSG-Baums repräsentiert einen Körper. Die Evaluierung kann beschleunigt werden, indem bei einigen CSG-Knoten die ausgewerteten Körper – beispielsweise als BRep- oder Zellmodell - gespeichert werden. Letztlich handelt es sich hierbei jedoch nur um ein „Cachen“ von Zwischenergebnissen.

3. Beispiel einer Klassenbibliothek zur Volumenmodellierung (H. Bröker und V. Nübel)

Nachdem im vorangegangenen Kapitel Grundlagen der Volumenmodellierung dargelegt wurden, soll nun eine Klassenbibliothek zur Volumenmodellierung betrachtet werden. Als Beispiel dient die Klassenhierarchie des kommerziellen Geometriemodellierkerns ACIS. Bevor jedoch speziell auf die Klassenstruktur eingegangen wird, wird ein kurzer Überblick über die Entstehungsgeschichte sowie über den allgemeinen Aufbau von ACIS gegeben. Die nachfolgenden Ausführungen orientieren sich dabei an den Handbüchern ACIS 5.0 sowie deren Homepage (<http://www.spatial.com>). Derzeit bildet ACIS 7.0 die aktuelle Version des Geometriemodellierkerns.

3.1 Allgemeines

ACIS wurde 1989 als erster objektorientierter in C++ geschriebener 3D Geometriemodellierkern auf dem internationalen Markt eingeführt. Entwickelt wurde er von den drei Professoren der Cambridge University in England Alan Grayer, Charles Lang und Ian Brad. Die weltweite Vermarktung übernahm die Firma Spatial Technology mit Sitz in Boulder (Colorado – U.S.A). Aus den jeweiligen Anfangsbuchstaben geht daher auch der Name ACIS hervor. Mittlerweile (Juli 2000) gehört ACIS der Firma Dessault an, die u.a. das CAD-System CATIA vermarkten.

Mit der Einführung von ACIS wurde den Softwareentwicklern erstmals eine Möglichkeit geboten, aufsetzend auf umfassende 3D Modellierungsfunktionalität eigene innovative und leistungsfähige Anwendungsprogramme zu entwickeln. Dies hat zur Folge, dass ACIS derzeit die Grundlage vieler CAD/CAM/CAE-Systeme, wie z.B. von AutoCAD der Firma Autodesk bildet. Daneben besitzen viele angesehenen Herstellerfirmen aus den verschiedensten Bereichen, wie z.B. Ford Motor Company, Sony Corporation, Sharp Corporation, Toshiba Corporation usw. eine ACIS-Lizenz für interne Anwendungszwecke. Aus diesem Grund darf man sicherlich zu Recht behaupten, dass es sich bei ACIS um den in der Welt wohl anerkanntesten 3D Geometriemodellierkern handelt. Die Vorteile von ACIS, auf der Basis eines Brep-Modellierers, liegen zum einen in einer umfassenden Speicherung relevanter Daten, sowie in der Exaktheit mit der die Form eines Körpers beschrieben wird. Die Genauigkeit der numerisch berechneten Werte, d.h. das Verhältnis zwischen dem größten und kleinsten Wert, beträgt 10^{-6} . Im Vergleich dazu weisen ältere Systeme (2D) nur eine Genauigkeit von 10^{-1} auf. Des weiteren bietet ACIS die Möglichkeit aus Volumenmodellen auch Flächen- oder Drahtmodelle abzuleiten sowie eine Darstellung von manifold und non-manifold Körpern.

Wie bereits eingangs erwähnt, ist ACIS objektorientiert in der Programmiersprache C++ geschrieben, so dass C++-Klassen und Methoden zur Entwicklung eigener Applikationen zur Verfügung stehen. Dabei unterscheidet man prinzipiell zwischen zwei verschiedenen Zugriffsmöglichkeiten auf den Geometrie kern. Dies ist zum einen das *Application Procedural Interface* (API) und zum anderen das *Direct Object Interface*. Während unter Benutzung des *Direct Object Interfaces* einzelne Klassen abgeleitet werden können und somit die volle Funktionalität der objektorientierten Programmierung zur Verfügung steht, bietet das *Application Procedural Interface* die Möglichkeit, vorhandene Funktionen direkt aus dem Anwenderprogramm heraus aufzurufen. Aufgrund dieser einfachen Handhabung stellen sie die meist genutzte Schnittstelle zwischen den Anwendungsprogrammen und ACIS dar. Mit Hilfe der API-Funktionen können beispielsweise Geometrien erzeugt, manipuliert und evaluiert werden. Weiterhin existieren Funktionen auf der Basis von GUI-Funktionalitäten, so dass grafische Interaktionen vom Benutzer definiert und ausgewertet werden können.

Neben dem eigentlichen Geometriemodellierkern ACIS 3D Toolkit besteht die ACIS Produktpalette von Spatial aber noch in weiteren Applikationen. Die sogenannten optionalen Husks erweitern die Kernfunktionalität um verschiedene Bereiche. Mittlerweile sind mehr als 15 weitere Husks erhältlich, wobei nachfolgend nur einige der wichtigsten aufgeführt werden, um einen Einblick in die Vielfalt zu geben.

- *Advanced Rendering Husk*: Diese Applikation bietet die Möglichkeit der photorealistischen Darstellung eines ACIS 3D Geometriemodells.
- *Healing Husk*: Im Hinblick auf Austauschgeometrien ist es sehr häufig notwendig das Modell zu überarbeiten und Ungenauigkeitstoleranzen auszugleichen.
- *IGES Translator Husk*: Diese Applikation ermöglicht den bidirektionalen Austausch von Daten im IGES und ACIS-Format (<*>.sat bzw. <*>.sab)

Eine weitere nützliche Applikation stellt das Scheme Application Interface dar. Dabei handelt es sich um eine Umgebung in der Funktionen nach der Interpretersprache Scheme entsprechend der API-Funktionen zur Verfügung stehen. Mit Hilfe dieser Funktionalität können recht einfach die API Funktionen getestet und somit Fehler leicht verifiziert werden.

3.2 Klassenhierarchie

Entsprechend der Unterscheidung zwischen geometrischen und topologischen Informationen bei einem Brep-Modellierer werden diesbezüglich die Klassen zunächst in zwei Gruppen (Geometrie- und Topologieklassen) gliedern.

Zusätzlich erweist es sich jedoch als vorteilhaft zwei weitere Klassengruppen einzuführen. Dies sind mathematische Klassen (TRANSFORM und POSITION), die zur Definition der allgemeinen Lage des Körpers im euklidischen Vektorraum benötigt werden, sowie die ATTRIBUT-Klasse, die es dem Benutzer ermöglicht zu den Objekten eigene nicht geometrische und topologische Informationen zu speichern.

3.2.1 Geometrie-Klassen

Grundsätzlich ist bei der Repräsentation der Geometrie eines Objektes zwischen der eigentlichen mathematischen Beschreibung, die nachfolgend, sowie auch in ACIS als *konstruktive Geometrie* bezeichnet wird und einer *abstrakten Geometrie* zu unterscheiden. Zu den Klassen der abstrakten Geometrie zählen APOINT, CURVE, SURFACE und PCURVE, wobei von zentraler Bedeutung die ersten drei Klassen sind. Von diesen Klassen werden alle weiteren Geometrieklassen ohne Verwendung privater Membervariablen abgeleitet, so dass auf die Daten direkt zugegriffen werden kann. Sie besitzen virtuelle Funktionen, wie beispielsweise zur Transformation, Skalierung und Parametrisierung, deren korrekte Operationen erst anhand der Typen der abgeleiteten Klassen ausgewählt werden. Da allein mit diesen Klassen kein geometrisches Objekt erzeugt bzw. beschrieben werden kann, werden diese Klassen mit Großbuchstaben dargestellt. Im Gegensatz dazu werden die Klassen der konstruktiven Geometrie mit kleinen Buchstaben bezeichnet. Hierbei existieren die Klassen *compcurve*, *cone*, *ellipse*, *intcurve*, *meshsurf*, *plane*, *sphere*, *spline*, *straight* und *torus*, die im wesentlichen die mathematische Definition einer allgemeinen Kurve und Fläche beinhalten. Auffällig ist, dass für den Kreis sowie für den Zylinder keine eigenen Klassen existieren. Sie werden über die allgemeine Form als Ellipsen und Kegel generiert. Die Klasse *spline* fasst die Freiformflächen zusammen, wobei das NURBS-Modellierungsverfahren die Basis darstellt.

Die mathematische Beschreibung der geometrischen Elemente erfolgt teilweise analytisch und/oder parametrisch. Die Methoden der jeweiligen Klasse beziehen sich auf die Konstruktion, die Vernichtung, die Modifikation, die Abfrage und die Evaluierung.

Eine Übersicht der Geometrie-Klassen und deren Zusammenhänge ist schematisch in Abbildung 3.1 dargestellt.

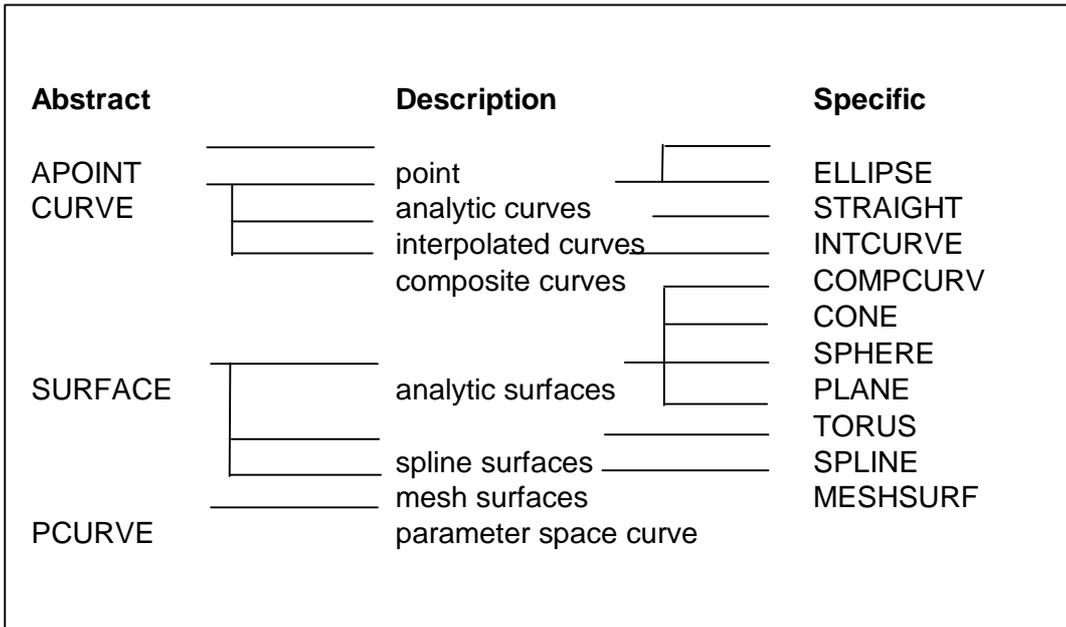


Bild 3.1: Schematischer Aufbau der Geometriebeschreibung – abstrakte und spezifische Geometrie

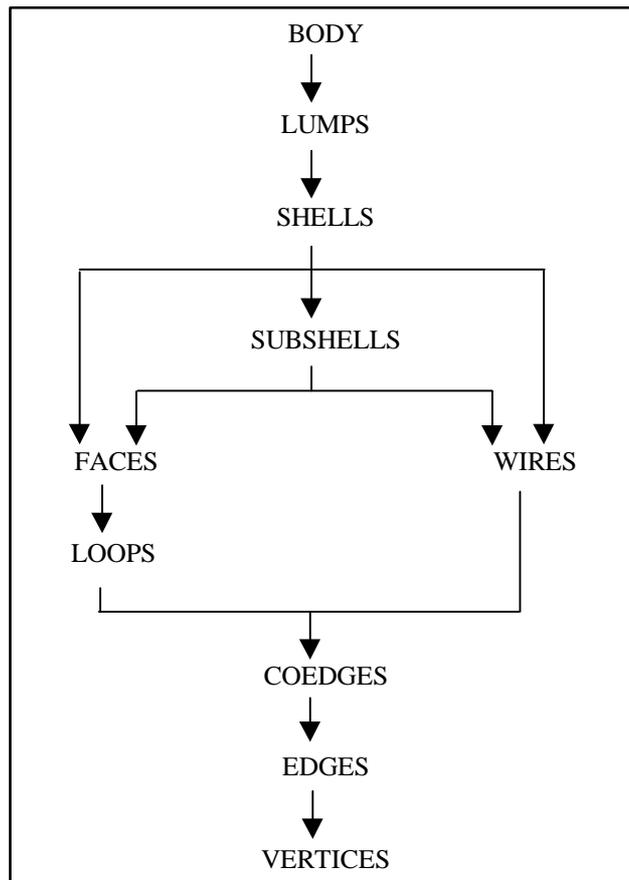


Bild 3.2: Schematischer Aufbau der Topologie in ACIS

3.2.2 Topologie-Klassen

Die Topologie betreffenden Informationen beschäftigen sich mit dem Aufbau eines allgemeinen geometrischen Objekts. Hierbei spielen qualitative Lagebeziehungen der Elemente des Körpers eine Rolle, wobei metrische Größen (Längen, Winkel) keine Berücksichtigung finden. Grundsätzlich findet eine hierarchische Einteilung der Modelltopologie in verschiedene Objekte so statt, dass die Objekte höherer Ordnung mit Hilfe von Objekten niedriger Ordnung beschrieben werden. Der schematische Aufbau der Topologie ist in Abbildung 3.2 dargestellt.

Ausgangspunkt eines allgemeinen geometrischen Objekts bildet der BODY, der sich aus durch die folgenden aufgeführten, ihm untergeordneten Objekte, darstellen lässt. Dies sind zunächst mehrere LUMPS, die wiederum aus SHELLS bestehen und die in SUBSHELLS gegliedert werden können. Die SHELLS bzw. SUBSHELLS ihrerseits können sich dann zum einen aus FACES, deren Randkanten mit Hilfe von LOOPS verwaltet werden, oder zum anderen aus WIRES zusammensetzen. Die darauf folgende Hierarchiestufe bilden die COEDGES denen EDGES zugeordnet werden. Schließlich bilden die VERTICES die letzte Gruppe in der Hierarchie.

Im folgenden werden die einzelnen Klassen der Topologie näher betrachtet, die sozusagen das ‚Herz‘ der Brep-Datenstruktur bilden. An dieser Stelle sei erwähnt, dass diese Klassen grundsätzlich in Großbuchstaben definiert werden.

- **BODY**

Sie stellen die höchste Stufe eines geometrischen Objekts (Entity) in einem ACIS Modell dar. Üblicherweise wird mit einem BODY ein einzelner Volumenkörper assoziiert. Daneben kann er allerdings auch mehrere disjunkte BODIES vereinigen, so dass diese in einer Gruppe zusammengefasst werden können.



Bild 3.3: Kugel als ein BODY

- **LUMP**

Ein LUMP repräsentiert eine zusammenhängende Region im Raum. Ein LUMP ist ein Menge von zusammenhängender Punkte, wobei dies in 3D, 2D, 1D oder in Kombination auftreten kann. Ein BODY kann sich demzufolge aus einem oder mehreren LUMPS zusammensetzen. Dabei repräsentieren die Punktemengen nicht zusammenhängen die durch die anderen LUMPS in dem BODY existieren.

Ein Beispiel für einen BODY mit acht LUMPS ist in Abbildung 3.4 dargestellt. Er entsteht durch Subtraktion einer Kugel mit einem Radius r für den gilt: $a < r < \sqrt{a^3}$ und einem Würfel mit der Kantenlänge a .

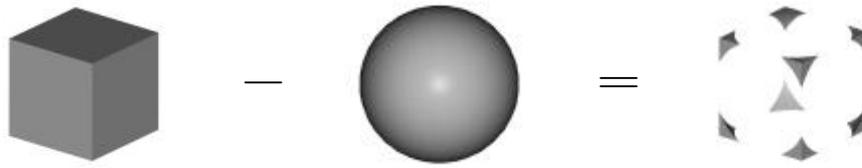


Bild 3.4: BODY mit acht LUMPS

- **SHELL**

Die SHELLS definieren eine Menge zusammenhängender FACES. Die FACES sind dabei über ihre Randkanten bzw. -punkte miteinander verbunden. Beispielsweise bildet ein Körper mit einer baumelnden Fläche eine SHELL im Gegensatz zu einem Volumenkörper mit einem Hohlraum. Dieser wird über zwei SHELLS, einer inneren und einer äußeren SHELL beschrieben.

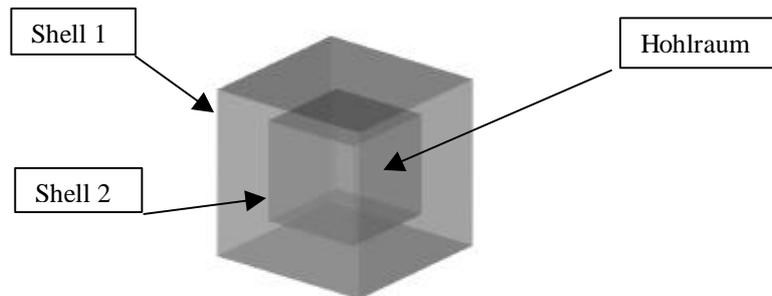


Bild 3.5: BODY – LUMP mit zwei SHELLS

- **SUBSHELL**

Die SUBSHELLS werden hauptsächlich für interne Zwecke von ACIS verwendet. Durch die Möglichkeit die SHELLS in eine Hierarchie von SUBSHELLS zu gliedern, wirkt sich dieses Vorgehen positiv auf die Leistungsfähigkeit aus. Der Zugriff auf diese Klassen durch API-Funktionen wird nicht unterstützt.

- **WIRE**

Ein WIRE wird durch eine Menge zusammenhängender EDGES definiert, die keiner Fläche zugeordnet werden können. In der Regel dienen sie der Repräsentation abstrakter Formen, wie Profile, Konstruktionslinien usw. . Dieses Topologieelement wird vorzugsweise für die Darstellung von Drahtmodellen verwendet.

- **FACE**

Die FACE hat nun einen direkten Bezug zur geometrischen Beschreibung des Objekts. Sie ist ein Teil einer geometrischen Flächendefinition im dreidimensionalen euklidischen Vektorraum, also das zweidimensionale Analogon zu einem BODY. Die Berandung der FACE erfolgt durch ein oder mehrere LOOPS, so dass auch hier das Analogon zu den SHELLS besteht.

Die FACE besitzt eine Orientierung, die der zugrunde liegenden Oberflächenbeschreibung entsprechen kann oder nicht. Stimmen die Richtungen der Normalenvektoren der FACE und der Oberflächenbeschreibung überein, so ist die Orientierung als FORWARD definiert, in dem anderen Fall als REVERSED.

Eine Fläche, die ein Volumen vollständig oder teilweise begrenzt, wird einseitig genannt. Ihr Normalenvektor zeigt vom Volumen weg. Steht eine Fläche jedoch nicht in Zusammenhang mit einem Volumen, so wird sie zweiseitig genannt. Dies trifft ebenfalls für Flächen zu, die in einen Volumenkörper eingebettet sind.

- **LOOP**

Ein LOOP repräsentiert eine zusammenhängende Folge von Randkanten einer Fläche. Die Folge von Randkanten kann dabei geschlossen oder auch offen sein. Ihnen wird jeweils eine Richtung zugewiesen, die in der Regel gegen den Uhrzeigersinn definiert ist, wenn man von außen auf den Volumenkörper schaut.

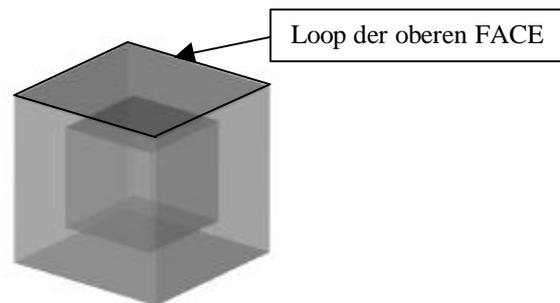


Bild 3.6: LOOP Boundary

- **COEDGE**

Eine COEDGE registriert das Vorkommen einer EDGE in einem LOOP einer FACE. Die Einführung von COEDGES erlaubt das Vorkommen einer EDGE in mehreren FACES, so dass das Modellieren von manifold und/oder non-manifold Objekten ermöglicht wird. Ein LOOP besitzt eine Referenz auf eine COEDGE in diesem LOOP und die COEDGES sind dann untereinander so referenziert, dass mit der Reihenfolge ein Umlaufsinn der zugrunde liegenden Fläche definiert wird. In der Regel ist die Orientierung im mathematischen positiven Sinn bezogen auf den nach außen gerichteten Normalenvektor.

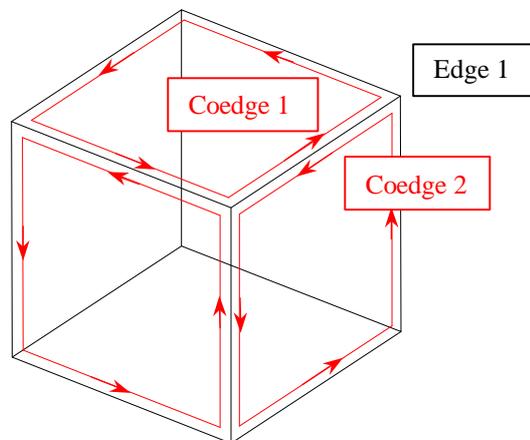


Bild 3.7: LOOP Boundary

- **EDGE**

Der (topologischen) EDGE wird die geometrische Kurvenbeschreibung zugeordnet. Sie ist begrenzt durch Referenzen auf keinen, einen oder mehrere VERTICES. Verweisen alle Referenzen auf einen NULL-Pointer, so handelt es sich um eine geschlossene EDGE. Die der EDGE zugrunde liegende Geometriedefinition ist damit unendlich, z.B. ein Kreis.

Jede EDGE besitzt eine Orientierung, die relativ zu der Richtung der zugrundeliegenden Kurvenbeschreibung definiert wird. Entsprechend der Konvention der Flächen kann die Orientierung FORWARD, also gleichgerichtet oder REVERSED sein

- **VERTEX**

Ein VERTEX begrenzt eine EDGE. In der Regel handelt es sich dabei um einen Eckpunkt einer FACE oder eines WIRES. Eine Referenz auf einen geometrischen Punkt im Raum erlaubt den Zugriff auf die kartesischen Koordinaten der Start- und Endpunkte einer EDGE.

In der nachfolgenden Abbildung sind die Zusammenhänge zwischen den jeweiligen Klassen nochmals detaillierter aufgeführt.

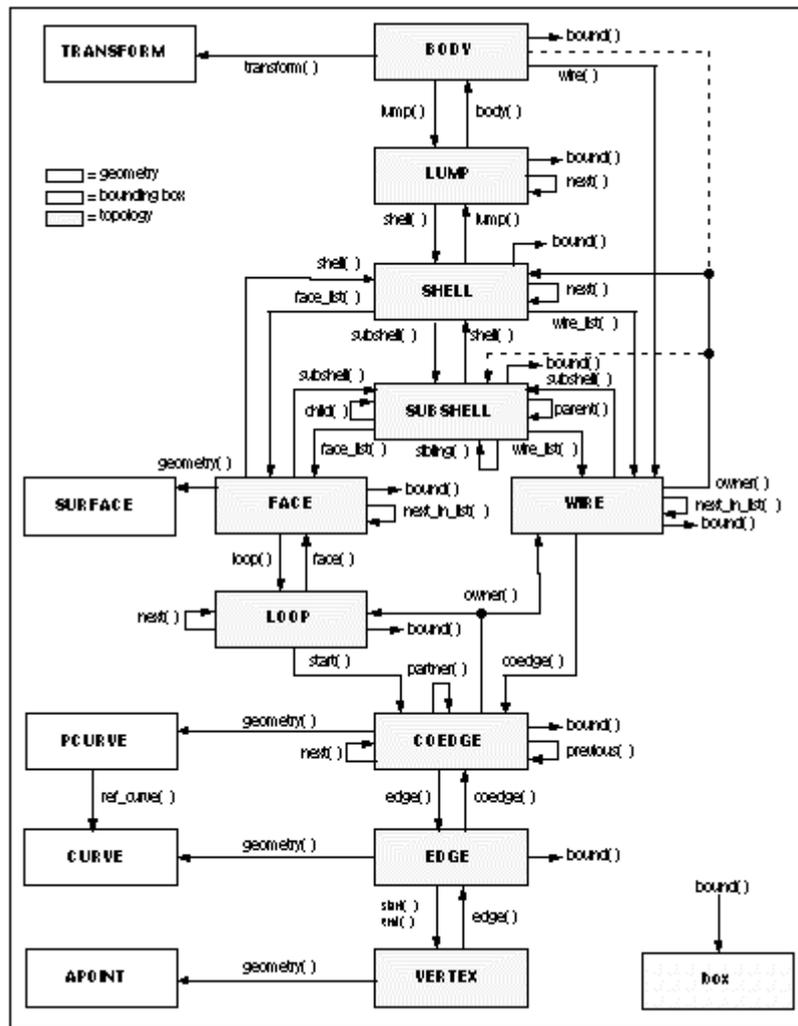


Bild 3.9: Gesamtüberblick über die Klassenverwaltung von ACIS (aus ACIS Technical Overview)

Zusammengefasst werden alle Klassen für die Geometriebeschreibung, die Topologie und die Attribute in einer Klasse ENTITY. Sie stellt die Basisklasse dar, von der alle anderen Klassen abgeleitet werden.

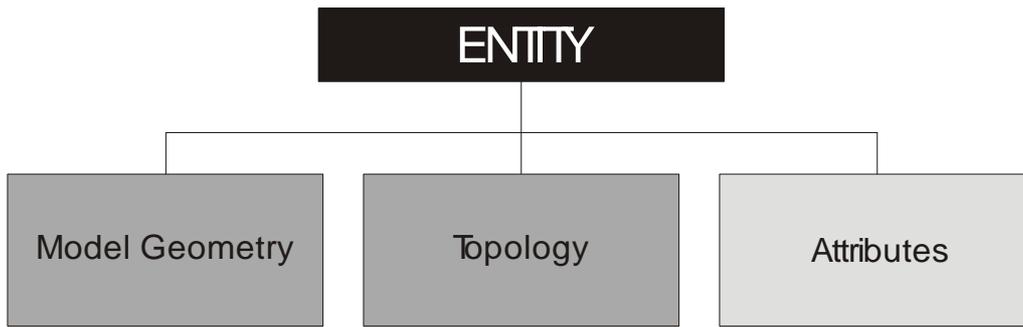


Bild 3.9: Die ENTITY-Klasse als oberste Klasse zur Verwaltung der Geometrie, Topologie und den Attributen

4. Modellieren mit Euler-Operatoren (M. Bernreuther)

4.1 Polyeder

Im Taschenbuch der Mathematik [4.1] wird folgende Definition für ein Polyeder gegeben:

Polyeder Unter einem *Polyeder* versteht man einen Körper, der von Ebenenteilen begrenzt wird.

Nach Jänich [4.3] wird in einer Definition ein Polyeder einem Simplicialen Komplex gleichgesetzt. Ein Simplicialer Komplex ist eine Menge von Simplexen, die sich nicht überschneiden. Ein n -dimensionaler Simplex ist die konvexe Hülle von $n + 1$ Punkten in allgemeiner Lage. Teilmengen dieser Punkte bilden Teilsimplexe niedriger Dimension. Ein n -dimensionaler Simplex enthält $\binom{n+1}{k+1}$ k -dimensionale Teilsimplexe ($0 \leq k < n$) als Seiten. Entsprechend enthält ein dreidimensionaler Simplex (Tetraeder) 4 zweidimensionale Simplexe (Dreiecke), 6 eindimensionale Simplexe (Strecken) und 4 nulldimensionale Simplexe (Punkte). Jedes Polyeder kann in Simplexe zerlegt werden. Dies bezeichnet man als Triangulierung. Bei einer allgemeinen Zerlegung eines Körpers in paarweise disjunkte nichtleere Teilmengen spricht man von einer Zellzerlegung. Im folgenden wird von einem CW^1 -Komplex, ausgegangen, der aus einem topologischen (Hausdorff-) Raum und einer Zellzerlegung besteht. Im Gegensatz zum Simplicialkomplex sind hier allgemeinere Zellen zulässig. n -dimensionale Zellen werden als n -Zellen bezeichnet. Zu jeder n -Zelle muß ein Homöomorphismus zur n -dimensionalen Vollkugel existieren. Die Vereinigung der höchstens $n - 1$ -dimensionalen Zellen läßt sich stetig auf eine $n - 1$ -Sphäre abbilden.

Es existieren fünf reguläre Polyeder, die auch als Platonische Körper bezeichnet werden und schon in der griechischen Antike bekannt waren. Reguläre Polyeder besitzen kongruente, regelmäßige Vielecke als Seitenflächen. In den Ecken stoßen bei diesen Körpern immer die gleiche Anzahl Seitenflächen zusammen. Ein reguläres Polyeder kann als eine 3-Zelle angesehen werden, die sich aus 0-, 1- und 2-Zellen zusammensetzt. Eine 0-Zelle entspricht einem Punkt v , eine 1-Zelle einer Kante e und eine 2-Zelle einer Fläche f .

Betrachtet man die Anzahl der Zellen in Tab. 1, so gilt die

Eulersche Polyederformel

$$v - e + f = 2 \tag{1}$$

| Bezeichnung | Seitenflächen | v | e | f |
|-------------|---------------|----|----|----|
| Tetraeder | Dreiecke | 4 | 6 | 4 |
| Hexaeder | Quadrate | 8 | 12 | 6 |
| Oktaeder | Dreiecke | 6 | 12 | 8 |
| Dodekaeder | Fünfecke | 20 | 30 | 12 |
| Icosaeder | Dreiecke | 12 | 30 | 20 |

Tabelle 4.1: Reguläre Polyeder

Diese Formel gilt auch für die Hülle allgemeiner dreidimensionaler sphärischer Polyeder². Polyederoberflächen können topologisch als planare Graphen $G(V, E)$ dargestellt werden. Auch in diesem Zusammenhang ist (1) bekannt (s. Harary [4.2]). Es ist zu beachten, daß die äußere Fläche um den Graphen hierbei mit berücksichtigt werden muß.

¹ "C" steht für "closure finite" (hüllenendlich) und "W" für "weak topology" (schwache Topologie)

² Diese sind homöomorph zur Sphäre

4.2 Topologische Invarianten

In der Topologie ist für eine als Zellkomplex dargestellte n -dimensionale Mannigfaltigkeit M die Eulersche Charakteristik durch

$$c(M) := \sum_{d=0}^n (-1)^d a_d \quad (2)$$

als topologische Invariante gegeben. a_d steht für die Anzahl der d -Zellen. Ein Vergleich von (2) mit (1) zeigt, daß für den Rand eines regulären Polyeders die Eulersche Charakteristik $c(P_{reg}) = 2$ ist.

Eine genauere Charakterisierung lassen die Betti-Zahlen zu, mit deren Hilfe die Eulersche Charakteristik bestimmt werden kann. Das n -Zahle tupel der Betti-Zahlen ist eine topologische Invariante, die mit der Eulerschen Charakteristik in folgender Weise zusammenhängt (Satz von Poincaré):

$$c(M) := \sum_{d=0}^{\infty} (-1)^d b^d(M) \quad (3)$$

b^d steht für die d -te Betti-Zahl, die als Rang einer Homologiegruppe H_d definiert ist. Anschaulich lassen sich für Polyeder die Betti-Zahlen mit Hilfe der Homotopie bestimmen (s. Boltjanskij/Efremovic [4.4, S. 136]), denn homotope Zyklen sind auch homolog. Diese hinreichende Bedingung läßt die Betti-Zahl b^i aus der Anzahl der wesentlichen i -Zyklen bestimmen. Im zweidimensionalen Raum entspricht damit

- b^0 der Anzahl der Zusammenhangskomponenten,
- b^1 der Anzahl der Löcher
- $b^i = 0 \quad \forall i > 1$

und im dreidimensionalen Raum

- b^0 der Anzahl der Zusammenhangskomponenten,
- b^1 der Anzahl der Löcher erster Art, bzw. Durchbohrungen
- b^2 der Anzahl der Löcher zweiter Art, bzw. Hohlräume
- $b^i = 0 \quad \forall i > 2$

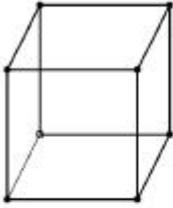
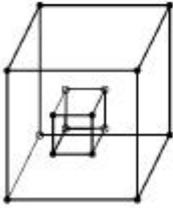
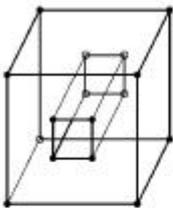
|  Würfel | <table border="1" data-bbox="775 271 1217 338"> <tr> <td>$v = \alpha_0$</td> <td>$e = \alpha_1$</td> <td>$f = \alpha_2$</td> <td>s</td> <td>h</td> <td>r</td> </tr> <tr> <td>8</td> <td>12</td> <td>6</td> <td>1</td> <td>0</td> <td>0</td> </tr> </table> <table border="1" data-bbox="786 360 1206 461"> <tr> <th colspan="4">Volumenkörper</th> <th colspan="4">Hülle</th> </tr> <tr> <td>β^0</td> <td>β^1</td> <td>β^2</td> <td>χ</td> <td>β^0</td> <td>β^1</td> <td>β^2</td> <td>χ</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>1</td> <td>0</td> <td>1</td> <td>2</td> </tr> </table> homöomorph zur Kugel | $v = \alpha_0$ | $e = \alpha_1$ | $f = \alpha_2$ | s | h | r | 8 | 12 | 6 | 1 | 0 | 0 | Volumenkörper | | | | Hülle | | | | β^0 | β^1 | β^2 | χ | β^0 | β^1 | β^2 | χ | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 2 |
|---|---|----------------|----------------|----------------|-----------|-----------|--------|----|----|----|---|---|---|---------------|--|--|--|-------|--|--|--|-----------|-----------|-----------|--------|-----------|-----------|-----------|--------|---|---|---|---|---|---|---|---|
| $v = \alpha_0$ | $e = \alpha_1$ | $f = \alpha_2$ | s | h | r | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 12 | 6 | 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Volumenkörper | | | | Hülle | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| β^0 | β^1 | β^2 | χ | β^0 | β^1 | β^2 | χ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  Würfel mit Vertiefung | <table border="1" data-bbox="775 568 1217 636"> <tr> <td>$v = \alpha_0$</td> <td>$e = \alpha_1$</td> <td>$f = \alpha_2$</td> <td>s</td> <td>h</td> <td>r</td> </tr> <tr> <td>16</td> <td>24</td> <td>11</td> <td>1</td> <td>0</td> <td>1</td> </tr> </table> <table border="1" data-bbox="786 658 1206 759"> <tr> <th colspan="4">Volumenkörper</th> <th colspan="4">Hülle</th> </tr> <tr> <td>β^0</td> <td>β^1</td> <td>β^2</td> <td>χ</td> <td>β^0</td> <td>β^1</td> <td>β^2</td> <td>χ</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>1</td> <td>0</td> <td>1</td> <td>2</td> </tr> </table> homöomorph zur Kugel | $v = \alpha_0$ | $e = \alpha_1$ | $f = \alpha_2$ | s | h | r | 16 | 24 | 11 | 1 | 0 | 1 | Volumenkörper | | | | Hülle | | | | β^0 | β^1 | β^2 | χ | β^0 | β^1 | β^2 | χ | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 2 |
| $v = \alpha_0$ | $e = \alpha_1$ | $f = \alpha_2$ | s | h | r | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | 24 | 11 | 1 | 0 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Volumenkörper | | | | Hülle | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| β^0 | β^1 | β^2 | χ | β^0 | β^1 | β^2 | χ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  Würfel mit Durchbohrung | <table border="1" data-bbox="775 866 1217 934"> <tr> <td>$v = \alpha_0$</td> <td>$e = \alpha_1$</td> <td>$f = \alpha_2$</td> <td>s</td> <td>h</td> <td>r</td> </tr> <tr> <td>16</td> <td>24</td> <td>10</td> <td>1</td> <td>1</td> <td>2</td> </tr> </table> <table border="1" data-bbox="786 956 1206 1057"> <tr> <th colspan="4">Volumenkörper</th> <th colspan="4">Hülle</th> </tr> <tr> <td>β^0</td> <td>β^1</td> <td>β^2</td> <td>χ</td> <td>β^0</td> <td>β^1</td> <td>β^2</td> <td>χ</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>2</td> <td>1</td> <td>0</td> </tr> </table> homöomorph zum Torus | $v = \alpha_0$ | $e = \alpha_1$ | $f = \alpha_2$ | s | h | r | 16 | 24 | 10 | 1 | 1 | 2 | Volumenkörper | | | | Hülle | | | | β^0 | β^1 | β^2 | χ | β^0 | β^1 | β^2 | χ | 1 | 1 | 0 | 0 | 1 | 2 | 1 | 0 |
| $v = \alpha_0$ | $e = \alpha_1$ | $f = \alpha_2$ | s | h | r | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | 24 | 10 | 1 | 1 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Volumenkörper | | | | Hülle | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| β^0 | β^1 | β^2 | χ | β^0 | β^1 | β^2 | χ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 1 | 0 | 0 | 1 | 2 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  Würfel mit Hohlraum | <table border="1" data-bbox="775 1173 1217 1240"> <tr> <td>$v = \alpha_0$</td> <td>$e = \alpha_1$</td> <td>$f = \alpha_2$</td> <td>s</td> <td>h</td> <td>r</td> </tr> <tr> <td>16</td> <td>24</td> <td>12</td> <td>2</td> <td>0</td> <td>0</td> </tr> </table> <table border="1" data-bbox="786 1263 1206 1364"> <tr> <th colspan="4">Volumenkörper</th> <th colspan="4">Hülle</th> </tr> <tr> <td>β^0</td> <td>β^1</td> <td>β^2</td> <td>χ</td> <td>β^0</td> <td>β^1</td> <td>β^2</td> <td>χ</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>2</td> <td>2</td> <td>0</td> <td>2</td> <td>0</td> </tr> </table> | $v = \alpha_0$ | $e = \alpha_1$ | $f = \alpha_2$ | s | h | r | 16 | 24 | 12 | 2 | 0 | 0 | Volumenkörper | | | | Hülle | | | | β^0 | β^1 | β^2 | χ | β^0 | β^1 | β^2 | χ | 1 | 0 | 1 | 2 | 2 | 0 | 2 | 0 |
| $v = \alpha_0$ | $e = \alpha_1$ | $f = \alpha_2$ | s | h | r | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | 24 | 12 | 2 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Volumenkörper | | | | Hülle | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| β^0 | β^1 | β^2 | χ | β^0 | β^1 | β^2 | χ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 1 | 2 | 2 | 0 | 2 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Tabelle 4.2: Topologische Invarianten eines Würfels

4.3 Euler-Poincaré-Formel

Setzt man (2) mit (3) gleich erhält man für eine zweidimensionale Fläche:

$$\mathbf{a}_0 - \mathbf{a}_1 + \mathbf{a}_2 = \mathbf{b}^0 - \mathbf{b}^1 \quad (4)$$

und einem dreidimensionalen Körper:

$$\mathbf{a}_0 - \mathbf{a}_1 + \mathbf{a}_2 - \mathbf{a}_3 = \mathbf{b}^0 - \mathbf{b}^1 + \mathbf{b}^2 \quad (5)$$

Die Gleichungen können als Hessesche Normalform einer Ebene identifiziert werden:

$$nx = d \quad (6)$$

Ausgehend von (5) ist

$$n = \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \\ -1 \\ 1 \\ -1 \end{pmatrix} \quad (7a) \quad x = \begin{pmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \\ \mathbf{b}^0 \\ \mathbf{b}^1 \\ \mathbf{b}^2 \end{pmatrix} \quad (7b)$$

und dem Abstand zum Nullpunkt

$$d = nx_0 = 0 \quad (7c)$$

Dieser Ansatz wird in Kruschwitz [4.12] weiterverfolgt.

Ein regulärer dreidimensionaler Körper kann durch seine zweidimensionale Oberfläche beschrieben werden. Diese wird hier auf 2-Mannigfaltigkeiten beschränkt, bei der jeder Punkt eine Umgebung besitzt, die homöomorph zur zweidimensionalen offenen Vollkugel ist. Für die Oberflächendarstellung eines Körpers ist

$$\mathbf{a}_0 = v \quad (8a)$$

$$\mathbf{a}_1 = e \quad (8b)$$

$$\mathbf{a}_2 = f \quad (8c)$$

$$\mathbf{a}_3 = 0 \quad (8d)$$

Die Oberflächen sind Zusammenhangskomponenten, deren Anzahl durch

$$\mathbf{b}^0 = \mathbf{b}^2 = s \quad (8e)$$

bestimmt werden kann.

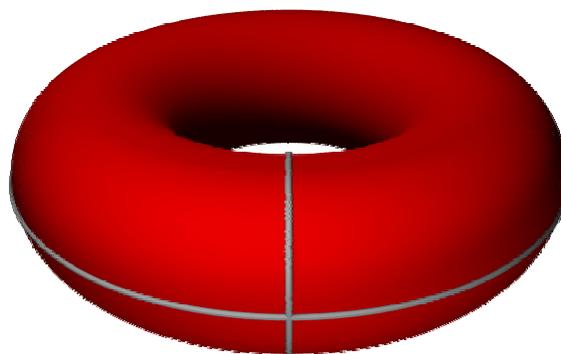


Abbildung 4.1: Torus: zwei wesentliche 1-Zyklen

Da bei jeder Durchbohrung zwei wesentliche 1-Zyklen (s. Abb. 1) entstehen, ist

$$\mathbf{b}^1 = 2h \quad (8f)$$

Ein Zellkomplex läßt ein Loch in einer Zelle nicht zu. Als Korrektur können zwei Kanten und eine Fläche hinzugefügt werden. Hieraus folgt der Korrektursummand r .

Die folgende Form der Euler-Poincaré-Formel

$$v - e + f = 2(s - h) + r \quad (9)$$

mit

v : Ecken (*v*ertices)

e : Kanten (*e*dgcs)

f : Flächen (*f*aces)

s : Zusammenhangskomponenten (*s*hells)

h : Löcher (erster Art) in Körpern (*h*oles)

r : Löcher in Flächen (*r*ings)

findet sich sowohl in Mäntylä [4.9, S. 15] als auch in Bungartz et al [4.6, S. 81]. Sie folgt durch Einsetzen von (8) in (5). Auch diese Gleichung läßt sich als Hessesche Normalform einer Ebene im \mathbb{N}^6 ausdrücken:

$$v - e + f - 2s + 2h - r = 0 \quad (10)$$

Mit

$$n = \begin{pmatrix} 1 \\ -1 \\ 1 \\ -2 \\ 2 \\ -1 \end{pmatrix} \quad (11a)$$

$$x = \begin{pmatrix} v \\ -r \\ f \\ s \\ h \\ r \end{pmatrix} \quad (11b)$$

und (7c) läßt sich (6) in (10) überführen.

Im zweidimensionalen Fall ergibt sich als Ebenengleichung, ausgehend von (4):

$$v - e + f - s + r = 0 \quad (12)$$

Dies entspricht einer vierdimensionalen Hyperebene im fünfdimensionalen Raum. Wird nur hier nur eine Zusammenhangskomponente und eine Fläche ohne Löcher betrachtet reduziert sich die Gleichung auf eine zweidimensionale Ebene.

4.4 Euler Operatoren

Gesucht ist eine Basis zum Teilraum \mathbb{N}^5 der dreidimensionalen Polyeder.

Wie bereits in Abschn. 3 angesprochen bildet dieser Teilraum eine Ebene im \mathbb{N}^6 mit dem in (11a) angegebenen Normalenvektor. Die 5 gesuchten Basisvektoren müssen orthogonal zum Normalenvektor und linear unabhängig zueinander sein. Aus (7c) folgt, daß die Ebene durch den

Nullpunkt geht und kein Ortsvektor benötigt wird. Jeder Basisvektor entspricht einer Operation. Eine mögliche Auswahl ist Mäntylä [4.9] entnommen und in Tabelle 4.3 mit den Abkürzungen der englischen Bezeichnungen aufgeführt (s. auch [4.5, 4.6]).

| Operation | v | e | f | s | h | R |
|--------------|----|----|----|----|----|----|
| <i>mvfs</i> | 1 | 0 | 1 | 1 | 0 | 0 |
| <i>kvfs</i> | -1 | 0 | -1 | -1 | 0 | 0 |
| <i>mev</i> | 1 | 1 | 0 | 0 | 0 | 0 |
| <i>kev</i> | -1 | -1 | 0 | 0 | 0 | 0 |
| <i>mef</i> | 0 | 1 | 1 | 0 | 0 | 0 |
| <i>kef</i> | 0 | -1 | -1 | 0 | 0 | 0 |
| <i>kemr</i> | 0 | -1 | 0 | 0 | 0 | 1 |
| <i>mekr</i> | 0 | 1 | 0 | 0 | 0 | -1 |
| <i>kfmrh</i> | 0 | 0 | -1 | 0 | 1 | 1 |
| <i>mfkrh</i> | 0 | 0 | 1 | 0 | -1 | -1 |

Tabelle 4.3: Euler Operatoren

Die lineare Unabhängigkeit der Basisvektoren kann durch Bestimmung der Determinante

$$\begin{vmatrix}
 1 & -1 & 1 & 2 & -2 & -1 \\
 1 & 0 & 1 & 0 & 1 & 0 \\
 1 & 1 & 0 & 0 & 0 & 0 \\
 0 & 1 & 1 & 0 & 0 & 0 \\
 0 & -1 & 0 & 0 & 0 & 1 \\
 0 & 0 & -1 & 1 & 0 & 1
 \end{vmatrix} = 12 \neq 0 \quad (13)$$

nachgewiesen werden. Weiterer Vektoren in der Ebene sind linear abhängig. Die Basisvektoren entsprechen den Euler-Operatoren. Die Verwendung weiterer, linear abhängiger Euler-Operatoren kann die Konstruktion bei Erhaltung der Konsistenz des Modells vereinfachen, die Eindeutigkeit geht aber verloren.

Make Vertex & Face & Shell / Kill Vertex & Face & Shell Der *mvfs*-Operator wird üblicherweise zu Beginn der Konstruktion eines Körpers verwendet. Aus dem leeren Modell wird ein Körper, der aus einem Vertex, einer Fläche und einer Oberfläche besteht. Einen solchen Körper kann man sich als eine Kugel vorstellen. Auch Hohlräume werden mit diesem Operator erzeugt. *kfs* bezeichnet den inversen Operator, der die Operation rückgängig macht und eine aus einer Fläche und einem Vertex bestehende Oberfläche löscht.

Make Edge & Vertex / Kill Edge & Vertex *mev* fügt eine weitere Kante an einen bestehenden Vertex an. Zunächst muß der zweite Vertex erzeugt werden. Eine Sonderform dieses Operators ist durch *semv* gegeben. Hier wird eine Kante geteilt (split edge) indem ein Vertex (make vertex) und eine weitere Kante erzeugt wird. *kev* löscht eine freie Kante mit dem entsprechenden Vertex. *jevk* (join edge, kill vertex) löscht einen Vertex mit dem Grad 2, und löscht beim der Vereinigung eine der zwei adjazenten Kanten.

Make Edge & Face / Kill Edge & Face *mef* schließt einen Kantenzug mit einer neu erzeugten Kante und erzeugt hiermit eine neue Fläche. *kef* löst hingegen eine Fläche auf, indem er eine Begrenzungskante herausnimmt.

Kill Edge & Make Ring / Make Edge & Kill Ring Ist der äußere Rand einer umschlossenen Fläche durch eine Kante mit dem äußeren Rand der umschließenden Fläche verbunden, entsteht beim

Löschen dieser Kante mit *kemr* eine Fläche in einer Fläche. *mekr* hingegen löscht ein Loch in einer Fläche, indem durch eine Kante eine einfach zusammenhängende umschließende Fläche entsteht.

Kill Face & Make Ring & Hole / Make Face & Kill Ring & Hole *kfmrh* löscht eine Fläche und läßt den äußeren Rand Loch einer anderen Fläche werden. Hierbei entsteht ein Volumenloch und das Geschlecht des Körpers wird inkrementiert.

Weiler [4.11] verwendet den Begriff loop für einen Rand-Kreis und faßt damit den äußeren Rand einer Fläche sowie die ring-Elemente zusammen. Beim Erzeugen einer Fläche oder statt eines rings muß ein loop erzeugt werden. Die Unterscheidung zwischen Flächen-loop und ring-loop ist implizit durch die Reihenfolge der Entstehung gegeben: der erste loop einer Fläche ist deren äußerer Rand. Analog können die shells "Hohlraum" und "äußeren Rand eines Körpers" unterschieden werden. Die gewählten Euleroperatoren in [4.11] weichen teilweise von den vorgestellten ab.

Auf zweidimensionale Flächen bezogene Euleroperatoren stellen sicher, daß (12) erfüllt wird. Hierfür sind vier linear unabhängige Operatoren nötig. Zunächst einmal muß eine Fläche generiert werden, die (12) erfüllt, wie z. B. ein 2-Simplex mit 3 Knoten, 3 Kanten, einer Fläche und einer Zusammenhangskomponente. Für eine zusammenhängende Fläche ohne Löcher sind ausgehend von einer bereits bestehenden Fläche, die beiden bereits vorgestellten Operatorenpaare *mev/kev* und *mef/kef* ausreichend. Ein Loch kann erzeugt werden, indem man eine Flächen löscht: *mrkf* (make ring, kill face). Entsprechend entfernt *krmf* ein Loch. Wird eine Zusammenhangskomponente in zwei Teile zerschnitten, muß die Schnittkante dupliziert werden. Es entstehen beim *mves* also n neue Knoten, $n - 1$ Kanten und eine neue Zusammenhangskomponente, die beim Zusammenfügen durch *kves* gelöscht werden. Diese Auswahl stellt eine Möglichkeit dar.

4.5 Beispiele zur Konstruktion mit Euleroperatoren

4.5.1 Tetraeder

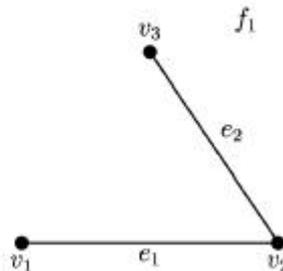
Zur Konstruktion eines Tetraeders benötigt man 7 Operationen:

1. *mvfs* erzeugt v_1, f_1 und die Hülle

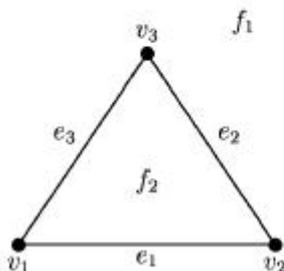


2. *mev* erzeugt e_1, v_2

3. *mev* erzeugt e_2, v_3

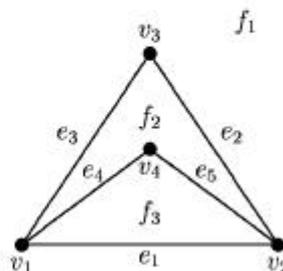


4. *mef* erzeugt e_3, f_2

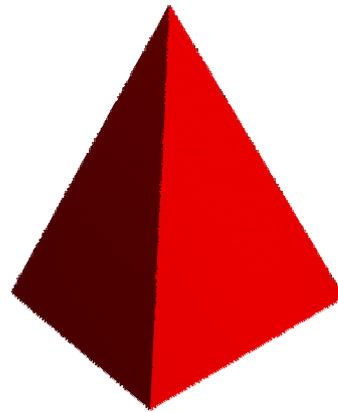
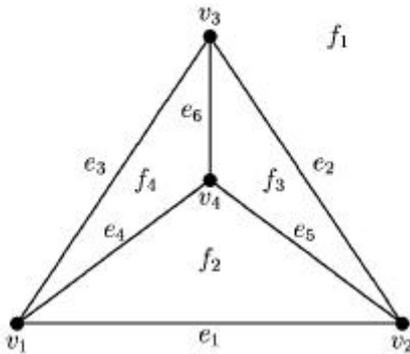


5. *mev* erzeugt e_4, v_4

6. *mef* erzeugt e_5, f_3



7. mef erzeugt e_6, f_4

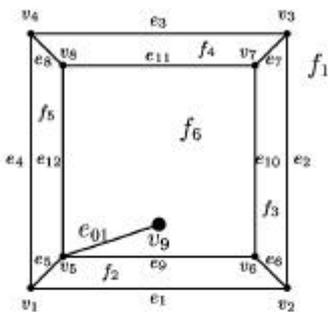


Werden die Abbildungen als Draufsichten interpretiert, kann die im ersten Schritt erzeugte Fläche f_1 als Grundfläche angesehen werden.

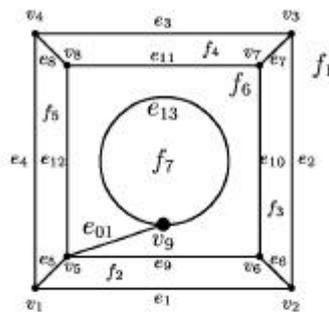
4.5.2 Würfel mit Durchbohrung

Der Würfel wird in 13 Schritten analog zum Tetraeder in Abschnitt 5.1 konstruiert. Im folgenden werden die Schritte zur Konstruktion einer zylinderförmigen Durchbohrung dargestellt.

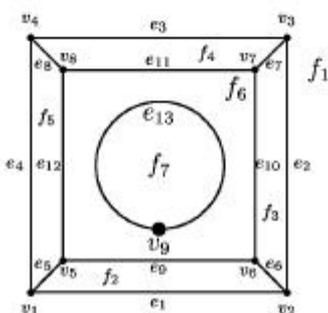
1. mev erzeugt e_{01}, v_9



2. mef erzeugt e_{13}, f_7



3. $kemr$ löscht e_{01}
und definiert f_7 in f_6



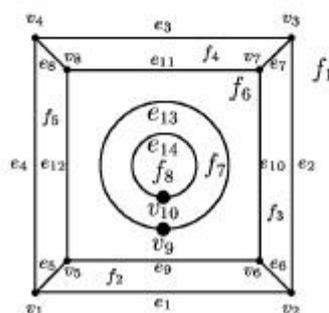
4. mev erzeugt e_{02}, v_{10}

5. mef erzeugt e_{14}, f_8

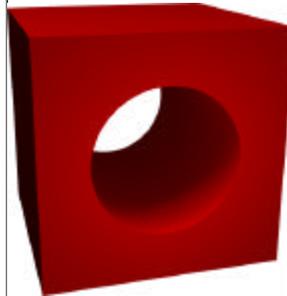
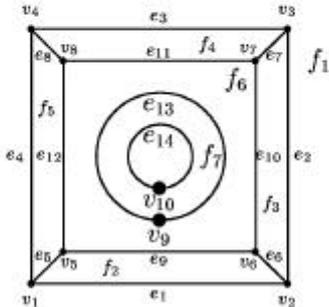
6. $kemr$ löscht e_{02}

und definiert f_8 in f_7

Man erhält einen Würfel mit Vertiefung



7. $kfmrh$ löscht f_8 und definiert dessen Umgrenzung als Loch von f_1 . Hierdurch wird eine Durchbohrung erzeugt.



4.6 Zellinzidenzgraph

Die zweimannigfaltige Oberfläche eines regulären Körpers wird als Zellkomplex durch Punkte(V), Kanten(E) und Flächen(F) beschrieben. Die Beschreibung benötigt neben den Elementen auch Relationen zwischen den Elementen, die durch einen Zellinzidenzgraphen bzw. vef-Graphen (s. [4.6]) $G = (V, E, F; R)$ dargestellt werden können. Die möglichen Relationen sind in Tab. 4 aufgeführt. Bei Betrachtung aller Zellinzidenzen sind redundante Informationen zu finden. Die Speicherung aller Relationen ermöglicht einen optimal schnellen Zugriff auf die Daten. Eine Modifikation der Daten erfordert allerdings die Anpassung aller Relationen. Die Konsistenz der Daten muß sichergestellt werden. Das Abspeichern redundanter Daten erhöht zudem das Datenvolumen. Werden nur die nötigsten Relationen abgespeichert müssen bei Modifikationen weniger Daten angepaßt werden und es wird weniger Speicher verwendet. Die Abfrage nicht gespeicherter Daten erfordert hier allerdings einen Berechnungsschritt, der aufwendig sein kann. Die im folgenden vorgestellten Datenstrukturen suchen einen sinnvollen Mittelweg zwischen diesen beiden Extremen.

| | V | E | F |
|---|--|--|---|
| V | $VV \subseteq V \times V$ Punkt ist adjazent zu Punkt | $VE \subseteq V \times E$ Punkt ist Endpunkt von (bzw. inzident zu) Kante | $VF \subseteq V \times F$ Punkt ist Eckpunkt von Fläche |
| E | $EV \subseteq E \times V$ Kante hat Punkt als Endpunkt | $EE \subseteq E \times E$ Kante ist benachbart zu (hat gemeinsamen Punkt mit) Kante | $EF \subseteq E \times F$ Kante begrenzt Fläche |
| F | $FV \subseteq F \times V$ Fläche hat Punkt als Eckpunkt | $FE \subseteq F \times E$ Fläche hat Kante als Begrenzungskante | $FF \subseteq F \times F$ Fläche ist benachbart zu (hat gemeinsame Kante mit) Fläche |

Tabelle 4.4: mögliche Zellinzidenzen

Diese Beziehungen können als ein Graph dargestellt werden, in dem jede Kante eine gespeicherte Relation und die Knoten die Elemente V, E und F repräsentiert. Die jeweils symmetrische Relation kann durch eine totale Enumeration der betreffenden Relation ermittelt werden. So kann z. B. die VE-Relation durch Untersuchung aller EV-Paare gewonnen werden. Sollen lokale Größen die Komplexität des Aufwandes bestimmen, berechnet man eine Relation aus zusammenhängenden Relationen. Zu beachten ist, daß nicht alle Verknüpfungen von einzelnen Relationen das gewünschte Ergebnis liefert. Wird z. B. durch die Relation FE alle Kanten einer Fläche und anschließend über EV alle Knoten

dieser Kanten ermittelt, so erhält man alle Knoten einer Fläche, was der Relation FV entspricht. Ebenso führt VE und EF zu VF. Werden aber alle Knoten einer Fläche mit FV ermittelt, um die zugehörigen Kanten anschließend mit VE zu bestimmen, erhält man auch Kanten, die nicht zur Relation FE gehören. Aufbauend auf Verknüpfungen von Relationen muß der Graph zusammenhängend sein, um alle fehlenden Relationen zu berechnen. Wählt man z. B. die Relationen VE, EV, EF und FE ergibt sich für die Adjazenzmatrix des Graphen

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad (14)$$

und für die transitive Hülle

$$\begin{aligned} T &= A + A^2 + A^3 \\ &= \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 2 & 0 \\ 2 & 0 & 2 \\ 0 & 2 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 3 & 1 \\ 3 & 2 & 3 \\ 1 & 3 & 1 \end{pmatrix} \end{aligned} \quad (15)$$

Der Graph ist zusammenhängend und die nicht gespeicherten Zellinzidenzen können nach Tab. 5 berechnet werden. Die Speicherung der gewählten Zellinzidenzen ist somit ausreichend.

| | V | E | F |
|---|-----------------|-----------------|-----------------|
| V | $VE \otimes EV$ | VE | $VE \otimes EF$ |
| E | EV | $EV \otimes VE$ | EF |
| F | $FE \otimes EV$ | FE | $FE \otimes EF$ |

Tabelle 4.5: gespeicherte und berechnete Zellinzidenzen

4.7 Datenstruktur

Topologische und geometrische Daten werden getrennt betrachtet. Im folgenden wird die Geometrie als Attribut der Zellen verstanden. Knoten erhalten so Koordinaten und Kanten oder Flächen Krümmungen. Im Zentrum steht die Kante.

4.7.1 Winged Edge

Im Zentrum der "Winged Edge"-Datenstruktur steht die Kante. Es werden alle Kanten-Relationen gespeichert: Die Kante speichert Referenzen der Endknoten v_p und $v_n(EV)$, der angrenzenden Flächen f_l und $f_r(EF)$, sowie der anschließenden Kanten der Flächen in Uhrzeiger- und Gegenuhrzeigerrichtung e_cwcp , e_cwn , e_cwn und e_cwp (Teilmenge von EE). Die Lage der zur Kante inzidenten Zellen ist in Abb. 2 dargestellt².

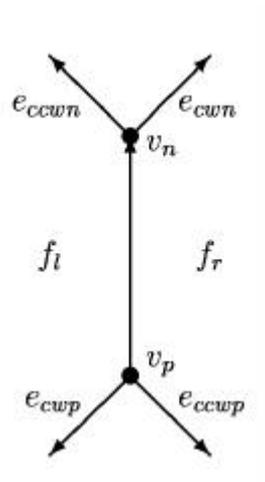


Abbildung 4.2: "Winged Edge"-Datenstruktur

Jeder Knoten und jede Fläche speichert eine anschließende Kante. Die Relationen können bestimmt werden, indem man durch die Kantenringlisten iteriert. Für $V \in E$ ermittelt man zunächst die dem Knoten inzidente Kante und bestimmt deren an diesem Knoten anschließende Kante in Uhrzeiger- bzw. Gegenuhrzeigerrichtung. Von dieser Kante aus wird das Vorgehen so lange wiederholt, bis die Ursprungskante wieder erreicht ist und alle Kanten bestimmt sind. Auch für FE ist ein analoges Vorgehen möglich. Ausgehend von einer Begrenzungskante kann im Uhrzeigersinn oder Gegenuhrzeigersinn über alle Begrenzungskanten der Fläche iteriert werden. Es wird deutlich, daß es eigentlich nicht erforderlich ist die Verweise auf die Kanten im Uhrzeigersinn und Gegenuhrzeigersinn zu speichern. Es ist ausreichend nur zwei dieser Verweise zu speichern. Man erhält die Half "Winged Edge"-Datenstruktur [4.6]. Bezogen auf die Komplexität gewinnt man allerdings nicht viel. Weiler [4.11] kombiniert in der Modified "Winged Edge"-Datenstruktur die "Winged Edge"-Datenstruktur mit der "Half Edge"-Datenstruktur, die im folgenden beschrieben wird.

4.7.2 Half Edge

Das Halbkantenmodell [4.8, 4.13] bzw. Doubly Connected Edge List (DCEL) [4.7] oder "Face-Edge"-Datenstruktur [4.11], teilt die "Winged Edge"-Datenstruktur in zwei Teile. Jede Halbkante erhält den Verweis auf einen Knoten, eine Fläche und zwei zu dieser Fläche gehörende angrenzende Halbkanten. Die "Winged Edge"-Datenstruktur wird hiermit in zwei Teile aufgespalten. Beim Iterieren entfällt die Abfrage, ob eine Kante in ihrer Orientierung oder entgegengesetzten Orientierung durchlaufen wird. Allerdings muß bei der Attributierung der Kante die Teilung in zwei Halbkanten in Kauf genommen werden.

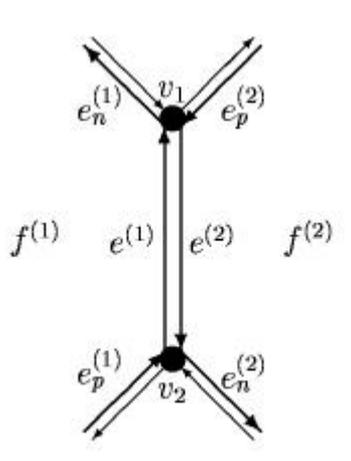


Abbildung 4.3: "Half Edge"-Datenstruktur

4.8 Programmumsetzung

Im Gegensatz zu [4.13] wird bei der Implementierung in der Programmiersprache C++ auf die Verwendung von Templates verzichtet³. Für die Erzeugung neuer Objekte wird das Entwurfsmuster der Abstract Factory [4.14] eingesetzt. Eine von einer vorgegebenen abstrakten Basisklasse abgeleitete Fabrik erzeugt in einer Applikation die spezifischen Objekte (s. 4). Die Topologie-Klassenbibliothek tritt gegenüber der Applikation als Klient auf. Für die topologischen Primitive sind die Klassen Vertex (Knoten), Edge(Kante), Loop, Face(Fläche), Shell und Solid vorgegeben. Eine CAD-Anwendung muß zumindest von der Vertex-Klasse ableiten, um die Koordinaten des Knotens zu speichern. Eine geometrische Attributierung ist in dieser Weise auch für die Kanten und Flächen möglich. Die topologischen Objekte werden von der Applikation nicht direkt, sondern über die in Abschn. 4 beschriebenen Euler-Operatoren in der Topologie-Klassenbibliothek erzeugt. Diese Prozeduren liefern der Applikation Referenzen auf neu erzeugte Elemente. Intern existieren Hilfsfunktionen für wiederkehrende Arbeitsschritte. Das Entwurfsmuster der Iteratoren [4.14] ermöglichen den Zugriff auf die Elemente. Das Datenmodell baut auf der "Winged Edge"-Datenstruktur (s. 7.1) auf. Die einzelnen Elemente sind somit durch Ringlisten miteinander verknüpft.

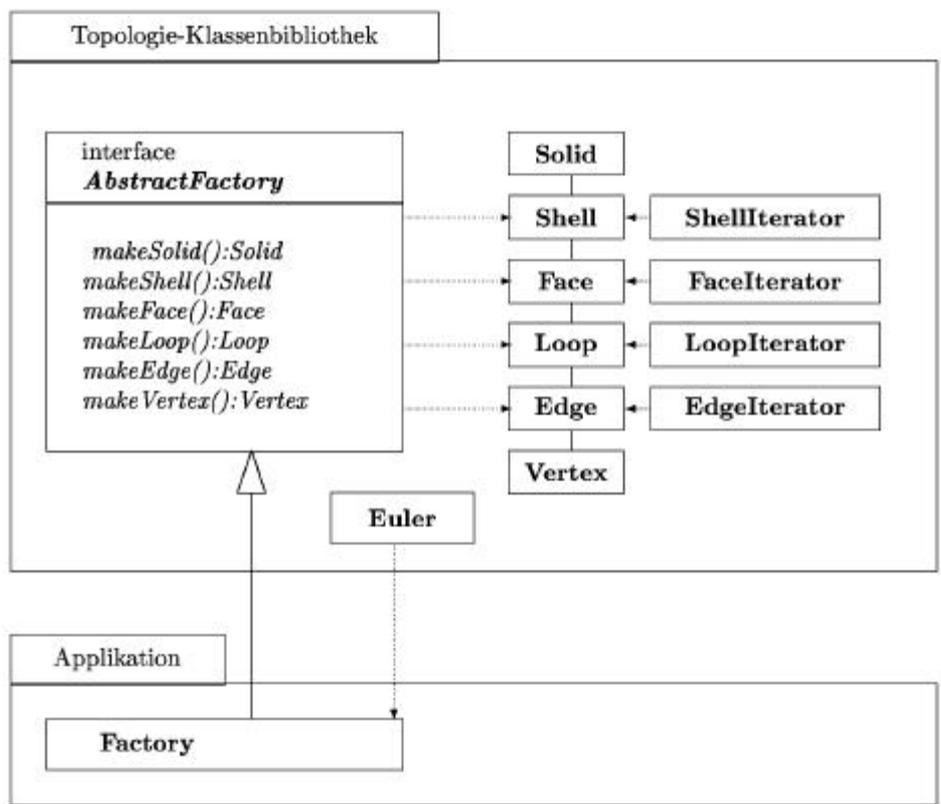


Abbildung 4.4: Topologie Klassenbibliothek

³ Der Entwurf stammt hauptsächlich von Herrn K. Schwebke

Literatur zu Kapitel 4

- [4.1] E. Zeidler (Hrsg.): Teubner-Taschenbuch der Mathematik; Teubner Verlag; 1996
- [4.2] Frank Harary: Graphentheorie; Oldenbourg Verlag; 1974
- [4.3] Klaus Jänich: Topologie; Springer Verlag; 1999
- [4.4] Vladimir G. Boltjanskij, V. A. Efremovic: Anschauliche kombinatorische Topologie; VEB Deutscher Verlag der Wissenschaften, Vieweg; 1986
- [4.5] Stephan Abramowski, Heinrich Müller: Geometrisches Modellieren; BI Wissenschaftsverlag; 1991
- [4.6] Hans-Joachim Bungartz, Michael Griebel, Christoph Zenger: Einführung in die Computergraphik: Grundlagen, Geometrische Modellierung, Algorithmen; Vieweg & Sohn Verlagsgesellschaft; 1996
- [4.7] Martti Mäntylä: An Introduction to Solid Modeling; Principles of Computer Science Series No. 13; Computer Science Press; 1988
- [4.8] Christoph M. Hoffmann: Geometric & Solid Modeling; Morgan Kaufmann Publishers Inc.; 1989
- [4.9] Martti Mäntylä: Computational Topology: A Study of Topological Manipulations and Interrogations in Computer Graphics and Geometric Modeling; Mathematics and Computer Science Series No. 37; Thesis, Helsinki University of Technology; 1983
- [4.10] Kevin Weiler: Topological structures for geometric modeling; Thesis, Rensselaer Polytechnic Institute; 1986
- [4.11] Erwin Kruschwitz: Euler-Modellierung dreidimensionaler Körper; Shaker Verlag; 1996
- [4.12] Franco P. Preparata, Michael Ian Shamos: Computational Geometry: An Introduction; Springer-Verlag; 1985
- [4.13] Mark de Berg, Marc van Kreveld, Mark Overmars, Otfried Schwarzkopf: Computational Geometry: Algorithms and Applications, 2nd edition; Springer-Verlag; 2000
- [4.14] Stefan Schirra, Remco Veltkamp, Mariette Yvinec, Ed.: CGAL Reference Manual, Release 2.2; <http://www.cgal.org/>; October 2000
- [4.15] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software; Addison-Wesley; 1996

5. Modellierung mit einem kommerziellen System (H. Bröker und V. Nübel)

Mittlerweile verfügen nahezu alle kommerziellen CAD-Systeme über eine Programmierschnittstelle, die dem Anwender eine individuelle Erweiterung der Funktionalitäten ermöglicht. Im folgenden soll die Programmierschnittstelle ObjectARX (AutoCAD Runtime eXtension) des CAD-System AutoCAD von Autodesk näher betrachtet werden. Der wesentliche Unterschied zu der in Kapitel 4 beschriebenen Programmierschnittstelle ACIS besteht nun darin, dass nicht nur die Modellierungsfunktionen sondern zusätzlich Verwaltungsfunktionen eines CAD-Systems zur Verfügung stehen.

5.1. Allgemeines über die Entwicklung

ObjectARX ist mittlerweile die dritte Programmierschnittstelle, die Autodesk für ihr CAD-System AutoCAD anbietet. Zunächst stand die Interpretersprache AutoLISP zur Verfügung um relativ einfach eigene kleinere Anwendungen auszuprobieren. Anschließend mit Einführung der AutoCAD Release 11 wurde auf die Programmiersprache C (Compilersprache) aufgesetzt und die Schnittstelle ADS (AutoCAD Development System) entwickelt. Der wesentliche Vorteil gegenüber AutoLISP liegt in einem schnellen Zugriff auf die Objekte, der mit Einführung von ObjectARX nochmals um Größenordnungen gesteigert werden konnte. Bei ObjectARX handelt sich um eine objektorientierte C++-Klassenbibliothek.

5.2. ARX-Klassen/Libraries

AcRx: In der Hierarchie von ObjectARX ist dies die oberste Klasse mit deren Hilfe die Initialisierung der .dll und die Registrierung von ARX-Klassen zur Laufzeit gesteuert wird.

AcEd: Diese Klassen dienen der Definition und Registrierung neuer AutoCAD Funktionen.

AcDb: Hierbei handelt es sich um die Datenbasis-Klassen von AutoCAD. Neben dem Zugriff auf die grafischen Objekte wird hier ebenfalls die Verwaltung der weiteren Attribute vorgenommen. Wie sich die Datenbasis im einzelnen gliedert, wird im nachfolgenden Abschnitt erläutert.

AcGi: Diese Klassenbibliothek stellt das Grafikinterface zum Zeichnen der AutoCAD-Objekte dar.

AcGe: Diese Klassenbibliothek stellt eine Reihe geometrischer Funktionalitäten, also Algorithmen der linearen Algebra zur Verfügung. Im Hinblick auf die Verwendung von ACIS innerhalb von ObjectARX kommt dieser Bibliothek eine besondere Bedeutung zu. Der Zugriff auf die unterschiedlichen Kurven- und Flächentypen des Brep-Modells findet hier statt.

AcBr: Mit Hilfe dieser Klassen erfolgt (ein etwas eingeschränkter) Zugriff auf die Brep-Struktur des in Kapitel 4 dargestellten ACIS Kernel.

Schließlich bietet AutoCAD ein MFC User Interface an. Dabei wird das Autodesk MFC-System in zwei Bibliotheken aufgeteilt. Das erste wird mit **AdUi** bezeichnet und ist nicht AutoCAD spezifisch, wohingegen das zweite, **AcUi** ein AutoCAD spezifisches Erscheinungsbild und Verhalten aufweist.

5.3. AutoCAD Datenbasis

Eine AutoCAD-Zeichnung (.dwg bzw. .dxf-Datei) ist zunächst ganz grob ausgedrückt eine Ansammlung von Objekten, die in einer Datenbasis gespeichert werden. Dabei wird jedes Objekt eindeutig durch eine ID identifiziert. Spezielle Objekte in der Datenbasis stellen die sogenannten

Entities dar. Es handelt sich dabei um grafische Objekte, wie zum Beispiel Linien, Kreise, Splines usw., die auf dem Bildschirm zu sehen und durch den Benutzer manipulierbar sind.

Um die Datenbasis-Objekte zu speichern, bedient man sich den sogenannten *SymbolTables* und den *Dictionaries*. Sie stellen Kontainer-Klassen dar, die den Datenbasis-Objekten einen Namen (Text-String) entsprechend ihrer Funktionen zuweisen. AutoCAD beinhaltet einen vorgegebenen Satz dieser *SymbolTables*, wovon jeder einzelne wiederum eine Instanz von besonderen Klassen, den *SymbolTableRecords* enthält. Eine Ausnahme bildet hierbei der *BlockTable*, der drei *Records* beinhaltet. Ein *Record* repräsentiert dabei den Modellbereich und die zwei anderen stellen vordefinierte Papierbereiche dar.

Ein Hinzufügen neuer *SymbolTables* in die AutoCAD Datenbasis ist nicht möglich.

Die *Dictionaries* stellen eine allgemeinere Kontainer-Klasse zur Speicherung von Objekten dar, als die *SymbolTables*. Sie können nahezu jedes Objekt der Klasse *AcDbObject* oder hiervon abgeleitete Klassen beinhalten. Mit jeder neuen AutoCAD-Zeichung wird ein sogenanntes *Named Object Dictionary* angelegt, das dann neue *Dictionaries* aufweisen kann, indem neue Datenbasis-Objekte hinzugefügt werden.

Schematisch stellt sich die AutoCAD Datenbasis dann wie folgt dar:

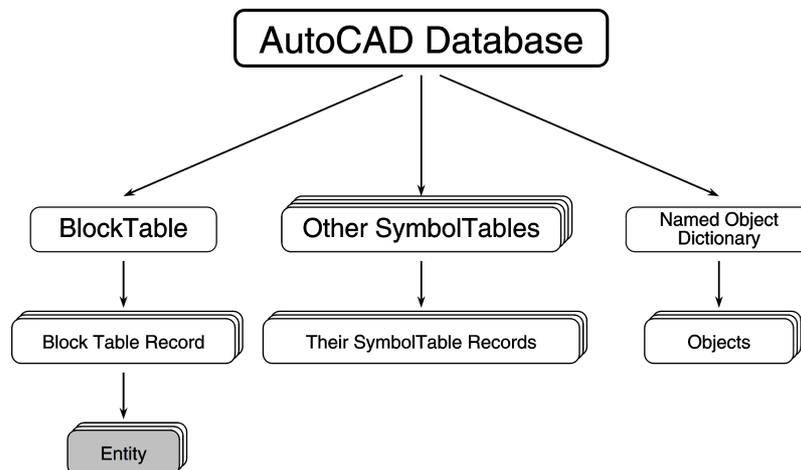


Bild 5.1: AutoCAD Datenbasis (© Autodesk 1999)

Nachfolgend sind die neun *SymbolTables* von der AutoCAD-Datenbasis aufgeführt:

1. *AcDbAbstractView*
2. *AcDbBlockTable*
3. *AcDbDimStyleTable*
4. *AcDbFontTable*
5. *AcDbLayerTable*
6. *AcDbLinetypeTable*
7. *AcDbRegAppTable*
8. *AcDbTextStyleTable*
9. *AcDbUCSTable*

Der Zugriff auf die aktuelle Datenbasis geschieht dann durch die folgende globale Funktion:

```
acdbHostApplicationServices()->workingDatabase()
```

Nachfolgend ist beispielhaft ein ObjectARX Programmcode für die Erstellung einer Linie im Modellbereich aufgezeigt:

```

AcDbObjectId createLine()
{
    // Start- und Endpunkt definieren
    AcGePoint3d startP(0.0,0.0,0.0);
    AcGePoint3d endP(10.0,15.0,20.0);

    // Linie generieren mit Hilfe von Start- und Endpunkt
    AcGeLine *pLine = new AcDbLine(startP,endP);

    AcDbBlockTable *pBlockTable;

    // Zugriff auf entsprechenden SymbolTable der aktuellen Datenbasis
    acdbHostApplicationServices()->workingDatabase()->getSymbolTable(pBlockTable,
    AcDb::kForRead)

    AcDbBlockTableRecord *pBlockTableRecord;

    // Zugriff auf Record im Modellbereich
    pBlockTable->getAt(ACDB_MODEL_SPACE, pBlockTableRecord, AcDb::kForWrite);

    // BlockTable schliessen
    pBlockTable->close();

    AcDbObjectId lineId;

    // Entity dem BlockTable Record hinzufügen
    pBlockTableRecord->appendAcDbEntity(lineId, pLine);

    // BlockTableRecord schliessen
    pBlockTableRecord->close();

    // Entity schliessen
    pLine->close();

    return lineId;
}

```

5.4. Anlegen einer AutoCAD –ObjectARX Anwendung

Wie bereits eingangs erwähnt, handelt es sich bei einer ObjectARX Anwendung um eine DLL (dynamic link library), die die unmittelbare Nutzung des Prozess- und Speicherbereichs von AutoCAD ermöglicht. Dabei kann die Erstellung einer solchen ObjectARX Anwendung in folgende Schritte aufgeteilt werden:

- (1) Anlegen individueller Klassen um neue AutoCAD Kommandos zu implementieren
- (2) Festlegen des neuen Kommandos unter AutoCAD mit dem die ObjectARX Anwendung ausgeführt werden soll
- (3) Implementation eines *entry points* für AutoCAD
- (4) Initialisierung der Implementation
- (5) Löschen der Implementation z.B.beim Verlassen von AutoCAD

Beispiel:

```

// -----
// EntryPointFunction
// -----
extern "C" AcRx::AppRetCode
acrEntryPoint(AcRx::AppMsgCode msg, void* pkt)
{
    // AcRx::AppRetCode: Beinhaltet den Status-Code der an AutoCAD gesendet wird

```

```

// msg: Die Nachricht die von dem ObjectARX kernel beim Ausführen der Anwendung gesendet wird
// pkt: Trägt Paket-Daten
switch (msg)
{
    case AcRx::kInitAppMsg:
        acrxDynamicLinker->unlockApplication(pkt);
        acrxDynamicLinker->registerAppMDIAware(pkt);
        initModule(pkt);
        break ;
    case AcRx::kUnloadAppMsg:
        unloadModule();
        break ;
    default:
        break ;
}
return AcRx::kRetOK;
}

// -----
// AutoCAD Initialisierungs Modul
// -----
static void
initModule(void* Ptr)
{
    // Register the class
    // -----
    AsdkMeshThree::rxInit();
    acrxBuildClassHierarchy();

    // Register the commands
    // -----
    acedRegCmds->addCommand("FEM3D_CMD", "readMesh",
                           "READMESH", ACRX_CMD_TRANSPARENT, &fem_readMesh);
    acedRegCmds->addCommand("FEM3D_CMD", "writeMesh",
                           "WRITEMESH", ACRX_CMD_TRANSPARENT, &fem_writeMesh);
}

// -----
// AutoCAD Unload Modul
// -----
static void
unloadModule()
{
    acedRegCmds->removeGroup("FEM3D_CMD");
    deleteAcRxClass(AsdkMeshThree::desc());
}

```