UNIVERSITÄT PADERBORN
*Die Universität der Informationsgesellschaft*

# Methoden und Werkzeuge zur Entwicklung dreidimensionaler visueller Sprachen

## Dissertation

zur Erlangung des akademischen Grades
**Doktor der Naturwissenschaften (Dr. rer. nat.)**
an der Fakultät für Elektrotechnik, Informatik und Mathematik
der Universität Paderborn

vorgelegt von

**Jan Wolter**

Paderborn, April 2015

**Betreuer:**

Prof. Dr. Uwe Kastens (Universität Paderborn)

**Gutachter:**

Prof. Dr. Uwe Kastens (Universität Paderborn)
Prof. Dr.-Ing. Mark Minas (Universität der Bundeswehr München)

**Weitere Mitglieder der Promotionskommission:**

Prof. Dr. Gitta Domik (Universität Paderborn)
Prof. Dr. Gregor Engels (Universität Paderborn)
Dr. Stefan Sauer (Universität Paderborn)

**Datum der mündlichen Prüfung:**

20. April 2015

# Danke!

Viele Personen haben mich innerhalb der letzten vier Jahre bei der Anfertigung meiner Dissertation unterstützt. Ihnen möchte ich an dieser Stelle herzlich danken.

Besonderer Dank gilt meinem Doktorvater Prof. Dr. Uwe Kastens. Er hat mir nach meiner Masterarbeit die Chance gegeben auf dem Gebiet der dreidimensionalen visuellen Sprachen weiterzuarbeiten. Er hat mich jederzeit unterstützt und durch anregende Diskussionen maßgeblich zum Gelingen der Arbeit beigetragen. Besonders seine detaillierten Anmerkungen zu Konferenzpapieren und dieser Dissertation waren stets hilfreich.

Ebenso danke ich meinem Zweitgutachter Prof. Dr.-Ing. Mark Minas für seine vielen wertvollen Hinweise und die Verbesserungsvorschläge zu einem ersten Entwurf der Dissertation.

Allen Kollegen der Fachgruppe danke ich für die angenehme und freundschaftliche Zusammenarbeit. Besonders bedanke ich mich bei Dr. Bastian Cramer, der mir zu Beginn meiner Beschäftigung durch seine Unterstützung und Erfahrung die Einarbeitung erleichtert hat.

Viele Studenten haben durch Bachelor- oder Masterarbeiten sowie als studentische Hilfskräfte zum Erfolg des Projekts beigetragen. Auch ihnen danke ich.

Für das sorgfältige Korrekturlesen meiner Dissertation bedanke ich mich herzlich bei meiner Mutter Petra Wolter.

Meinen Eltern Eckhard und Petra Wolter danke ich für ihre fortwährende Unterstützung, insbesondere während meines Studiums. Äußerst wertvoll war auch die Unterstützung durch meine Freundin Angelina Thole, die an so manchem Wochenende, an dem ich an der Dissertation gearbeitet habe, auf mich verzichten musste.

*Jan Wolter*
Paderborn, 3. März 2015

# Zusammenfassung

Visuelle Sprachen sind besonders für domänenspezifische Anwendungsbereiche nützlich. Sie unterstützen grafische Metaphern ihrer Domäne und ermöglichen so Domänenexperten, Programme in ihrer gewohnten Ausdrucksweise zu beschreiben. Die meisten visuellen Sprachen nutzen zweidimensionale Darstellungen, aber für manche Anwendungszwecke ist auch die Verwendung der dritten Dimension vorteilhaft oder notwendig. Eine Darstellung in drei Dimensionen ist für *inhärente 3D-Sprachen*, wie z. B. den *Kugel-Stäbchen-Modellen* von Molekülen, nützlich. Weitere 3D-Sprachen weisen jeder Dimension eine *semantische Bedeutung* zu oder *überwinden Limitierungen von 2D-Darstellungen*.

Zur Konstruktion visueller Diagramme einer bestimmten Domäne werden dedizierte grafische Strukturediteren verwendet, die spezialisierte Interaktions- und Navigationstechniken anbieten. Die Implementierung dreidimensionaler visueller Sprachen und ihrer Editoren benötigt ein umfangreiches konzeptionelles und technisches Wissen im Bereich von 3D-Grafiken und Übersetzerbau. Die Entwicklung von grafischen Editoren für domänenspezifische visuelle 3D-Sprachen kann durch die Verwendung von Generatorsystemen vereinfacht werden. Diesem Ansatz folgend, brauchen Sprachentwickler nicht zu wissen, wie 3D-Grafiken und die Interaktion mit ihnen implementiert wird, da das Generatorsystem dies für die Implementierung jeder Sprache automatisch erledigt.

Diese Arbeit präsentiert Methoden und Werkzeuge zur Implementierung visueller 3D-Sprachen, die in dem Generatorsystem DEViL3D (*De*velopment *E*nvironment for *Vi*sual *L*anguages in *3D*) gebündelt sind. DEViL3D bekommt als Eingabe eine Menge von Spezifikationen hohen Niveaus und generiert daraus einen grafischen 3D-Strukturediteor als Frontend der 3D-Sprachimplementierung. Sogenannte *visuelle Muster* sind das Instrument, welches Interaktions-, Layout- und Repräsentationstechniken kapselt. Der Entwickler einer bestimmten 3D-Sprache muss lediglich die visuellen Muster auf Konstrukte der abstrakten Syntax anwenden, die die, der Sprache zugrundeliegende, Struktur beschreibt. Die generierten 3D-Editoren stellen Interaktions-, Navigations- und Layouttechniken bereit, die auf die Bedürfnisse der Sprache zugeschnitten sind.

# Abstract

Visual languages are beneficial particularly for domain-specific applications. They support graphical metaphors of their domain, which enable domain experts to use their conventional way of description. Most visual languages use two-dimensional representations, but for some purposes, using the third dimension is advantageous or necessary. A representation in three dimensions is useful for *inherently 3D languages*, like the *ball-and-stick* models of molecules. Further 3D languages assign a *semantic meaning* to each dimension or *overcome limitations in 2D arrangements*.

To construct visual diagrams of a particular domain, dedicated graphical structure editors are used, which offers specialized interaction and navigation techniques. The implementation of three-dimensional visual languages and their editors requires a wide range of conceptional and technical knowledge in 3D graphics and compiler construction. The development of graphical editors for domain-specific visual 3D languages can be simplified by using generator frameworks. Following this approach, language designers do not need to know about implementation of 3D graphics and interaction with them, because the generator framework supports this for each language implementation automatically.

This thesis presents methods and tools to implement visual 3D languages that are incorporated in the generator framework DEViL3D (*De*velopment *E*nvironment for *Vi*sual *L*anguages in *3D*). DEViL3D gets as input a set of high-level specifications and generates a dedicated 3D graphical structure editor as the front-end of a 3D language implementation. So called *visual patterns* are the vehicle to encapsulate interaction, layout and visual representation techniques. A designer of a particular 3D language only has to apply visual patterns to constructs of the abstract syntax, which defines the basic structure of the language. The generated 3D editors provide interaction, navigation, and layout techniques that are tailored to the needs of its language.

# Inhaltsverzeichnis

KAPITEL 1

# Einleitung

Visuelle Sprachen haben sich für den Entwurf von Strukturen aus domänenspezifischen Anwendungsgebieten als sehr gut geeignet erwiesen. Sie sind insbesondere für Anwendungen, die grafische Notationen nutzen, etabliert und weit verbreitet. Dies ermöglicht es Experten des jeweiligen Anwendungsgebiets, effizient Programme in der Symbolik ihrer Domäne zu erstellen, auch wenn sie keine Programmiererfahrung besitzen. Bisher basieren fast alle solche Sprachen auf zweidimensionalen Grafiken, die auf 2D-Zeichenflächen angeordnet werden. In anderen Gebieten wie Visualisierung großer Datenmengen oder Softwaresystemen [TC09], industrieller Produktplanung [DWS13] oder Filmen [Joc14] ist der Einsatz dreidimensionaler Darstellungen schon weitgehend selbstverständlich und wird gewinnbringend angewandt.

Eine verstärkte Verwendung dreidimensionaler visueller Sprachen wurde 1987 von Glinert postuliert [Gli87]. In den darauf folgenden Jahren wurden mit Cube [Naj96], SAM [GMR98] oder 3D-PP [OT99] einige 3D-Programmiersysteme konzipiert und manuell implementiert, die die dritte Dimension auf verschiedene Weise nutzen. In der letzten Dekade hingegen hat es kaum neue Entwicklungen in diesem Bereich gegeben. Nichtsdestotrotz gibt es einige Anwendungsgebiete für visuelle Sprachen, für die die Verwendung dreidimensionaler Grafiken nützlich ist. Dies gilt für Domänen, deren Notation inhärent dreidimensional ist, wie z. B. die *Kugel-Stäbchen-Modelle* von Molekülen. Weiterhin gibt es Sprachen, die den Dimensionen des 3D-Raums zusätzliche semantische Informationen zuordnen. Dieser Ansatz ermöglicht auch zweidimensionale Diagramme gemäß einer definierten Ordnungsrelation im 3D-Raum aufzuspannen.

Visuelle Sprachen sind häufig speziell auf die visuellen Notationen eines Anwendungsgebiets zugeschnitten, es handelt sich also meist um *domänenspezifische*

*Sprachen* (DSLs). Visuelle Diagramme einer bestimmten Sprache werden häufig mit sogenannten *Strukturediteren* konstruiert. Der Kreis der Nutzer für solche Sprachen bzw. deren Editoren ist gewöhnlich recht klein. Die Entwicklung eines auf eine Sprache zugeschnittenen Editors ist nur dann gerechtfertigt, wenn der Aufwand dafür angemessen gering ist. Deshalb haben sich für zweidimensionale visuelle Sprachen effektive generierende Systeme etabliert. Solch automatisierte Entwicklungswerkzeuge sind mir für 3D-Sprachen nicht bekannt. Aus diesem Grund soll in dieser Arbeit untersucht werden, welche Methoden und Werkzeuge notwendig sind, um die Entwicklung von Editoren für dreidimensionale visuelle Sprachen zu automatisieren. Dabei soll ein breites Spektrum an charakteristischen Sprachstilen mit deren typischen Sprachkonstrukten unterstützt werden.

## 1.1 Ziele

In dieser Arbeit soll untersucht werden, wie das Beschreibungsmittel visuelle Sprachen durch Nutzen der dritten Dimension verbessert werden kann und für welche Arten von Sprachen der Einsatz der dritten Dimension besonders vorteilhaft ist. Dabei sollen charakteristische Sprachstile und typische Sprachkonstrukte identifiziert werden. In diesem Zusammenhang muss auch systematisch untersucht werden, für welche Arten von Domänen der Einsatz dreidimensionaler visueller Sprachen besonders vorteilhaft ist.

Vergleicht man die Konstruktion visueller Diagramme in der zweidimensionalen Ebene und im dreidimensionalen Raum, fällt auf, dass der Konstruktionsprozess im 3D-Raum prinzipiell anders und komplexer ist. Zur Konstruktion von 2D-Diagrammen betrachtet der Anwender die Ebene aus der dritten Dimension heraus, wohingegen bei der Konstruktion in 3D der Anwender selbst Teil der 3D-Szene ist. Daraus ergibt sich eine fundamental andere Art der Interaktion mit dem visuellen Diagramm. In 2D kann das Diagramm nur durch Scrollen entlang zweier Dimensionsachsen verschoben werden. Ein vollständig neuer Aspekt in 3D hingegen sind spezielle Navigationstechniken, die es dem Anwender ermöglichen, sich im 3D-Raum zurechtzufinden. Da in 3D-Editoren der Anwender gewissermaßen selbst Teil der Szene ist, hat dies die gleichen Konsequenzen wie in der realen Welt: Sprachkonstrukte können sich gegenseitig verdecken oder sehen in Abhängigkeit des Blickwinkels unterschiedlich aus. Um dies zu kompensieren, sind Navigationstechniken, die das Einfügen, Selektieren und Modifizieren von Sprachkonstrukten unterstützen, für 3D-Spracheditoren elementar. Ein für 3D neuer Aspekt ist auch das Ineinanderschachteln von Sprachkonstrukten, sodass diese – wie bei russischen Matrjoschka-Puppen – vollständig umschlossen werden. Auch dafür müssen Techniken entworfen werden, die ein effizientes Interagieren mit eingeschachtelten Objekten ermöglichen.

Es müssen die oben angedeuteten Anforderungen an 3D-Editoren herausgearbeitet werden, um daraus gut bedienbare 3D-Spracheditoren zu entwickeln. Dafür müssen existierende 3D-Editoren untersucht werden, die der Konstruktion von 3D-Strukturen dienen, auch wenn es nicht Strukturediteren im Sinne von Sprachimplementierungen sind. Ziel ist, Techniken zur Konstruktion, Darstellung und Manipulation dreidimensionaler Konstrukte sowie zur Navigation in dreidimensionalen Darstellungen zu identifizieren.

In der Arbeit soll die Methode der *visuellen Muster*, mit denen sich Darstellungseigenschaften zweidimensionaler visueller Sprachen beschreiben lassen, auf die Spezifikation dreidimensionaler Sprachen übertragen werden. Es ist das Ziel, einen auch für dreidimensionale Sprachen möglichst vollständigen Satz an visuellen Mustern zu identifizieren und zu implementieren. Die visuellen Muster sollen neben Interaktionstechniken, wie auch im 2D-Fall, Layoutberechnungen kapseln. Das Layout visueller Diagramme entscheidet über die Platzierung seiner grafischen Elemente, wobei für den dreidimensionalen Anwendungsfall zu überlegen ist, physikalische Effekte wie die Gravitation in das Layout einzubeziehen.

Auf bisherigen Erfahrungen bei der Entwicklung des Generatorsystems DEViL [Sch06] soll aufgebaut werden, um in dieser Arbeit Methoden und Werkzeuge zu entwickeln, welche das notwendige Expertenwissen kapseln und die Entwicklung und Implementierung dreidimensionaler visueller Sprachen weitgehend automatisieren.

## 1.2 Beiträge

Diese Arbeit leistet verschiedene methodische Beiträge zur Erforschung dreidimensionaler visueller Sprachen. Bisher publizierte 3D-Sprachen werden vorgestellt und systematisch analysiert. Ich stelle ein Klassifikationsschema für 3D-Sprachen vor, welches anhand verschiedener Kategorien 3D-Sprachen bewertet. Insgesamt lässt sich damit die Frage beantworten, zu welchem Zweck die dritte Dimension in einer Sprache verwendet wird und letztendlich sogar ableiten, ob der Einsatz der dritten Dimension gewinnbringend ist. Anhand dieses Schemas werden die zuvor vorgestellten 3D-Sprachen klassifiziert.

In der Arbeit wird gezeigt, dass das zur Beschreibung visueller Darstellungen in zweidimensionalen visuellen Sprachen etablierte Konzept und Beschreibungsmittel der visuellen Muster auch für 3D-Sprachen anwendbar ist. Bei der bisherigen Entwicklung visueller Muster spielte die Anzahl der zur Verfügung stehenden Dimensionen keine Rolle, da die Muster für zweidimensionale Darstellungen entwickelt wurden. In der Arbeit formuliere ich abstrakte Musterbeschreibungen, die von der zugrundeliegenden Anzahl der Dimensionen abstrahieren. Aus diesen

lassen sich alle konkreten visuellen Muster für zweidimensionale als auch neue Muster für dreidimensionale visuelle Sprachen ableiten.

Die visuellen Muster kapseln verschiedene Methoden zum Layout der Sprachkonstrukte, die z. B. deren Nichtdurchdringung sicherstellen oder umschließende Sprachkonstrukte automatisch ausdehnen, wenn sich der Platzbedarf eingeschachtelter Konstrukte erhöht. Einige Sprachen erfordern allerdings ein individuelles sprachspezifisches Layout, welches von den Standard-Layoutmethoden nicht geleistet werden kann. In der Arbeit präsentiere ich eine Methode, mit der sprachspezifische Layoutanforderungen umgesetzt werden können.

Es gibt zahlreiche 3D-Editoren, die meist von Hand entwickelt und auf spezifische Anwendungszwecke zugeschnitten sind. In dieser Arbeit zeige ich, wie sich 3D-Spracheditoren für prinzipiell beliebige 3D-Konstrukte generieren lassen. Die Funktionalität dieser Editoren hinsichtlich Interaktion ist speziell auf die Anforderungen der Sprache zugeschnitten und bietet eine ähnliche Qualität, wie sie von handimplementierten Editoren bekannt ist. In diesem Zusammenhang sind gute Interaktions- und Navigationstechniken, die das Konstruieren von 3D-Diagrammen direkt im 3D-Raum erlauben, von besonderer Bedeutung. In diesem Bereich leistet die Arbeit folgende Beiträge:

- Es werden verschiedene in etablierten 3D-Editoren verwendete Interaktionstechniken identifiziert und für 3D-Spracheditoren generisch verfügbar gemacht. Die Interaktionstechniken werden in visuellen Mustern gekapselt und sind so unmittelbar an die Erfordernisse der Sprachkonstrukte angepasst.

- Ein für 3D-Sprachen neuer Aspekt ist die Notwendigkeit von 3D-Navigationstechniken, die verwendet werden, um die 3D-Szene aus verschiedenen Blickwinkeln zu betrachten. Es wird im Wesentlichen zwischen *Wayfinding*- und *Travelling*-Techniken unterschieden, für die jeweils mehrere Techniken konzipiert werden.

- Eine für viele 3D-Sprachen charakteristische Eigenschaft ist das Ineinanderschachteln von Sprachkonstrukten, sodass sich diese vollständig umschließen. Es werden Techniken entwickelt, die eine effiziente Interaktion mit eingeschachtelten Konstrukten sicherstellen.

- Es wird untersucht, inwieweit stereoskopische Darstellungen der 3D-Szene in 3D-Spracheditoren den Tiefeneindruck verbessern und so dem Anwender die Interaktion mit der Szene erleichtern.

Zur praktischen Erprobung der oben genannten Aspekte ist das Generatorsystem *DEViL3D* (*De*velopment *E*nvironment for *Vi*sual *L*anguages in *3D*) konzipiert

und implementiert worden. Das System unterstützt die oben beschriebenen Methoden und generiert aus Spezifikationen für 3D-Sprachen vollautomatisch funktionsfähige 3D-Strukturediteren. In der Arbeit wird ein breites Spektrum an 3D-Sprachen präsentiert, für die mit DEViL3D Strukturediteren generiert wurden. Die von den Strukturediteren bereitgestellten Interaktions- und Navigationstechniken sind auch Gegenstand einer ausführlichen Evaluation, die zeigt, dass die Editoren gut benutzbar sind.

## 1.3 Struktur der Arbeit

Das nachfolgende zweite Kapitel stellt die Grundlagen dieser Arbeit vor. Dies umfasst insbesondere – neben einer kurzen allgemeinen Einführung in visuelle Sprachen – die Vorstellung schon existierender dreidimensionaler visueller Sprachen. An dieser Stelle wird auch das Klassifikationsschema für 3D-Sprachen vorgestellt und auf die zuvor präsentierten Sprachen angewendet. Auch Editoren für dreidimensionale Anwendungen und 3D-Visualisierungen werden vorgestellt. Weiterhin beschreibe ich Generatorsysteme für zweidimensionale visuelle Sprachen, insbesondere das DEViL-System.

Kapitel 3 fungiert als Überblicks- und Einführungskapitel in den Themenkomplex eines Generatorsystems für dreidimensionale Sprachen. Dort werden die Besonderheiten von 3D-Sprachen herausgestellt, dann werden aus Sicht eines Sprachentwicklers die notwendigen Schritte zur Spezifikation einer 3D-Sprache mit DEViL3D vorgestellt sowie eine Systemübersicht des DEViL3D-Systems gegeben.

Das vierte Kapitel widmet sich dem für das Generatorsystem zentralen Konzept der dreidimensionalen visuellen Muster. Zunächst stelle ich die abstrakten Musterbeschreibungen vor, ehe ich auf die wichtigsten Aspekte der daraus abgeleiteten konkreten visuellen Muster eingehe. Weiterhin präsentiere ich eine Methode, mit der Sprachentwickler mit DEViL3D das visuelle Erscheinungsbild von Sprachkonstrukten bestimmen können. Es wird außerdem argumentiert, dass sich die visuellen Repräsentationen der in dieser Arbeit vorgestellten 3D-Sprachen mit der Menge der entwickelten visuellen Muster erfolgreich beschreiben lassen.

In Kapitel 5 beschreibe ich Layouteigenschaften dreidimensionaler Sprachen. Dies umfasst u. a. das Schachteln von Sprachkonstrukten und auf Constraints basierendes Layout. Zentral sind sprachspezifische Layouteigenschaften und Methoden, wie der Sprachentwickler diese für eine spezielle Sprache umsetzen kann.

Das sechste Kapitel widmet sich der Interaktion und Navigation in Strukturediteren für dreidimensionale Sprachen. Die Interaktionsmechanismen unterteilen sich in das Einfügen, Selektieren und Manipulieren von Sprachkonstrukten. Sie

sind in den visuellen Mustern gekapselt und so für die visuelle Repräsentation der einzelnen Sprachkonstrukte maßgeschneidert. Die Navigation ist ein bei 3D-Sprachen neu auftretender Aspekt. Es werden verschiedene Wayfinding- und Travelling-Techniken vorgestellt.

Im siebten Kapitel evaluiere ich das Generatorsystem DEViL3D. Dies erfolgt zweigeteilt: Zunächst betrachte ich die Usability des Generators, was die Spezifikation von Beispielsprachen und deren Komplexität umfasst, bevor ich auf die Usability der generierten Editoren eingehe. Dafür habe ich kontrollierte Experimente durchgeführt, um Interaktions- und Navigationsmechanismen zu vergleichen. Ich stelle die Experimente samt statistischer Auswertung vor.

Kapitel 8 fasst die wichtigsten Ergebnisse und Erkenntnisse der vorliegenden Arbeit zusammen und gibt einen Ausblick auf lohnenswerte Erweiterungen von DEViL3D.

## 1.4 Publikationen des Autors mit Bezug zu dieser Arbeit

Die Arbeit an dem Generatorsystem für dreidimensionale Sprachen wurde von der Deutschen Forschungsgemeinschaft (DFG) in dem Projekt *Methoden und Werkzeuge zur Entwicklung dreidimensionaler visueller Sprachen* über einen Großteil des Bearbeitungszeitraums hinweg gefördert. Teilergebnisse wurden bei folgenden Konferenzen, Workshops und Journals präsentiert.

Einen ersten motivierenden Beitrag zu dreidimensionalen Sprachen im Allgemeinen und einen Ausblick auf ein Generatorsystem wurde 2011 unter dem Titel *Towards Three-Dimensional Visual Languages* [WCK11b] auf der Konferenz *Compilers, Programming Languages, Related Technologies and Applications (CoRTA)* vorgestellt.

Einen konkreten Überblick über das Generatorsystem DEViL3D gibt das Papier *DEViL3D – A Generator Framework for Three-Dimensional Visual Languages* [Wol12], welches ein Jahr später auf dem *International Workshop on Visual Languages and Computing (VLC)* präsentiert wurde.

Das Konzept dreidimensionaler generischer Darstellungen, mit dem das visuelle Erscheinungsbild von Sprachkonstrukten definiert wird, wurde 2013 in dem Papier *Specifying Generic Depictions of Language Constructs for 3D Visual Languages* [Wol13a] auf dem *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* präsentiert. In dieser Arbeit sind Aspekte aus dem Papier in den Abschnitten 4.7 und 5.3.1 nachzulesen.

Ein Jahr später wurde ebenfalls auf der *VL/HCC* das Poster-Papier *Layout Requirements of a 3D Molecular Editor Specified with DEViL3D* [Wol14] vorgestellt, welches beschreibt, wie Sprachentwerfer sprachspezifische Layoutvorgaben für ihre Sprache mit DEViL3D spezifizieren können. Dies wurde anhand des Beispiels für Molekülmodelle beschrieben und ist in dieser Arbeit in Abschnitt 5.5 zu finden.

Ebenfalls 2014 wurde auf dem *7th International Symposium on Visual Information Communication and Interaction (VINCI)* das Papier *Encapsulating Interaction Techniques of 3D Language Editors in Visual Patterns* [WK14] präsentiert, welches Interaktions- und Navigationstechniken von mit DEViL3D generierten Editoren beschreibt. Derartige Aspekte finden sich in dieser Arbeit in Kapitel 6. Weiterhin werden in dem Papier Interaktions- und Navigationstechniken evaluiert. Die Ergebnisse der Evaluation sind in Abschnitt 7.3 zu finden.

Das letztgenannte Papier wurde als eines der besten Papiere der Konferenz VINCI 2014 ausgewählt, um in einer erweiterten Fassung im *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)* veröffentlicht zu werden. Der erweiterte Artikel *Generating 3D Visual Language Editors: Encapsulating Interaction Techniques in Visual Patterns* [WK15] beschreibt viele Grundlagen und DEViL3D ausführlicher und umfasst als neue Inhalte die Untersuchung der Komplexität von Sprachspezifikationen, die stereoskopische Darstellung der 3D-Szene eines mit DEViL3D generierten Strukturediteors sowie deren Evaluation. In dieser Arbeit sind diese Aspekte in den Abschnitten 7.2, 6.8 und 7.3.8 zu finden.

KAPITEL 2

# Grundlagen

In diesem Kapitel stelle ich die Grundlagen dieser Arbeit vor. Der erste Abschnitt beschäftigt sich zunächst mit visuellen (zweidimensionalen) Sprachen und Strukturediteren. In Abschnitt 2.2 folgt eine Einführung in dreidimensionale visuelle Sprachen, die den Leser mit Beispielsprachen aus der Literatur vertraut macht. Im darauf folgenden Abschnitt werden diese Sprachen klassifiziert und ihr Nutzen hinsichtlich der dritten Dimension herausgearbeitet. Der Abschnitt 2.3 stellt ausgewählte 3D-Editoren vor, um anhand dieser zu ermitteln, welche Standards es hinsichtlich Interaktion und Navigation in 3D-Desktop-Anwendungen gibt. In Abschnitt 2.6 werden Generatorsysteme für visuelle Sprachen vorgestellt. Der Fokus liegt dabei auf dem DEViL-System, dessen Konzepte eine wichtige Grundlage für die Entwicklung eines Generatorsystems für 3D-Sprachen sind. Weiterhin werden die Generatorsysteme DiaGen/DiaMeta vorgestellt, in deren Kontext bereits erste prototypische Ansätze zur Generierung von 3D-Sprachen verfolgt wurden. Abschließend folgt eine Zusammenfassung, die die Erkenntnisse dieses Kapitels in den Gesamtkontext einordnet.

## 2.1 Visuelle Sprachen und Strukturediteren

Dieser Anschnitt führt in das Themengebiet visueller Sprachen ein und gibt Definitionen aus der Literatur dafür an. Anschließend wird auf Eigenschaften von Strukturediteren eingegangen, mit denen Sätze visueller Sprachen konstruiert werden können. Schiffer hat sich ausführlich mit der Terminologie visueller Sprachen auseinandergesetzt und definiert visuelle Sprachen wie folgt [Sch98, S. 64]:

> Eine *visuelle Sprache* ist eine formale Sprache mit visueller Syntax oder visueller Semantik und dynamischer oder statischer Zeichengebung.

Dem Terminus *visuell* stellt er folgende Definition voran [Sch98, S. 63]:

> *Visuell* ist die Bezeichnung für jene Eigenschaft eines Objekts, durch die mindestens eine Information über das Objekt, die für das Erreichen eines Handlungsziels unverzichtbar ist, nur durch das visuelle Wahrnehmungssystem des Menschen gewonnen werden kann.

In der Definition sind mit *Objekt* sowohl abstrakte als auch konkrete Gegenstände gemeint. Eigenschaften des Objekts, die nur durch das visuelle Wahrnehmungssystem interpretiert werden können, umfassen u. a. Farben, Formen, Verbindungen, Überlagerungen und Berührungen. Die in der ersten Definition erwähnte formale Sprache besteht aus einer Menge von Sprachkonstrukten, Regeln zur Bildung von Ausdrücken, bestehend aus solchen Konstrukten, sowie Interpretationsregeln zur Ermittlung der Semantik. Ausdrücke, die nicht der definierten Syntax genügen, sind ungültig und solche, die nicht der definierten Semantik genügen, bedeutungslos. Mit *statischer Zeichengebung* ist die persistente Speicherung eines Programms bzw. Diagramms in einer Datei gemeint. Die *dynamische Zeichengebung* hingegen beruht auf flüchtigen Vorgängen, wie sie in Systemen vorkommen, die das Programmierparadigma *Programming by Example* verfolgen.

Eine alternative Definition, die darüber hinaus visuelle Sprachen von textuellen abgrenzt, stammt von Meyers [Mye90] (Übersetzung nach Schiffer [Sch98, S. 40]):

> *Visuelle Programmierung* (VP) bezieht sich auf jedes System, das dem Anwender erlaubt, ein Programm auf zwei- (oder mehr-) dimensionale Weise zu spezifizieren. Obwohl das eine sehr breite Definition ist, werden konventionelle textuelle Sprachen nicht als zweidimensional betrachtet, weil Compiler oder Interpretierer sie als lange, eindimensionale Ströme verarbeiten.

Diese Definition ist hinsichtlich der genauen Eigenschaften von visuellen Objekten ungenauer, bringt allerdings den Aspekt ins Spiel, dass Objekte eines visuellen Programms auch dreidimensional sein dürfen. Alle in dieser Arbeit behandelten dreidimensionalen visuellen Sprachen sind also von dieser Definition abgedeckt. Weiterhin erwähnt Meyers textuelle Sprachen, deren Programme für ihn immer eindimensionale Ströme sind. Dass dies nicht zwangsläufig so sein muss, beschreibt Schmidt [Sch06, S. 9], indem er Python als eine Sprache anführt, bei der Zeileneinrückungen Semantik tragen. Zur Abgrenzung von solch visuellen Eigenschaften textueller Sprachen beschreibt Schmidt visuelle Sprachen, die zu einem Großteil aus visuellen Konstrukten bestehen, als *echt visuell*.

Der Anwendungsbereich von visuellen Sprachen erstreckt sich meist auf eine ganz bestimmte Domäne. Besonders für Domänen die auf visuellen Notationselementen aufbauen, deren Syntax und Semantik dem Domänenexperten vertraut

sind, können einfach Implementierungen visueller Sprachen entworfen werden. Diese greifen die wohlbekannten Notationselemente auf und stellen einen *Strukturediteor* bereit, mit dem der Domänenexperte strukturiert Diagramme seiner Domäne editieren kann. Ein solcher Konstruktionsprozess ist ein charakteristisches Merkmal visueller Sprachen und grenzt diese von Visualisierungen (siehe Abschnitt 2.5) ab, die visuelle Darstellungen aus einer Datenmenge ableiten.

Visuelle Sprachen werden in einer großen Bandbreite von Domänen eingesetzt und folgen dabei verschiedenen Paradigmen und Eigenschaften.[1] Eine dahingehende Übersicht und Klassifikation von Publikationen zu visuellen Sprachen, die aber auch als Klassifikation für die Sprachen selber aufgefasst werden kann, wurde von Burnett und Baker [BB94] entwickelt. Die Klassifikation wurde über Jahre gepflegt und weiterentwickelt, wobei die aktuellste Version aus dem Jahr 2009 stammt und im Internet eingesehen werden kann.[2]

Im Folgenden werde ich etwas näher auf die oben genannten Strukturediteoren eingehen, die ein strukturiertes Konstruieren visueller Diagramme ermöglichen. Im Wesentlichen folge ich dabei den Darstellungen von Schmidt [Sch06, S.25ff.] und Jung [Jun00, S. 46ff.]. Ich fasse deren Erkenntnisse kurz zusammen und gebe – wo inhaltlich passend – Vorwärtsreferenzen zu nachfolgenden Kapiteln, die den entsprechenden Aspekt für dreidimensionale Sprachen behandeln.

Abstrakt gesehen ist ein Strukturediteor ein Programm, mit dem eine anwendungsspezifische Datenstruktur manipuliert werden kann. Der Benutzer manipuliert die Datenstruktur anhand einer konkreten Repräsentation. Um unterschiedliche Aspekte der anwendungsspezifischen Datenstruktur zu realisieren, kann ein Strukturediteor mehrere *Sichten* anbieten. Um Sachverhalte gleicher Detailebene auf unterschiedliche Weise darzustellen, werden eine Menge *paralleler Sichten* verwendet. Eine gröbere bzw. verfeinerte Darstellung wird in *Übersichts-* bzw. *Detailsichten* realisiert. All diese Sichten umfassen Interaktionsmechanismen, die dem Benutzer z. B. andeuten an welcher Stelle ein Sprachkonstrukt einzufügen ist und so ein strukturiertes Editieren ermöglichen. Dadurch wird sichergestellt, dass das visuelle Diagramm jederzeit syntaktisch korrekt ist.

Schmidt [Sch06, S. 26f.] hat ein *funktionales Grundmodell* zur Sprachimplementierung entworfen, welches auch die Weiterverarbeitung der im Strukturediteor konstruierten Diagramme berücksichtigt. Das Modell unterscheidet grundlegend zwischen der *editierbaren Struktur* und der *semantischen Struktur*, die es ermöglichen, zwischen der Darstellung und der zugrundeliegenden Information zu un-

[1] Dies umfasst sowohl Eigenschaften der visuellen Sprache und ihrer visuellen Repräsentation – wie z. B. ob es sich um datenflussbasierte oder Diagrammsprachen handelt – als auch Charakteristika des Spracheinsatzes, wie z. B. Datenbanksprachen oder Sprachen zur Programmierung von Web-Anwendungen.

[2] `http://web.engr.oregonstate.edu/~burnett/vpl.html`

terscheiden. Aspekte der editierbaren Struktur wurden schon im letzten Absatz angesprochen und umfassen die konkreten Informationen einer visuellen Diagrammrepräsentation, wie z. B. Layoutentscheidungen des Editornutzers. Die semantische Struktur abstrahiert von Informationen der Diagrammdarstellung, stellt nur den semantisch relevanten Informationsgehalt des Diagramms dar und vermeidet Redundanzen.

Schmidt klassifiziert Strukturediteren basierend auf dem funktionalen Modell [Sch06, S. 29ff.]. Dafür führt er sechs orthogonale Dimensionen ein, die Strukturediteren erschöpfend charakterisieren. Bei der ersten Dimension handelt es sich um die *Art der konkreten Repräsentation*, die Eigenschaften visueller Sichten und der Diagrammrepräsentation beschreibt. Als eine Darstellungsart werden *dreidimensionale Repräsentationen* erwähnt. Da dies zentraler Inhalt dieser Arbeit ist, finden sich erste Beispiele dafür im Abschnitt 2.2. Kapitel 4 beschreibt, wie 3D-Repräsentationen wiederverwendbar gekapselt werden können. Die Dimension *Layoutfreiheiten* beschreibt, inwieweit der Editorbenutzer Darstellungsdetails beeinflussen kann. In Kapitel 5 werden Layouteigenschaften für 3D-Sprachen diskutiert. Innerhalb der Dimension *Interaktionsmechanismen* werden Techniken zur Konstruktion von visuellen Diagrammen erörtert. Für dreidimensionale Sprachen sind neue Mechanismen – insbesondere zur Navigation – erforderlich, die in Kapitel 6 dargestellt werden. Das *Niveau der editierbaren Struktur* beschreibt Niveauunterschiede zwischen den Konzepten der editierbaren Struktur und denen der semantischen Struktur. Mit der *Schärfe der Editorsyntax* wird der Spielraum zwischen syntaktisch korrekten und semantisch korrekten Diagrammen beschrieben. Die letzte Dimension behandelt die *Anzahl der Repräsentanten eines semantischen Objekts*, die z. B. bei parallelen Sichten größer als eins sein kann.

Zur Konstruktion von visuellen Diagrammen werden neben Strukturediteren, wie sie hier vorgestellt wurden, auch Editoren eingesetzt, in denen das visuelle Diagramm durch Freihandzeichnungen erstellt und anschließend geparst wird (vgl. [Bri09]). Dies hat allerdings den Nachteil, dass die syntaktische Korrektheit nicht jederzeit sichergestellt werden kann. Brieler zitiert sogar erste Ansätze [Bri09, S. 31], die dreidimensionale Konstrukte durch Freihandzeichnungen erzeugen. Der in dieser Arbeit beschriebene Ansatz zur Konstruktion von 3D-Sprachen möchte allerdings von den Vorteilen klassischer Strukturediteren profitieren, weshalb Freihandzeichnungen nicht in Frage kommen.

## 2.2 3D-Sprachen

Die erste Idee für visuelle dreidimensionale Sprachen geht auf Glinert [Gli87] zurück. Er formulierte 1987 eine allgemeine Motivation und ein Vorgehen, wie sich die von ihm entwickelte *BLOX-Methodik* auf 3D erweitern lässt. In den darauf

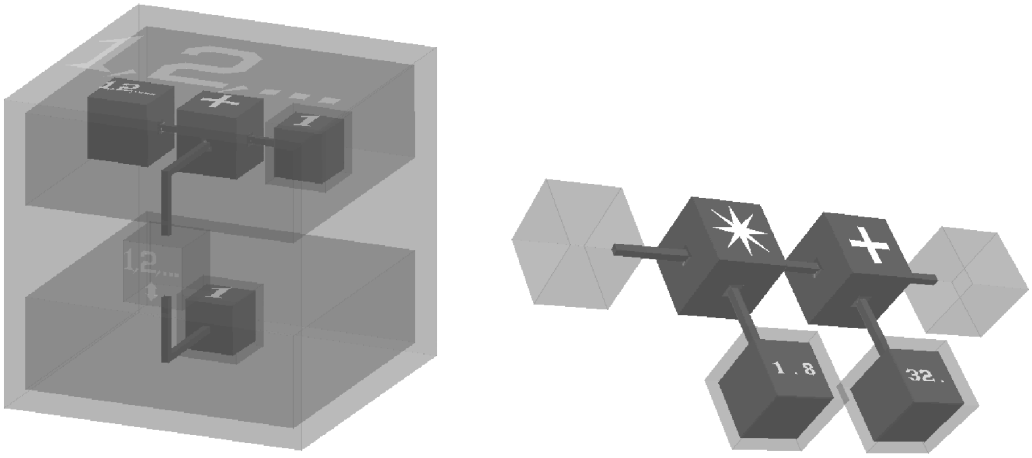
folgenden Jahren gab es von verschiedenen Wissenschaftlern Implementierungen und Ideen für 3D-Sprachen, die ich in diesem Abschnitt kurz präsentiere. Die Vorstellung erfolgt in grob chronologischer Reihenfolge. Eine Klassifikation und Einordnungen der 3D-Sprachen ist im nachfolgenden Abschnitt 2.3 zu finden.

### 2.2.1 Cube

*Cube* [Naj96] ist eine visuelle dreidimensionale Datenflusssprache und basiert auf der logischen Programmiersprache Prolog. Cube visualisiert Sprachkonstrukte wie Prädikate, benutzerdefinierte Datentypen oder einfache Werte als Würfel, die über Verbindungen, die den Datenfluss angeben, miteinander verbunden sind.

Typen und Werte werden durch verschiedenfarbige Würfel unterschieden, die ineinander verschachtelt werden können. Beinhaltet ein Würfel einen anderen, wird dieser als *Inhaber-Würfel* bezeichnet und halbtransparent dargestellt. Ein Inhaber-Würfel kann einen Term enthalten und ist deshalb mit Variablen in textuellen Sprachen vergleichbar. Sind zwei Würfel, die je einen weiteren beinhalten, miteinander verbunden, wird ihr Inhalt per Unifikation kombiniert. Die Anordnung von Würfeln zueinander hat in Cube eine semantische Bedeutung. Vertikal angeordnete Würfel drücken eine Disjunktion aus; eine Konjunktion wird durch eine horizontale Anordnung erreicht.

In Abbildung 2.1a ist ein *Prädikatdefinitions-Würfel* dargestellt, der einen Generator für natürliche Zahlen beschreibt, die rekursiv wie folgt definiert werden: (a) 1 ist eine natürliche Zahl und (b) falls $n$ eine natürliche Zahl ist, so gilt dies auch für $n + 1$. Teil (a) der Definition wird in dem Würfel durch die untere Ebene definiert,




**(a)** Generator für natürliche Zahlen. **(b)** Programm zur Temperaturumrechnung.

**Abbildung 2.1:** Bildschirmfotos zweier Cube-Konstrukte; Quelle: [Naj94].

die einen Würfel mit dem Wert 1 beinhaltet. Dieser ist mit einem transparenten Anschluss-Würfel, der sich zwischen den beiden Ebenen befindet und den formalen Parameter der Definition repräsentiert, verbunden. Teil (b) der Definition ist in der oberen Ebene dargestellt und enthält einen Additionsprädikat-Würfel, der ebenfalls mit dem Anschluss-Würfel verbunden ist. Dieser sorgt für die Addition der zuvor berechneten natürlichen Zahl (dargestellt durch die rekursive Anwendung des Prädikat-Würfels) und den Wert 1.

Abbildung 2.1b zeigt ein Cube-Programm[3], welches einen Temperaturwert zwischen Fahrenheit und Celsius umrechnet. Diese Umrechnung ist durch die Formel $F = 1,8 \cdot C + 32$ definiert. Das Cube-Programm enthält zwei Prädikat-Würfel zur Multiplikation und Addition sowie zwei Würfel, die die in der Formel vorkommenden Werte 1,8 und 32 enthalten. Der gegebene Temperatur-Wert kann in einen der beiden leeren Würfel eingesetzt werden, aus dem das Programm dann den gesuchten Wert berechnet.

### 2.2.2 SAM

*SAM* (Solid Agents in Motion) [GMR98] ist eine parallele, synchrone und zustandsorientierte Programmiersprache. Ein SAM-Programm besteht aus einer Menge von Agenten, die über Nachrichten miteinander kommunizieren. Für diese Agenten und Nachrichten gibt es sowohl eine abstrakte als auch eine konkrete 3D-Darstellung. Die konkrete Darstellung stammt direkt aus der Anwendungsdomäne, wohingegen die abstrakte Darstellung das Verhalten eines Agenten spezifiziert und eine Weiterentwicklung der 2D-Sprache *Pictorical Janus* darstellt. Ein Beispiel für eine solche Spezifikation eines Agenten ist in Abbildung 2.2 zu sehen. Ein Agent besteht aus einer beliebigen Anzahl von Eingangs- und Ausgangsports sowie aus Nachrichten, die über diese Ports verschickt werden. Das Verhalten, also wann und an wen Nachrichten verschickt werden, wird durch die Angabe von Regeln

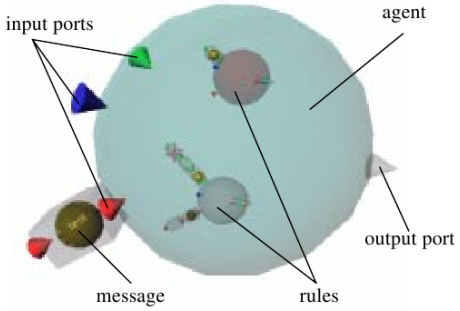



**Abbildung 2.2:** Abstrakte Repräsentation eines SAM-Agenten; Quelle: [GMR98].

[3] In diesem Fall spreche ich von Programm, da das visuelle Diagramm ausführbar ist.

ausgedrückt, die aus einer Vorbedingung und einer Sequenz von Aktionen bestehen. Eine Ausführung eines SAM-Programms kann animiert werden, wodurch der Anwender das Schalten der Regeln und Verschicken der Nachrichten genau beobachten kann.

Ein Agent wird durch eine transparent dargestellte Kugel visualisiert, die an ihrer Oberfläche durch Kegel visualisierte Eingangs- bzw. Ausgangsports besitzt. Nachrichten haben in Abhängigkeit ihres Typs eine unterschiedliche visuelle Darstellung; sind sie z. B. vom Typ String, werden sie als Kugel dargestellt. Regeln werden um eine Vorbedingung und eine Folge von Aktionen erweitert, die in einer zylinderförmigen Röhre angeordnet werden. Die Regeln werden analog zu den Agenten dargestellt, da sie eine Kopie des umfassenden Agenten sind. Zur strukturellen Anordnung der Sprachkonstrukte werden in SAM prinzipiell zwei Varianten genutzt: Zum einen die freie Positionierung innerhalb einer umgebenden Hülle (Regeln innerhalb eines Agenten) und zum anderen das Andocken aneinander (Ports an Agenten).

### 2.2.3 3D-PP

*3D-PP* wird von ihren Schöpfern als eine allgemein anwendbare[4] dreidimensionale visuelle Sprache beschrieben [OT99], die auf der parallelen, logischen Programmiersprache *GHC* (Guarded Horn Clauses) basiert. Ein GHC-Programm ist deklarativ definiert und besteht aus einer Menge von Klauseln, deren Programmkonstrukte im Wesentlichen atomare Werte, Listen, Ein-/Ausgabedaten, Ziele („goal“) und eingebaute Ziele („built-in goal“) sind. Diese Programmkonstrukte werden in 3D-PP als verschiedene dreidimensionale Icons dargestellt. Ein dreidimensionales 3D-PP-Programm entsteht durch die Komposition dieser Programmkonstrukte.

GHC-Klauseln bestehen aus einer linken und rechten Seite, wie das folgende Beispiel zeigt:

```
primes(...) :- gen_primes(...), pkup(...).
```

Im 3D-PP-Programm wird dies durch hierarchisch geschachtelte Programmelemente ausgedrückt (vgl. Abbildung 2.3[5]), die entsprechend der linken und rechten Seiten einer Klausel eine Vater-Kind-Beziehung ergeben. Um bei rekursiver Schachtelung den Überblick zu behalten, werden immer nur die obersten Hierarchie-Ebenen angezeigt, wobei weitere sichtbar gemacht werden können. Das 3D-PP-Programm aus Abbildung 2.3 berechnet die 1000te Primzahl. Die Ausführung des 3D-PP-Programms kann mithilfe einer Animationskomponente visualisiert werden.

[4] engl.: *general purpose*

[5] Das Bildschirmfoto der Abbildung ist [OT99] nur in schwarz-weißer Darstellung enthalten. Der Autor Takashi Oshiba hat mir die farbige Ausführung zur Verfügung gestellt.

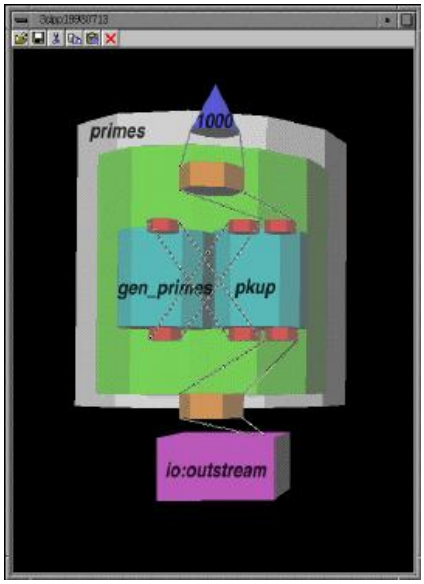



**Abbildung 2.3:** Ausschnitt aus einem 3D-PP-Programm; Quelle: [OT99].

### 2.2.4 3D-Visulan

*3D-Visulan* [Yam96] ist eine visuelle dreidimensionale regelbasierte Sprache, die sowohl Programme als auch Daten als dreidimensionale Bitmaps darstellt. Dabei ist ein Bitmap als eine Menge von verschiedenfarbigen Würfeln definiert. Die Regeln, die ein Anwender in 3D-Visulan definieren kann, bestehen aus einer linken und rechten Seite und beschreiben, wie Bitmaps transformiert werden. Die Regeln und das dreidimensionale Programm befinden sich in einer atomaren 3D-Sicht, was den Vorteil hat, dass die Regeln und ihre Anwendung gleichzeitig sichtbar sind. Regeln

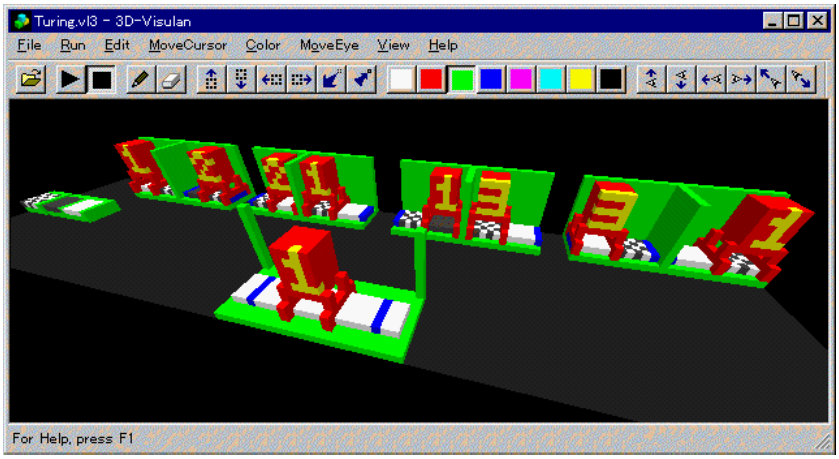


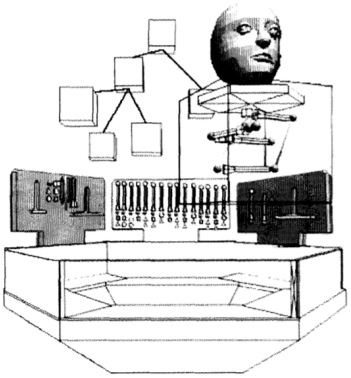

**Abbildung 2.4:** Eine Turingmaschine in 3D-Visulan; Quelle: [Yam96].

können durch Konjunktion miteinander verknüpft werden und außerdem Prioritäten besitzen, die durch ihre Position innerhalb der 3D-Welt bestimmt werden: weiter hinten liegende Regeln haben eine höhere Priorität. Ob Regeln anzuwenden sind, wird über ein Pattern-Matching-Verfahren bestimmt. Abbildung 2.4 zeigt im Vordergrund ein 3D-Programm, welches eine Turingmaschine beschreibt, und im Hintergrund vier verschiedene Regeln, nach denen sich der Schreib-/Lesekopf nach links und rechts bewegen kann.

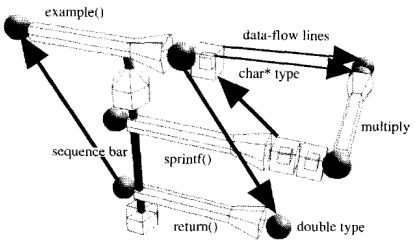
### 2.2.5 Lingua Graphica

Bei *Lingua Graphica* (LG) [SP92] handelt es sich um eine dreidimensionale visuelle Sprache zur Programmierung von virtuellen Umgebungen. Teile einer virtuellen Umgebung können in einer Art Selbstanwendung mit LG programmiert werden. LG kapselt verschiedene textuelle Programmiersprachen, die als *Basissprachen* bezeichnet werden, und ermöglicht eine zur Basissprache äquivalente Konstruktion von Programmen in 3D. Neben einer nicht näher benannten Skriptsprache gibt es mit C++ zwei in LG umgesetzte Basissprachen. Konstrukte aus C++ wie Funktionen, bedingte Ausdrücke oder Typen erhalten in Lingua Graphica eine Repräsentation durch verschiedene dreidimensionale Formen. Zur Definition zusätzlicher 3D-Formen für weitere Konstrukte oder als Alternative für vorhandene stellt LG einen speziellen Editor bereit. Dieser kann aus dem Lingua Graphica *Workspace* (siehe Abbildung 2.5a), der die Hauptsicht der virtuellen Umgebung darstellt, heraus aufgerufen werden.

Innerhalb des Workspace wird auch das dreidimensionale LG-Programm konstruiert. Abbildung 2.5b zeigt ein Beispiel mit C++ als Basissprache, wobei im unteren Teil das äquivalente C++-Programm dargestellt ist. Mit Lingua Graphica



**(a)** Lingua Graphica Workspace.




```
double example(double num, char *str)
  {
    sprintf(str," %f",(num*num));
    return(num);
  }
```

**(b)** Beispielprogramm.

**Abbildung 2.5:** Ausschnitte aus dem Lingua Graphica System; Quelle: [SP92].

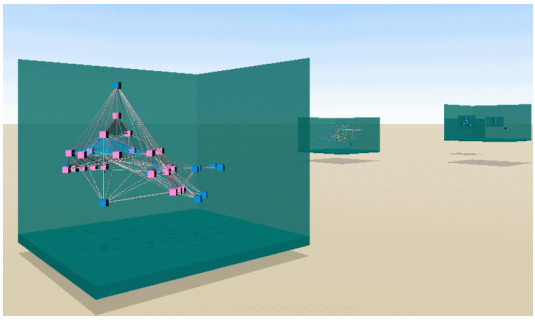
erstellte 3D-Programme können in textuelle Programme ihrer Basissprache transformiert und abgespeichert werden. Initial können auch textuelle Programme geladen werden, die dann eine dreidimensionale LG-Repräsentation erhalten.

### 2.2.6 UML in 3D

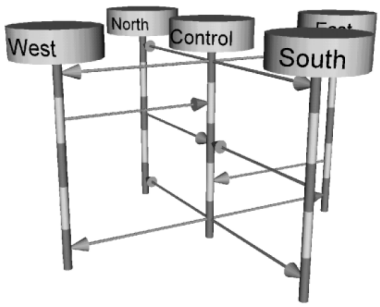
Die Darstellung von Diagrammen der *Unified Modeling Language* (UML) wurde schon von zahlreichen Wissenschaftlern aufgegriffen. Viele Ansätze können allerdings eher der 3D-Visualisierung zugeschrieben werden, da sie von bestehenden 2D-UML-Diagrammen ausgehen und die Zusammenhänge dieser 2D-Diagramme in drei Dimensionen darstellen (vgl. [PD08; GK98]). Andere fokussieren sich eher auf Aspekte der Layoutberechnung in dreidimensionalen Klassendiagrammen [Dwy01]. Einen Weg hin zu echter dreidimensionaler Darstellung machen Alfert und Fronk [AFE01; AF02] durch ihre Darstellung von Klassendiagrammen, die u. a. durch Schachtelung von 3D-Objekten eine Enthaltensein-Relation ausdrücken (vgl. Abbildung 2.6a).

Gogolla et al. [GRR99; RG00] haben sich damit beschäftigt, wie Sequenzdiagramme gewinnbringend in 3D dargestellt werden können. Das Problem, dass Nachrichtenlinien zwischen nicht benachbarten Objekten durch Lebenslinien dazwischenliegender Objekte geschnitten werden, kann durch ein Verschieben der Objekte in der dritten Raumdimension vermieden werden (siehe Abbildung 2.6b).

Neben der Durchdringungsfreiheit der Nachrichtenlinien bietet die dritte Dimension noch zwei weitere Anwendungsmöglichkeiten für Sequenzdiagramme. Da Sequenzdiagramme durch Verwendung sogenannter kombinierter Fragmente in mehrere Teildiagramme (die durch Interaktionsreferenzen miteinander verbunden sind) aufgeteilt werden können, bietet dies die Möglichkeit, das gesamte Diagramm inklusive der Fragmente im 3D-Raum übersichtlich darzustellen. Weiterhin haben Radfelder und Gogolla [RG00] die Kombination von statischen und



**(a)** 3D-Klassenstruktur; Quelle: [AFE01].




**(b)** 3D-Sequenzdiagramm; Quelle: [GRR99].

**Abbildung 2.6:** Ansätze für dreidimensionale UML-Diagramme.

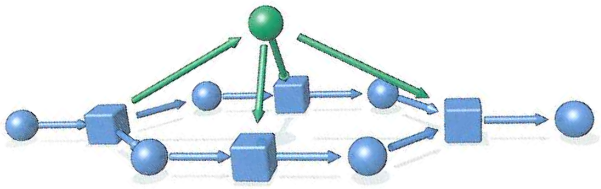


**Abbildung 2.7:** Dreidimensionales Petri-Netz; Quelle: [Röl07].

dynamischen Aspekten des Softwareentwurfs in einer einzigen Sicht betrachtet. So kann ein Klassendiagramm im Hintergrund statische Aspekte beschreiben und ein Sequenzdiagramm im Vordergrund die dynamischen. Linien entlang der $z$-Achse verbinden korrespondierende Objekte der beiden Diagramme. Die dynamischen Aspekte des Sequenzdiagramms werden außerdem durch eine Animation unterstrichen, indem Tokens entlang der Nachrichtenlinien bewegt werden.

### 2.2.7 3D-Petri-Netze

Die Idee, *Petri-Netze* in drei Dimensionen darzustellen, wurde von Rölke [Röl07] beschrieben. Die Grundidee dabei ist die Separierung von Teilaspekten eines Petri-Netzes auf verschiedene Ebenen im dreidimensionalen Raum. Dies löst nicht nur das Problem der Unübersichtlichkeit oder der sich überschneidenden Kanten, sondern gibt den im 3D-Raum angeordneten (z. B. gestapelten) Ebenen des Petri-Netzes eine eigene Semantik, indem jede Ebene einen bestimmten Teilaspekt des Petri-Netzes darstellt. Ein Petri-Netz kann z. B. zwischen Daten- und Kontrollfluss unterteilt werden.

In Abbildung 2.7[6] ist auf der unteren Ebene ein Kontrollfluss dargestellt, der eine bedingte Verzweigung und am Ende eine Zusammenführung enthält. Auf der zweiten Ebene ist eine Stelle dargestellt, die den Datenfluss repräsentiert. Die erste Transition des Datenflusses versorgt die Stelle mit Daten, die von allen anderen Transitionen gelesen werden kann.

### 2.2.8 AgentCubes

*AgentCubes* [IRW09] ist eine Weiterentwicklung von *AgentSheets* und ein System, mit dem Endanwender unterstützt werden mit möglichst geringem Aufwand 3D-Spiele zu programmieren. Diese Spiele basieren auf Agenten, deren autonomes Verhalten vom Benutzer definiert werden kann. Jedem Agenten kann eine visuelle

[6] Das in der Abbildung dargestellte Petri-Netz wird blockieren, da die grüne Stelle nach dem Schalten der ersten Transition nur ein Token besitzt. Es müssten hingegen drei Tokens sein, um die Blockierung zu verhindern. Ich möchte dieses Beispiel trotzdem nutzen, da es sehr übersichtlich die Idee für 3D-Petri-Netze darstellt und die einzige farbige Skizze von Rölke dazu ist.

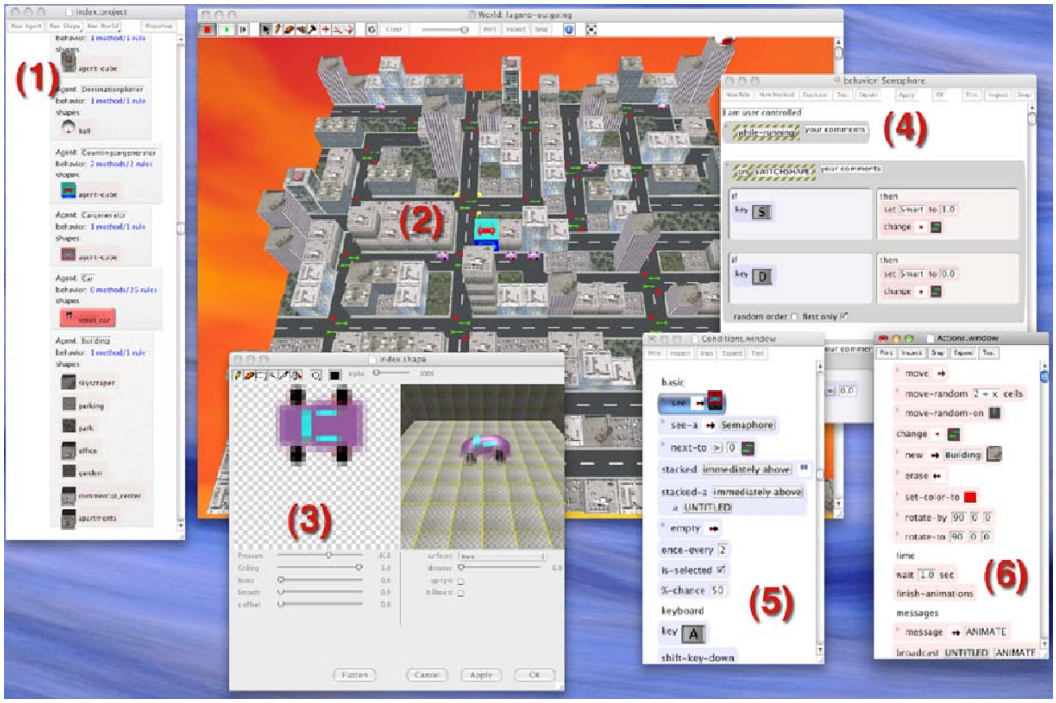


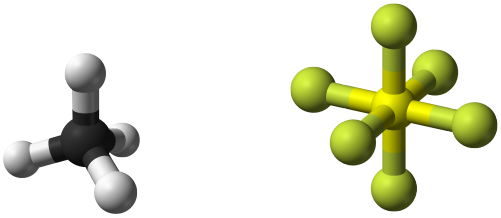

**Abbildung 2.8:** Bildschirmfoto der AgentCubes-Anwendung; Quelle: [IRW09].

Gestalt zugewiesen werden. Dies können entweder Objekte aus der realen Welt (wie Autos, Menschen, Tiere) oder abstrakte Objekte (z. B. Zahlen) sein. Die Agenten werden in dem *Agentcube* – einem dreidimensionalen Würfel – angeordnet, der schichtbasiert ist. Jede Schicht enthält Zeilen und Spalten, sodass eine Zelle, die einen Stapel von Agenten beinhalten kann, durch diese drei Parameter bestimmt wird.

In Abbildung 2.8 ist ein Bildschirmfoto der AgentCubes-Anwendung zu sehen, in der ein Verkehrssimulationsspiel implementiert wird. Neben der *Welt*-Sicht (2), die die Spielfläche von AgentCubes inklusive allen definierten Agenten darstellt, gibt es weitere Sichten, in denen das Aussehen, das Verhalten und die Interaktion mit den Agenten definiert werden kann. Die regelbasierte, visuelle Sprache des *Verhaltenseditors* in Sicht (4) war schon in ähnlicher Form im AgentSheets-System vorhanden. Der *Editor für aufpumpbare Zeichnungen* in Sicht (3) ist hingegen 3D-spezifisch und ermöglicht es zunächst zweidimensionale Zeichnungen anzufertigen, die anschließend in die dritte Dimension aufgepumpt werden.

### 2.2.9 Molekülmodelle

Eine dreidimensionale Darstellungsweise von Molekülmodellen ist das *Kugel-Stäbchen-Modell*, das die Atome als Kugeln und die Atombindungen als Stäbchen



**(a)** Methan – $CH_4$. **(b)** Schwefelhexafluorid – $SF_6$.

**Abbildung 2.9:** Kugel-Stäbchen-Modelle von Molekülen.

darstellt und dadurch die räumliche die Gestalt eines Moleküls veranschaulicht. Ein formales Modell, mit welchem die geometrische Gestalt eines Moleküls bestimmt werden kann, ist das *VSEPR-Modell*[7] der Chemiker Gillespie und Nyholm [GR05]. Die räumliche Gestalt wird aus den abstoßenden Kräften zwischen den bindenden Elektronenpaaren berechnet, die sich möglichst weit voneinander entfernen.

Die geometrische Gestalt eines Moleküls hängt von den abstoßenden Kräften zwischen den Elektronenpaaren ab. Diese befinden sich nach der Abstoßung an den Ecken eines Polyeders. Bei drei bindenden Elektronenpaaren befinden sich diese an den Ecken eines gleichseitigen Dreiecks. Bei vier Elektronenpaaren ergibt sich als geometrische Form ein Tetraeder, wie in Abbildung 2.9a zu sehen ist. Der Winkel zwischen den vier Verbindungen beträgt 109,5° und wird auch als *Tetraederwinkel* bezeichnet. Ein Oktaeder entsteht bei sechs Elektronenpaaren, wie z. B. beim Schwefelhexafluorid-Molekül (siehe Abbildung 2.9b). Der Winkel zwischen den Bindungen beträgt hier 90°. Diese Beschreibung vermittelt nur einen sehr groben Eindruck davon, wie die räumliche Gestalt eines Moleküls bestimmt wird. In Abschnitt 5.5.1 werde ich eine Strategie zur Berechnung des Molekül-Layouts genauer vorstellen.

### 2.2.10 ToneCraft

*ToneCraft*[8] ist eine 3D-Web-Anwendung der schwedischen Firma DinahMoe mit der sich elektronische Musik komponieren lässt. Die grundlegende Idee dabei ist, dass verschiedenfarbige Würfel, die unterschiedliche Musikinstrumente repräsentieren, in eine matrixartige 3D-Welt eingefügt werden. Darin hat jede der drei Dimensionen eine feste Semantik: Entlang der $x$-Achse wird die Zeit abgebildet, die $y$-Achse repräsentiert die Tonhöhe und die $z$-Achse wird genutzt, um ver-

[7] Abk. für *valence shell electron pair repulsion*; oder im Deutschen *EPA-Modell* für Elektronenpaarabstoßungsmodell

[8] `http://labs.dinahmoe.com/ToneCraft`

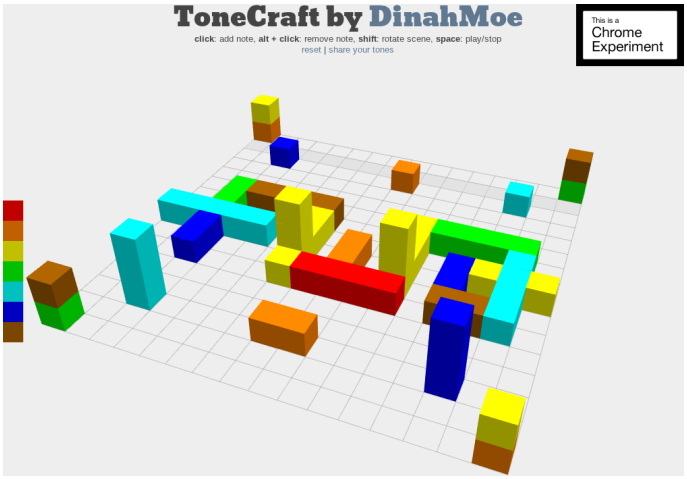



**Abbildung 2.10:** Tonecraft-Beispiel.

schiedene Instrumente zur selben Zeit zu überlagern. Während der Konstruktion wird die durch die Würfel komponierte Musik abgespielt. Dem Anwender wird die aktuelle Position auf der Zeit-Achse durch eine dunkelgrau unterlegte Zeile angedeutet. Dies ist auch in dem Tonecraft-Beispiel in Abbildung 2.10 zu sehen, in dem die Zeitachse von hinten nach vorne verläuft.

### 2.2.11 Weitere Ansätze

Der Vollständigkeit halber seien an dieser Stelle kurz weitere Ansätze für dreidimensionale Sprachen erwähnt. Sie werden hier komprimiert zusammengefasst, da sie entweder Parallelen zu bereits vorgestellten Sprachen besitzen – sodass eine ausführliche Vorstellung zu Redundanzen führt – oder der Ansatz in Summe zu unergiebig für eine detaillierte Vorstellung ist.

Zu 3D-Projekten, die zu AgentCube sehr ähnlich und im Bereich der Programmiererziehung für Kinder angesiedelt sind, zählen *Alice* [KP07], *ToonTalk* [MK08] und *VRMath* [YN04]. Ziel von Alice ist es, eine virtuelle 3D-Welt mit Charakteren zu bevölkern, Regeln für diese zu definieren und gemäß dieser, Animationen – ähnlich wie in Zeichentrickfilmen – der 3D-Modelle zu erzielen. ToonTalk ist eine visuelle Programmiersprache, die alle Programmkonstrukte als dreidimensionale LEGO-artige Figuren darstellt und Interaktionsmöglichkeiten mit diesen anbietet. VRMath ist eine Übertragung der bekannten *Logo-Programmiersprache* ins Dreidimensionale, bei der die Logo-Schildkröte durch die 3D-Welt bewegt werden kann, um 3D-Objekte zu konstruieren.

*PrologSpace* [YF96] wird als allgemeine visuelle Programmiersprache beschrieben, die auf der logischen textuellen Sprache *Prolog* basiert und eine 3D-Darstellung

für Prolog-Programme anbietet. Damit ist dieser Ansatz ähnlich zu Lingua Graphica. Einen allgemeinen visuellen Ansatz zur besseren Verständlichkeit von Programmen und ihrer Ausführung stellen Freeman et al. [FGJ95; FGJ96] mit ihrem *visuellen Vokabular* innerhalb ihrer visuellen Programmierumgebung *MAP* vor. Die Struktur von MAP weist jeder räumlichen Dimension eine feste Semantik zu und nutzt Schachtelung, um Gültigkeitsbereiche textueller Programmiersprachen nachzubilden. Del Bimbo und Vicario stellen eine 3D-Darstellung für *Computation Tree Logic*-Formeln (kurz: CTL) vor [DV99]. Grundlage ist eine planare Darstellung von Grundelementen der Formeln, wobei jedes Auftreten einer temporalen Komposition zu einer neuen planaren Ebene führt, die mit anderen Ebenen entsprechend ihrer Komposition verbunden ist. In der Architektur-Domäne finden sich weitere Anwendungsfälle für 3D-Sprachen, da deren Elemente ohne Informationsverlust nur dreidimensional dargestellt werden können. In Abschnitt 2.4.4 wird ein Editor aus dieser Kategorie betrachtet.

## 2.3 Klassifikation von 3D-Sprachen

In diesem Abschnitt werde ich die zuvor beschriebenen 3D-Sprachen analysieren und auf gemeinsame Aspekte der Sprachen näher eingehen. Aus diesen Aspekten werden wichtige Prinzipien für dreidimensionale Sprachen herausgearbeitet, die dann als Kriterien zur Klassifikation der 3D-Sprachen beitragen.

In vielen der vorgestellten 3D-Sprachen werden Sprachkonstrukte in andere geschachtelt. Das *Prinzip der Schachtelung* wird in Cube und 3D-PP angewendet, die beide auf textuellen logischen Sprachen basieren. Dort wird durch Schachteln von Sprachkonstrukten die Struktur von Klauseln und Prädikaten nachgebildet. So werden in 3D-PP die Elemente der rechten Seite einer Klausel in ein Sprachkonstrukt geschachtelt, welches die linke Seite der Klausel darstellt. In Cube werden Definitionen von Prädikaten in transparenten Würfeln geschachtelt. In den anderen drei Sprachen, in denen Schachtelung auftritt, wird damit eine Enthaltensein-Relation ausgedrückt. So werden in der Sprache SAM Regeln in die kugelförmige Darstellung eines Agenten geschachtelt. Bei dreidimensionalen Petri-Netzen werden Tokens in Stellen geschachtelt und in 3D-Klassendiagrammen lassen sich Klassen-Strukturen zu Paketdarstellungen einschachteln. Sprachen aus der Architektur-Domäne unterstützen eine ganz natürliche Form der Schachtelung, indem z. B. Einrichtungsgegenstände in Räume geschachtelt werden.

Bei dreidimensionaler Schachtelung ist besonders ihr Konstruktions- und Interaktionsaspekt zu beachten. Im Zweidimensionalen werden geschachtelte Darstellungen von einer umgebenden (oft rechteckigen) Darstellung umhüllt. Ein Zugriff eines Editornutzers auf diese Elemente ist unmittelbar gegeben. Anders verhält es sich bei der dreidimensionalen Schachtelung. Dort werden geschachtelte Objekte

von anderen echt inkludiert. Dabei ist der direkte Zugriff auf die geschachtelten Objekte durch die Hülle des aufnehmenden Objekts behindert. Auf Techniken, die solche Probleme handhabbar machen, werde ich in Kapitel 6 näher eingehen. Außerdem ist zu beachten, dass bei der dreidimensionalen Schachtelung die Sichtbarkeit der geschachtelten Objekte die Transparenz des umgebenden Objekts bedingt. Auf diese Aspekte haben bereits die Autoren von 3D-PP [OT99] hingewiesen und in diesem Zusammenhang eine Methode vorgestellt mit der die Sichtbarkeit von tief geschachtelten Objekten begrenzt werden kann, da andernfalls die Darstellung zu unübersichtlich wird.

In einigen 3D-Sprachen hat die Anordnung von Sprachkonstrukten eine bestimmte und für die Bedeutung der Sprache zentrale *Semantik*. Hier unterscheide ich zwei Fälle, nach denen Semantik eine Rolle spielt. Zum einen hat die relative Anordnung von Sprachkonstrukten zueinander eine bestimmte Semantik. So verhält es sich bei Cube, wo vertikal (horizontal) zueinander angeordnete Würfel Disjunktion (Konjunktion) bedeuten. Bei 3D-Visulan gibt die relative Position von Regeln zueinander die Priorität der Regeln an. Charakteristisch für diesen Fall ist, dass bei nur einem Sprachkonstrukt keine direkte semantische Aussage gemacht werden kann. So kann z. B. nur ein Würfel in Cube keine Disjunktion oder Konjunktion eingehen. Anders verhält es sich bei ToneCraft, 3D-Petri-Netzen und MAP. Hier ist nicht die relative Anordnung zueinander ausschlaggebend, sondern die Position eines Sprachkonstrukts auf einer Koordinatenachse. In diesem Fall wird einer Dimension fest eine Semantik zugewiesen. Bei ToneCraft werden Dimensionen Eigenschaften wie Zeit und Tonhöhe zugeordnet. Ein Instrumentenwürfel ist gleichzeitig der Zeit- und der Tonhöhenachse zugehörig und ihm wird gemäß seiner diskreten Position auf der Dimensionsachse ein Zeitpunkt bzw. eine Tonhöhe zugeordnet. Bei Petri-Netzen wird einer Ebene, die auf der $xz$-Ebene angeordnet ist, eine Semantik zugewiesen, da sie einen ganz bestimmten Teilaspekt darstellt.

In Sprachen, deren Sprachkonstrukte durch Linienverbindungen miteinander verbunden sind, kann durch deren Darstellung im Dreidimensionalen *Überschneidungsfreiheit* sichergestellt werden. So verhält es sich z. B. bei 3D-Petri-Netzen und auch dreidimensionalen Darstellungen der UML. Nicht verwunderlich ist, dass es sich dabei um ursprünglich zweidimensionale Sprachen handelt. Gerade in der klassischen zweidimensionalen Darstellungsweise können Linienüberschneidungen leicht auftreten. Die dritte Dimension erlaubt weiterhin die Darstellung von Verbindungen zwischen Konstrukten verschiedener Diagramme in einer Sicht. So können z. B. Verbindungen zwischen Objekten in Sequenzdiagrammen und ihren definierenden Klassen dargestellt werden [RG00]. Nach dem gleichen Prinzip werden die Computation Tree Logic-Formeln in 3D miteinander komponiert.

Ein weiterer charakteristischer Aspekt ist die Farbe von Sprachkonstrukten wie sie bei ToneCraft und Cube eine Rolle spielt. Dies ist aber hinsichtlich der dritten Dimension kein Alleinstellungsmerkmal, da verschiedenfarbige Sprachkonstrukte, die dadurch eine bestimmte Bedeutung haben, auch in 2D-Sprachen üblich ist. Etwas anders verhält es sich mit dem Andocken von Sprachkonstrukten, wie es in SAM angewendet wird, um einen kugelförmigen Agenten mit Ports zu verbinden. Das Andocken von Sprachkonstrukten ist schon aus dem Zweidimensionalen bekannt, wird allerdings durch die dritte Dimension komplexer. Interessant ist weiterhin zu beobachten, dass viele der vorgestellten 3D-Sprachen entweder auf textuellen oder visuellen 2D-Sprachen basieren. Für welche das gilt, ist Tabelle 2.1 zu entnehmen, die die 3D-Sprachen auflistet und diese anhand verschiedener Kriterien klassifiziert.

Erste Ansätze für eine Klassifikation von 3D-Sprachen habe ich bereits in meiner Masterarbeit [Wol11] erarbeitet. Erstes Kriterium ist danach die Unterscheidung

| | **Klassifikationskriterium** | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | **inhärente 3D-Darst.** | | | **Semantik** | | **basiert auf** | |
| | **SK abstr. Form** | **SK aus realer Welt** | **erweit. 2D-Darst.** | **adapt. 2D-Darst.** | **Sprachkonstrukte** | **Beziehung zw. SK** | **Schachtelung** | **Dimension** | **rel. Anordnung** | **textueller Sprache** | **vis. 2D-Sprache** |
| Cube | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | ✓ | ✓ | ✗ |
| SAM | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |
| 3D-PP | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ |
| 3D-Visulan | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |
| Lingua Graphica | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| UML in 3D | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |
| 3D-Petri-Netze | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ |
| AgentCubes | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |
| Molekülmodelle | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| ToneCraft | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ |
| Architektur | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ |
| MAP | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ |
| CTL | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |

**Tabelle 2.1:** Klassifikation der 3D-Sprachen.

der Sprachkonstrukte anhand ihrer äußeren Darstellung: entweder stellen sie Objekte aus der realen Welt dar oder sind abstrakter Form. Um allerdings eine Aussage zur zweckmäßigen Anwendung der dritten Dimension zu machen, ist dieses Kriterium noch nicht ausreichend. Zu diesem Zweck habe ich eine Klassifikation für 3D-Visualisierungen von Stasko und Wehrli [SW93] für dreidimensionale Sprachen adaptiert und erweitert. Diese Klassifikation kennt drei verschiedene Kategorien von 3D-Darstellungen:

**Erweiterte 2D-Darstellungen** benötigen typischerweise nur zwei räumliche Dimensionen, wobei die dritte Dimension nur aus ästhetischen Gründen hinzugefügt wird.

Als Beispiel hierfür kann Lingua Graphica (siehe Seite 17) dienen, da die visuelle Darstellung von textuellen Programmen nicht in sinnvoller Weise Gebrauch von der dritten Dimension macht. Ähnlich verhält es sich bei auf 3D erweiterte Zustandsdiagrammen (siehe Abbildung 2.22 auf Seite 46).

**Adaptierte 2D-Darstellungen** benötigen minimal zwei räumliche Dimensionen. Der Unterschied zu erweiterten 2D-Darstellungen ist allerdings, dass die dritte Dimension nicht nur der Ästhetik dient, sondern eine weitere Funktionalität darstellt, die in 2D so nicht darstellbar gewesen wäre.

Dreidimensionale Petri-Netze, wie ich sie auf Seite 19 vorgestellt habe, sind ein guter Repräsentant für adaptierte 2D-Darstellungen. Es werden auf verschiedenen Ebenen Teilaspekte inklusive deren Beziehungen zueinander dargestellt, die in solcher Form im Zweidimensionalen nicht möglich gewesen wären.

**Inhärente 3D-Darstellungen** Diese Kategorie umfasst Darstellungen, die inhärent dreidimensionale Entitäten beinhalten. Hier lassen sich – wie ich schon in [Wol11, S. 64f.] argumentiert habe – zwei Fälle unterscheiden, woher die Inhärenz der 3D-Darstellung stammt:

1. Die Sprachkonstrukte sind inhärent dreidimensional und lassen sich nur so adäquat darstellen.

   Beispiele für diese Kategorie sind 3D-Sprachen aus der Architektur-Domäne, da sie inhärent dreidimensionale Sprachkonstrukte verwenden. AgentCubes (siehe Seite 19) zählt ebenfalls in diese Kategorie, da die damit erstellten Spielwelten diesem Kriterium genügen.

2. Die Beziehungen zwischen Sprachkonstrukten benötigen zwingend die dritte Dimension.

   Molekülmodelle (siehe Seite 20) erfüllen dieses Kriterium, da ihre dreidimensionale Gestalt von der Anzahl der bindenden Elektronenpaare

abhängt. So ordnen sich Atome z. B. tetraedrisch um ein Zentralatom an.

Einer 3D-Sprache kann demnach nur ein gewinnbringender Einsatz der dritten Dimension attestiert werden, wenn eine adaptierte 2D-Darstellung oder eine inhärente 3D-Darstellung gegeben ist. In Tabelle 2.1 sind diese Kriterien samt ihrer Anwendung auf alle in Abschnitt 2.2 vorgestellten Sprachen dargestellt. Als weitere Klassifikationskriterien habe ich die eingangs angesprochenen Aspekte aufgegriffen. So ist eine Klassifikation hinsichtlich Schachtelung und der Zuordnung von Semantik möglich. Weiterhin ist dargestellt, ob die 3D-Sprache eine Weiterentwicklung einer textuellen oder visuellen 2D-Sprache ist.

## 2.4 3D-Editoren

In den bisherigen Abschnitten wurden 3D-Sprachen beschrieben, aufgezeigt welchen Nutzen sie aus der dritten Dimension ziehen und abschließend klassifiziert. Die Konstruktion von Instanzen einer 3D-Sprache stand in den meisten frühen Publikationen zu 3D-Sprachen nicht im Fokus. Najork gesteht [Naj96, S. 238], dass der Editor zur Konstruktion von Cube-Programmen so primitiv ist, dass seine Benutzung keinem anderen als ihm selbst zuzumuten ist. Da Interaktions- und Navigationsprinzipien, die die Konstruktion von 3D-Diagrammen ermöglichen, in dieser Arbeit von entscheidender Bedeutung sind, werde ich derartige Aspekte anhand von 3D-Editoren anderer Anwendungsdomänen beleuchten. Bei den ausgewählten Editoren handelt es sich ausschließlich um Desktop-Anwendungen, die mit klassischen Eingabegeräten wie Maus und Tastatur bedient werden können. Dies ist für die Zwecke dieser Arbeit ausreichend, da das zu entwickelnde Generatorsystem ebenfalls Editoren generieren soll, die als Desktop-Anwendung auf herkömmlichen Computern laufen und mit klassischen Eingabegeräten bedient werden. Aus diesem Grund stehen z. B. 3D-Anwendungen für Tabletcomputer, die durch Berühren des kapazitiven Displays gesteuert werden, nicht im Fokus.

### 2.4.1 3ds Max

Mit *3ds Max*[9] lassen sich 3D-Grafiken und deren Animation für Computerspiele oder Filme erstellen. Ein Bildschirmfoto des Editors ist in Abbildung 2.11 zu sehen. In diesem lassen sich 3D-Szenen, die anschließend auch animiert werden können, durch Komposition von vordefinierten Grundprimitiven konstruieren, deren Form durch Anpassung von Parametern verändert werden kann.

[9] `http://www.autodesk.de/products/autodesk-3ds-max/overview`

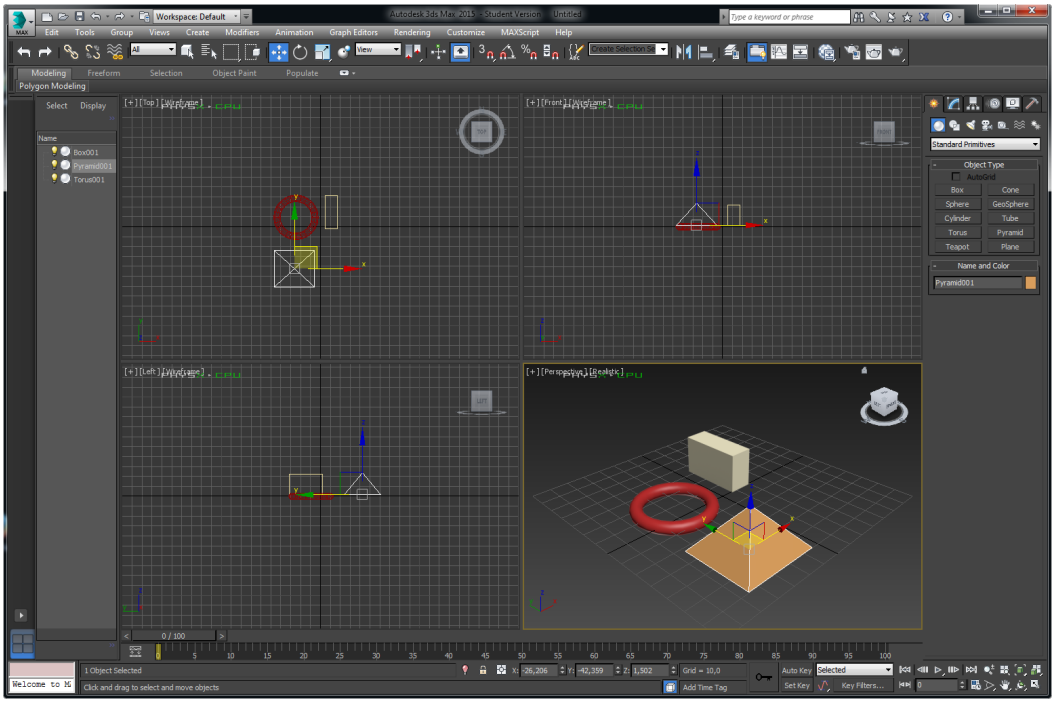


**Abbildung 2.11:** Bildschirmfoto des 3ds Max Editors.

Das Interaktionskonzept von 3ds Max ist objektorientiert, was bedeutet, dass jedes Objekt Informationen über die auf sich anwendbaren Operationen besitzt. Eine zentrale Interaktionsoperation ist die Selektion, die *direkt* oder *indirekt* erfolgen kann. Objekte können direkt mithilfe der Maus selektiert werden. Dabei wird ein *Raycasting*-Ansatz angewendet, bei dem ein unendlich langer Strahl beginnend am Mauscursor in die 3D-Szene geschossen wird. Das erste Objekt, welches von dem Strahl geschnitten wird, gilt als selektiert und wird als solches durch einen transparenten Hüllkörper markiert, der die gleiche Form wie das Objekt besitzt. Unter indirekter Selektion werden Operationen verstanden, die mehrere Objekte, die eine gemeinsame Eigenschaft besitzen, gleichzeitig auswählen. So lassen sich alle Objekte einer bestimmten Farbe oder eines bestimmten Layers (das sind transparente Overlays, mit der sich die 3D-Szene strukturieren lässt) auswählen. Weiterhin wird eine Mehrfachselektion angeboten, bei der eine geometrische zweidimensionale Form, wie ein Rechteck oder Kreis, gezeichnet werden kann, die dann orthogonal zur Kameraposition in die 3D-Szene expandiert und alle darin enthaltenen 3D-Objekte auswählt.

Auf selektierte Objekte können anschließend Manipulationsoperationen angewendet werden, wobei auch hier zwischen direkter und indirekter Manipulation unterschieden wird. Direkte Manipulation erfolgt durch sogenannte *Widgets*, die

Objekten angehängt werden können, um diese zu verschieben, zu rotieren oder zu skalieren. In Abbildung 2.11 ist an die Pyramide ein Translation-Widget angehängt, um es entlang einer der drei Koordinatenachsen oder simultan entlang zweier Achsen zu verschieben. Die indirekte Manipulation erfolgt durch Änderung von Objekt-Parametern in einer Textbox.

Bei der Navigation kann die Kamera entweder frei den 3D-Raum erkunden oder zielorientiert ein bestimmtes Objekt anvisieren. Bei Letzterem ist die Kamera in festem Abstand zum anvisierten Objekt platziert und kann sich nur um dieses Objekt herum bewegen. Bei der freien Erkundung ist die Kamera um alle sechs Freiheitsgrade bewegbar und seine Position unabhängig von den Objekten der Szene wählbar. Weiterhin kann der Benutzer mit einem *Walkthrough-Assistenten* eine Route durch die Szene festlegen, entlang der sich anschließend die Kamera bewegt. Die Szene kann außerdem gleichzeitig aus verschiedenen Perspektiven betrachtet werden. So ist in der Hauptsicht in dem unteren rechten Quadranten die 3D-Szene aus einer individuellen Perspektive zu sehen und die anderen drei Quadranten zeigen die Szene orthogonal zu den drei Koordinatenachsen. In der oberen rechten Ecke der 3D-Sicht befindet sich der sogenannte *ViewCube*, der eine Rückmeldung über die aktuelle Position in der 3D-Sicht gibt, mit dem diese verändert werden kann. So ist jede Seite des Würfels mit einer Richtungsangabe wie „Front“, „Left“ oder „Top“ beschriftet. Ein Klick auf diese Flächen sorgt dafür, dass die Szene aus der gewählten Perspektive betrachtet werden kann.

### 2.4.2 FreeCAD

FreeCAD[10] ist ein plattformübergreifender 3D-CAD-Editor[11] mit dem sich zusammenhängende 3D-Modelle konstruieren lassen. Diese Modelle sind im Bereich technischer Zeichnungen von Bauteilen oder in der Architektur angesiedelt. FreeCAD verfolgt einen parametrischen Ansatz, was bedeutet, dass jedes Objekt über eine Reihe von Parametern definiert ist, die es ermöglichen durch einfache Änderung dieser Parameter einen vorherigen Zustand der Versionshistorie wieder herzustellen.

Die Konstruktion der 3D-CAD-Modelle erfolgt durch Komposition von vordefinierten dreidimensionalen Grundprimitiven. Über die Werkzeugleiste lassen sich Quader, Zylinder, Kugeln, Pyramiden und Tori einfügen. So eingefügte Primitive erscheinen in der Mitte der 3D-Sicht und können anschließend mithilfe eines Widgets verschoben und rotiert werden (in Abbildung 2.12 ist dem Torus ein Widget angehängt). Weitere Eigenschaften, wie das Zuweisen einer Farbe oder die Festlegung der exakten Größe, werden in einem Dialog an der linken Seite des Fensters

[10] `http://www.freecadweb.org/`
[11] CAD ist ein Akronym für *Computer-Aided Design*

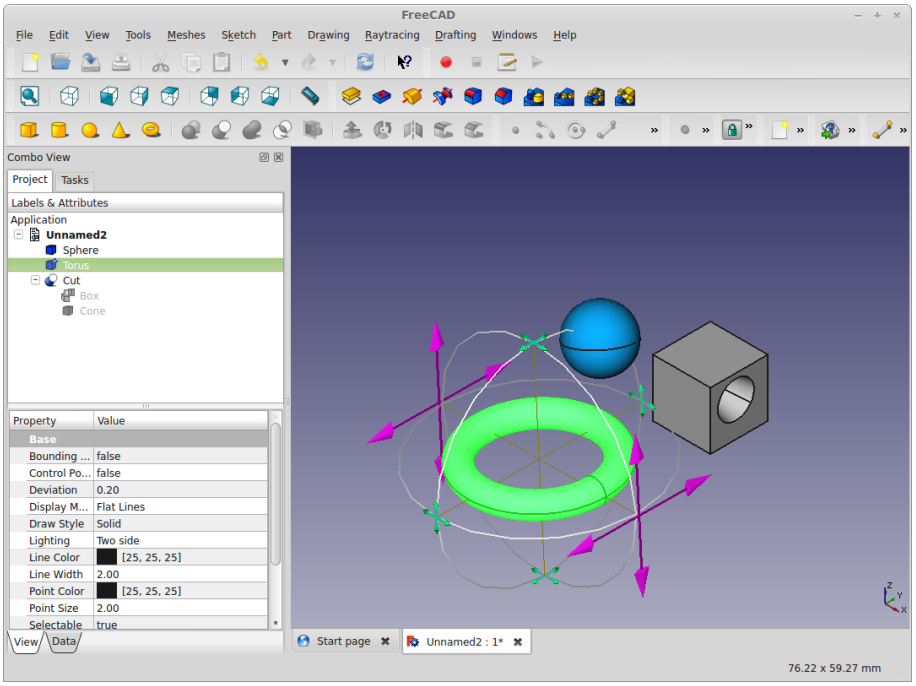



**Abbildung 2.12:** Bildschirmfoto des FreeCAD Editors.

definiert. Um aus den Grundprimitiven komplexere Modelle erschaffen zu können, lassen sich Primitive durch Angabe von boolschen Funktionen miteinander kombinieren. So lassen sich aus überlappenden Primitiven neue Konstrukte formen, die deren Vereinigung, Differenz oder Schnittmenge beschreiben. So wurde z. B. aus dem Quader in Abbildung 2.12 die Spitze eines Kegels „herausgeschnitten“.

Die Möglichkeiten in der 3D-Szene zu navigieren sind recht beschränkt. So ist es z. B. nicht möglich die Kamera frei zu bewegen und die Szene so aus beliebig vielen Blickwinkeln zu erkunden. Stattdessen kann aus sieben festgelegten Kameraperspektiven gewählt werden. Die Standardansicht blickt von rechts oben auf die Szene, die anderen Perspektiven zeigen die Szene von den sechs Seiten.

### 2.4.3 Avogadro

Avogadro[12] ist ein freier plattformübergreifender Editor zur Konstruktion von Molekülmodellen. Der Editor (siehe Abbildung 2.13) besteht aus einer dynamisch konfigurierbaren Werkzeugleiste an der linken Seite und der 3D-Sicht, in der Moleküle direkt konstruiert werden können. Das Darstellungskonzept der Moleküle kann beliebig geändert werden. So wird z. B. neben dem Kugel-Stäbchen-Modell auch das Kalotten- und Drahtmodell unterstützt.

[12] `http://avogadro.cc/wiki/Main_Page`

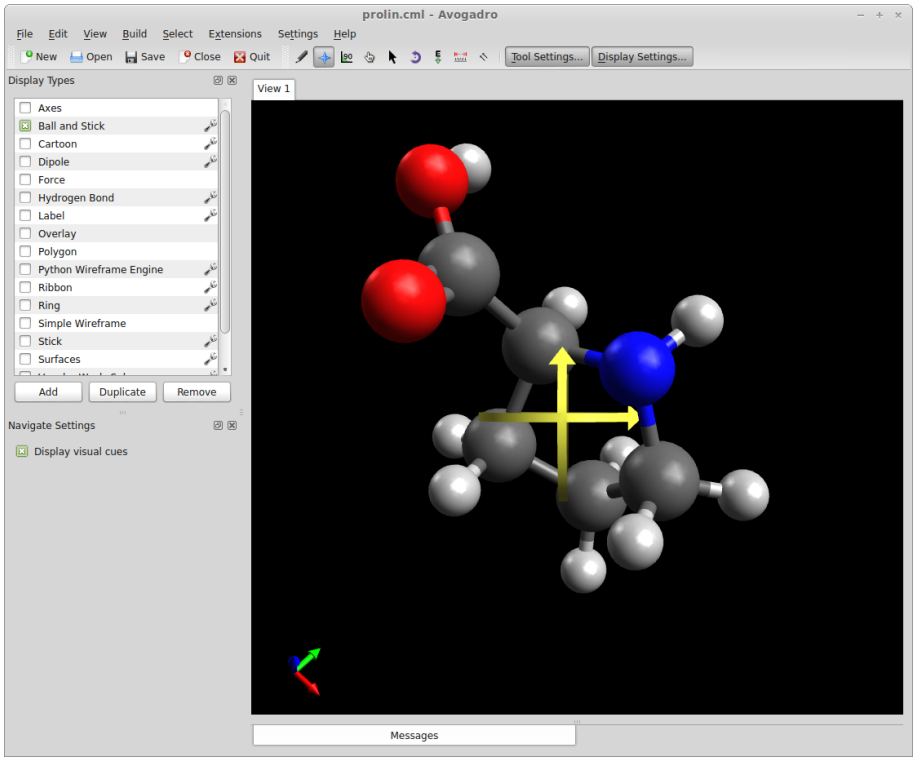



**Abbildung 2.13:** Bildschirmfoto des Avogadro Moleküleditors.

Zur Konstruktion eines Molekülmodells lassen sich mit dem Zeichenwerkzeug Atome, die aus dem Periodensystem gewählt werden, direkt in die 3D-Szene einfügen. Das Einfügen einer Bindung zwischen zwei Atomen erfolgt durch Drag-and-Drop zwischen den zu verbindenden Atomen. Mit dem Selektionswerkzeug können beliebig viele Atome und Bindungen ausgewählt werden, welche dann durch transparente Hüllkörper gekennzeichnet werden. Die ausgewählte Menge an Objekten kann anschließend mit dem Manipulationswerkzeug verschoben und rotiert werden.

Mit dem Navigationswerkzeug kann der Blick auf die 3D-Szene variiert werden. Dafür kann die Szene mit gedrückt gehaltener linker Maustaste rotiert, mit der rechten Maustaste verschoben und mit der mittleren gezoomt werden. Wird in den freien Raum geklickt erfolgt eine Rotation bzw. Verschiebung oder Zoom um den Mittelpunkt des bisher konstruierten Modells; andernfalls um ein ausgewähltes Atom.

### 2.4.4 LEGO Digital Designer

Der von dem Spielwarenhersteller LEGO entwickelte *LEGO Digital Designer*[13] ist ein Editor mit dem sich alle Modelle gemäß des bekannten Bausteinkonzepts erstellen lassen, die sich auch mit echten Steinen in der realen Welt konstruieren lassen. Zentraler Teil des Editors ist der *Aufbaumodus* (siehe Abbildung 2.14) in dem in einer 3D-Sicht das LEGO-Modell erstellt wird. Links neben der 3D-Sicht befindet sich eine Liste mit zahlreichen Bausteinen, die dort ausgewählt und dann in der 3D-Sicht platziert werden können.

Beim Einfügen von Bausteinen in die 3D-Sicht bildet der Einfügemechanismus Gravitationskräfte nach und erlaubt somit die Positionierung ausschließlich auf der anfangs leeren Grundfläche oder auf bereits dort platzierten Bausteinen. Einzige Ausnahme dieses Prinzips ist, dass auch Bausteine direkt unterhalb von schon eingefügten Bausteinen – also zwischen diesen und der Grundfläche – platziert werden können, was allerdings im Anschluss dazu führt, dass die Grundfläche nach unten rückt. Der einzufügende Stein ändert seine Position in Abhängigkeit der Mausbewegungen gemäß der oben beschriebenen Prinzipien und nimmt eine feste Position erst nach einem bestätigenden Klick mit der rechten Maustaste ein.

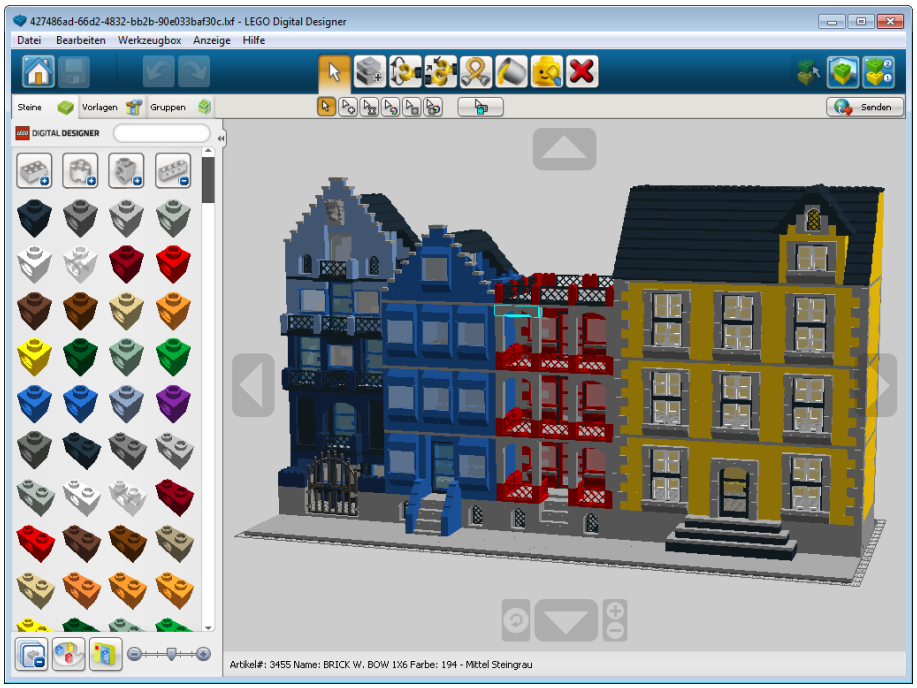



**Abbildung 2.14:** Der Aufbaumodus des LEGO Digital Designers.

[13] http://ldd.lego.com/

Bereits positionierte Steine können natürlich jederzeit ausgewählt und umpositioniert werden. Die Auswahl eines Steins erfolgt durch Mausklick auf diesen und wird durch eine *Bounding-Box* gekennzeichnet, die den minimal umgebenden Quader des Bausteins beschreibt. Die manuelle Auswahl mehrerer Bausteine zur gleichen Zeit sowie die automatische Auswahl aller Steine einer bestimmten Form oder Farbe ist mit speziellen Werkzeugen möglich. Weiterhin können Bausteine mithilfe des Rotier- und Umklappwerkzeugs bearbeitet werden.

Zur Navigation in der 3D-Sicht umfasst diese vier Buttons, mit der die Sicht nach links, rechts, oben und unten gedreht werden kann. Zusätzlich gibt es Buttons zum Hineinzoomen in die Sicht und zum Zurückkehren in eine Position, sodass alle eingefügten Bausteine des Modells sichtbar sind.

### 2.4.5 Analyse der vorgestellten Editoren

Die Auswahl der vorgestellten vier 3D-Editoren gibt einen anschaulichen Überblick über weitverbreitete Anwendungen für solche Editoren. Von der Erstellung von 3D-Modellen für Computerspiele, über eine CAD-Anwendung zur Konstruktion von Bauteilen hin zu einem Editor für Molekülmodelle und eine an Kinder gerichtete Anwendung zur Erstellung von LEGO-Modellen. Alle haben gemein, dass sie Editierfunktionen direkt im 3D-Raum zur Verfügung stellen und nicht etwa ein 3D-Modell aus 2D-Darstellungen interpolieren. Damit einher gehen Techniken zur Navigation, die den 3D-Raum für den Benutzer besser beherrschbar machen.

Die Interaktionsoperationen der Editoren weisen viele Gemeinsamkeiten auf. So können zunächst vordefinierte Objekte frei in der 3D-Sicht platziert werden. Einzig der LEGO Digital Designer erlaubt bei der Positionierung nicht alle Freiheiten und schränkt sie dahingehend ein, dass Bausteine nur auf der Grundfläche oder auf schon vorhandenen Objekten platziert werden können. Anschließend können in den meisten Editoren die Objekte mithilfe von Widgets verschoben, rotiert oder skaliert werden. Einige Editoren warten mit recht spezifischen Funktionen auf, wie z. B. FreeCAD mit boolschen Funktionen zur Schaffung neuer Konstrukte oder Avogadro, welcher das erstellte Molekül gemäß den chemischen Regeln layoutet.

Bei den angebotenen Navigationsmöglichkeiten unterscheiden sich die Editoren deutlicher. In 3ds Max und Avogadro kann der Anwender die Kamera frei bewegen und so die 3D-Szene aus beliebigen Blickwinkeln erkunden. Bei einem solchen Vorgehen spricht man nach Bowman et al. [BKLP04] auch von der *Kamera in der Hand* Metapher. Die Kamera des LEGO-Editors hingegen kann nur mit vier Buttons geschwenkt werden, wobei die Kamera allerdings stets den Nullpunkt der 3D-Szene anvisiert. Noch eingeschränkter ist die Navigation in FreeCAD, die nur feste Kameraperspektiven anbietet, ein Justieren der Kameraposition ist nicht möglich. Einige der von FreeCAD bereitgestellten seitlichen Ansichten stellt auch

3ds Max bereit; dies allerdings zusätzlich zu der vom Anwender steuerbaren 3D-Sicht, was insgesamt einen besseren Überblick über die 3D-Szene ermöglicht.

## 2.5 3D-Visualisierung

Card et al. [CMS99] beschreiben *Visualisierung* als den „Gebrauch von computerunterstützten, interaktiven, visuellen Repräsentationen von Daten zur Vergrößerung der Kognition“, wobei Kognition die Informationsverarbeitung des Menschen zur Erlangung von mehr Wissen meint. Das Forschungsfeld der Visualisierung teilt sich hauptsächlich in *wissenschaftliche Visualisierung* und *Informationsvisualisierung*. Erste beschäftigt sich meist damit, Objekte oder Konzepte der realen Welt, die eine räumliche Komponente besitzen, darzustellen, wohingegen die Informationsvisualisierung sich mit abstrakten Konzepten oder Verbindungen, die keine räumlichen Eigenschaften besitzen, auseinandersetzt. Die *Softwarevisualisierung* ist ein Teilgebiet der Informationsvisualisierung, welches das Ziel verfolgt Software durch Visualisierung verständlicher zu machen. Die Differenzierung dieser Teilgebiete werde ich nicht näher vertiefen, sondern Charakteristika der 3D-Visualisierung herausstellen, die in allen diesen Teilgebieten Anwendung finden.

Doch zunächst werde ich auf den von Bertin [Ber83] geprägten Begriff der *visuellen Variablen* eingehen, mit denen sich visuelle Notationen charakterisieren lassen. Abbildung 2.15 veranschaulicht die visuellen Variablen, die in *ebene Variablen* und *retinale Variablen*, also vom menschlichen Auge wahrnehmbare Eigenschaften, aufgeteilt sind. Die visuellen Variablen wurden zur Beschreibung von zweidimensionalen Darstellungen konzipiert, lassen sich aber sehr einfach auf den dreidimensionalen Raum übertragen. Die retinalen Variablen Form, Größe, Farbe, Helligkeit, Ausrichtung und Textur können dabei unverändert bleiben. Lediglich die Kategorie der ebenen Variablen, die im Kontext dreidimensionaler Repräsentationen besser als *räumliche Variablen* bezeichnet werden sollten, muss um eine Variable

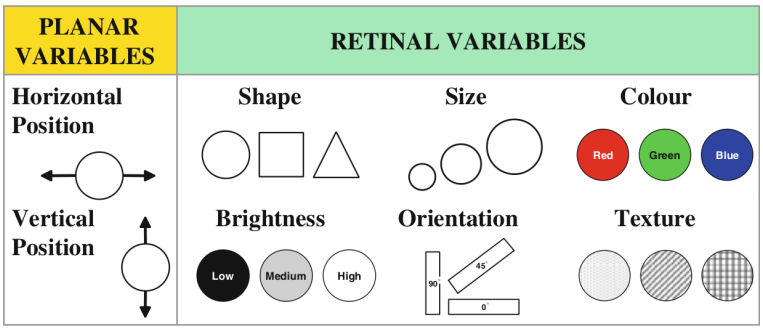
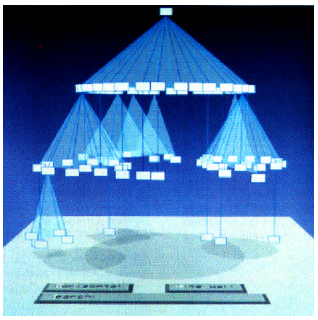



**Abbildung 2.15:** Visuelle Variablen zur Charakterisierung von visuellen Notationen; Quelle: [MHM10].

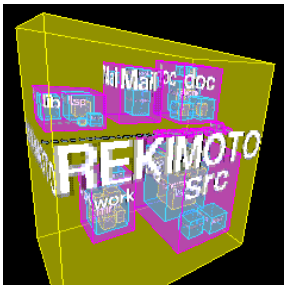
erweitert werden, die die Position auf der dritten Koordinatenachse angibt. In Abschnitt 4.7 werde ich beschreiben, welche Rolle visuelle Variablen im Kontext generischer 3D-Darstellungen für dreidimensionale Sprache spielen.

Für die meisten 3D-Visualisierungen gilt, dass sie die dritte Dimension nutzen um Darstellungen zu erreichen, die sich Metaphern aus der realen Welt bedienen oder den zur Verfügung stehenden Platz besser ausnutzen. Stasko und Wehrli haben eine Klassifikation für 3D-Visualisierungen entwickelt, mit der sich ermitteln lässt, ob eine 3D-Visualisierung die dritte Dimension gewinnbringend nutzt [SW93]. Sie unterscheiden drei verschiedene Arten von Visualisierungen: (1) erweiterte 2D-Visualisierungen, (2) adaptierte 2D-Visualisierungen und (3) inhärente 3D-Visualisierungen. Erstere nutzten die dritte Dimension nur aus ästhetischen Gründen und ziehen damit keinen gewinnbringenden Nutzen aus der dritten Dimension. Adaptierte 2D-Darstellungen nutzen sie hingegen zur Darstellung weiterer Funktionalität, die in 2D so nicht darstellbar ist. Inhärente 3D-Visualisierungen bestehen aus dreidimensionalen Entitäten und nutzen die dritte Dimension somit gewinnbringend, da deren Darstellung nur in 3D möglich ist. Diese Klassifikation ist dem Leser bereits aus Abschnitt 2.3 bekannt, wo ich gezeigt habe, dass sie auch zur Klassifikation von 3D-Sprachen eingesetzt werden kann.

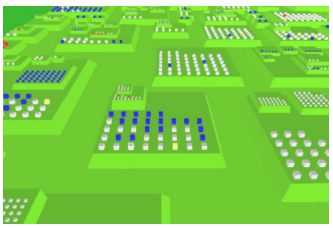
Einen guten Überblick über 3D-Visualisierungen und Systeme, die sie nutzen, geben Teyseyre und Campo [TC09]. Ich werde hier nur einige ausgewählte Aspekte aus ihrem umfassenden Überblickspapier vorstellen. Häufig werden Visualisierungen von Datensätzen benötigt, die eine hierarchische Struktur aufweisen. Aus diesem Anwendungsbereich stammen auch einige der ersten Ideen für 3D-Visualierungen, von denen drei in Abbildung 2.16 abgebildet sind. *Kegelbäume* (oder auch *Cam-Bäume*, wenn sie in horizontaler anstatt vertikaler Richtung expandieren) visualisieren Bäume in 3D, wobei die Kinderknoten auf einem Kreis



**(a)** Kegelbaum; Quelle: [RMC91].




**(b)** Informationswürfel; Quelle: [RG93].

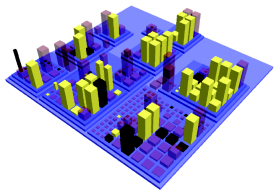


**(c)** Informationspyramiden; Quelle: [AWP97].

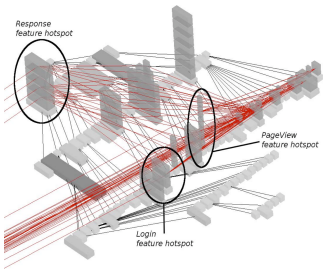
**Abbildung 2.16:** 3D-Visualisierungen zur Beschreibung von hierarchischen Strukturen.

angeordnet sind, der mit dem Vaterknoten als Spitze einen Kegel bildet [RMC91]. Sie ermöglichen eine kompakte Darstellung von großen Bäumen, wodurch der Bildschirmplatz effizient genutzt wird. Bei dem *Informationswürfel* [RG93] werden Hierarchien rekursiv ineinander geschachtelt, wobei jeder Knoten als Würfel dargestellt wird. Durch semi-transparente Darstellung äußerer Würfel kann dessen Inhalt sichtbar gemacht werden. Bei den *Informationspyramiden* wird die Hierarchie-Wurzel auf unterstem Niveau angeordnet; die Unterbäume befinden sich auf einem höheren Niveau, welches näherungsweise pyramidenförmig ist. Diese Visualisierung ist besonders bei großen Hierarchien übersichtlicher als Informationswürfel, die durch tiefe Schachtelung unübersichtlich werden können. Eine solche vergleichende Studie wurde von Wiss et al. [WCJ98] durchgeführt, die zu dem Ergebnis kamen, dass für unterschiedliche Datensätze verschiedene Visualisierungen am besten abschnitten, es also keine eindeutig zu bevorzugende Visualisierung für diesen Anwendungsfall gibt.

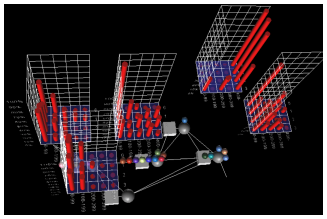
Abbildung 2.17 zeigt die Verwendung von 3D-Visualisierungen in Systemen der Software- oder Informationsvisualisierung. *Codecity* [WL07] visualisiert eine Menge von zusammenhängendem Programmcode derart, dass Pakete als Stadtteile und Klassen als Gebäude dargestellt werden. Das *TraceCrawler*-Projekt [GLW06] stellt neben dem statischen Modell eines Softwaresystems auch deren dynamisches Verhalten dar. Ausgangspunkt ist ein UML-Klassendiagramm, welches Klassen als Knoten und die Vererbungsrelationen als Kanten darstellt. Dieses wird um einen Stapel von Objektinstanziierungen und Nachrichten zwischen diesen erweitert. Ein weiteres Beispiel ist in Abbildung 2.17c die Darstellung von Geschäftsprozessen in 3D. Die Struktur der Geschäftsprozesse wird als Graph dargestellt, der um Histogramme erweitert ist, die zu einzelnen Knoten weitere Informationen bereitstellen.



**(a)** Codecity; Quelle: [WL07].

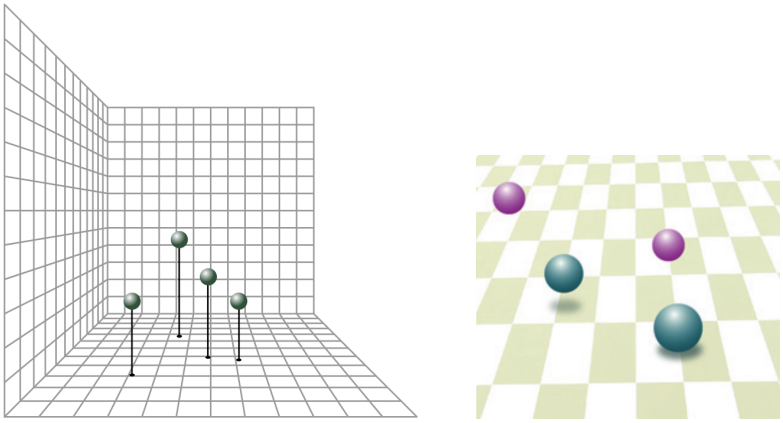



**(b)** TraceCrawler; Quelle: [GLW06].



**(c)** 3D-Geschäftsprozess-Visualisierung; Quelle: [SBE00].

**Abbildung 2.17:** Verschiedene 3D-Visualisierungssysteme.



**(a)** Fallende Linien; Quelle: [War12]. **(b)** Schatten; Quelle: [War08].

**Abbildung 2.18:** Tiefenhinweise ermöglichen eine bessere Orientierung im Raum.

Es gibt zahlreiche Untersuchungen und empirische Studien, die Vor- und Nachteile der dritten Dimension in Visualisierungen beleuchten. Einen guten Überblick gibt [TC09]: einige Untersuchungen zeigen, dass Anwender in 3D-Visualisierungen dargestellte Informationen schneller identifizieren können als in 2D-Visualisierungen. In Verbindung mit 3D-Brillen, die einen besseren Tiefeneindruck vermitteln, kann dieser Effekt noch verstärkt werden. Allerdings gibt es auch einige Probleme, die aus der 3D-Darstellung resultieren. Neben der Notwendigkeit von leistungsfähiger Hardware und aufwendiger Implementierung haben Benutzer teilweise Probleme mit der Anpassung an 3D-Visualisierungen, was zu Desorientiertheit führen kann. Dies geht einher mit doppelt so vielen *Freiheitsgraden*[14] in 3D verglichen mit 2D. Konsequenz dieser Tatsache ist, dass sich Objekte in Abhängigkeit des Blickwinkels leichter verdecken und die Werkzeuge zur Navigation in der 3D-Sicht entsprechend komplex sind, was in Summe bei manchen Benutzern den Eindruck vermittelt, dass sie den 3D-Raum nicht beherrschen.

Um dem Anwender einen besseren Eindruck von der Platzierung der Objekte im 3D-Raum zu geben, gibt es sogenannte *Tiefenhinweise*[15] [War12, S. 240ff.], [War08, S. 89ff.]. Der oben erwähnte Effekt der sich verdeckenden Objekte ist ebenfalls ein Tiefenhinweis, der andeutet, dass sich das verdeckende vor dem verdeckten Objekt befindet. Dies gilt natürlich nur, wenn noch ein Teil des weiter hinten befindlichen Objekts sichtbar ist. Weitere Hinweise auf die räumliche Platzierung gibt die Tatsache, dass weiter hinten befindliche Objekte kleiner dargestellt werden als weiter

[14] Freiheitsgrade beschreiben die Möglichkeiten der Bewegung im Raum. Im Zweidimensionalen gibt es mit der Translation in $x$- bzw. $y$-Richtung und dem Drehen um den Nullpunkt drei Freiheitsgrade. Im Dreidimensionalen gibt es hingegen sechs Freiheitsgrade: Dort können Objekte entlang der $x$-, $y$- und $z$-Achse verschoben und auch um alle diese drei Achsen rotiert werden.

[15] engl.: *depth cues*

vorne liegende. Der Verlauf einer Textur ändert sich ebenfalls mit zunehmender Tiefe. So ist in den Beispielen aus Abbildung 2.18 zu erkennen, dass die Elemente der Textur (in dem Fall die Rechtecke, die das Raster auf der Grundebene bilden) mit zunehmender Entfernung kleiner werden. All diese Tiefenhinweise ergeben sich implizit in einer 3D-Darstellung. Es gibt allerdings auch Tiefenhinweise, die Platzierungen von Objekten noch expliziter machen. So können Linien, die Objekte mit einer Grundebene verbinden, deren genaue Position deutlich besser vermitteln (siehe Abbildung 2.18a). In Abbildung 2.18b werden Schatten eingesetzt, um anzudeuten, dass sich die Objekte in der Nähe der Grundebene befinden. Diese Art der Darstellung ermöglicht auch eine Abschätzung, wie nah sich ein Objekt an der Grundebene befindet.

## 2.6 Generatorsysteme für visuelle Sprachen

In diesem Abschnitt werde ich Generatorsysteme vorstellen, die Editoren für visuelle Sprachen generieren. Für zweidimensionale visuelle Sprachen gibt es zahlreiche solcher Systeme; neben vielen universitären auch kommerzielle Projekte: *DEViL* [SKC06; SK03], *DiaGen/DiaMeta* [Min02; Min06], *VisPro* [ZZC01], *VLDesk* [CDP04], *MetaEdit+* [Met15]. Ich werde mich hier allerdings im Wesentlichen auf die Vorstellung von DEViL sowie DiaGen/DiaMeta beschränken. Die Konzepte, die DEViL nutzt, um aus Spezifikationen hohen Abstraktionsniveaus Strukturediteren zu generieren, sind Grundlage für das zu entwickelnde Generatorframework für dreidimensionale Sprachen. Die von DiaGen/DiaMeta generierten Editoren unterstützen nicht nur strukturiertes, sondern auch freies Editieren. Von besonderem Interesse ist dieses Generatorsystem im Kontext dieser Arbeit, da bisher in zwei Ansätzen versucht wurde, die mit DiaGen/DiaMeta generierten Editoren um eine dreidimensionale Ansicht zu erweitern. Einen Überblick über diese beiden Generatorframeworks hinausgehende Systeme findet der interessierte Leser in [Sch06, S. 52ff.] und [Cra10, S. 41ff.].

### 2.6.1 DEViL

Das Generatorsystem DEViL (Development Environment for Visual Languages) [Sch06] basiert auf Vorarbeiten seines Vorgängers *VL-Eli* [Jun00] und generiert aus einer Menge von Spezifikationen voll funktionsfähige Strukturediteren, die optional mit einem Analyse- und einem Codegenerierungsmodul ausgestattet sein können. Im Wesentlichen spezifiziert der Sprachentwerfer die abstrakte Syntax der visuellen Sprache und darauf aufbauend die konkrete Syntax, die die visuelle Darstellung festlegt. Für letzteren Aspekt verfügt DEViL über eine Bibliothek sogenannter *visueller Muster*, die häufig auftretende Darstellungskonzepte (wie z. B. Lis-

ten, Mengen oder Verbindungen zwischen Objekten) wiederverwendbar kapseln. Die Implementierung der konkreten Darstellung inkl. Layoutentscheidungen und Interaktionsmechanismen wird dem Sprachentwerfer damit abgenommen. Die Implementierung der visuellen Darstellung ebenso wie die der Codegeneratoren erfolgt mithilfe von Modulen des Übersetzergenerator-Frameworks *Eli* [KPJ98]. Die mit DEViL generierten Strukturediteren trennen nach dem in Abschnitt 2.1 vorgestellten Grundmodell strikt zwischen semantischer und editierbarer Struktur. Die semantische Struktur enthält redundanzfrei nur Informationen, die für die Bedeutung des Diagramms essentiell sind. Die editierbare Struktur hingegen umfasst Angaben zur Diagrammrepräsentation, wie z. B., an welchen Koordinaten Sprachkonstrukte positioniert werden.

Dieser Abschnitt wird die zentralen Aspekte des DEViL-Systems komprimiert vorstellen. Wichtig ist mir dabei die Vermittlung der wichtigsten Konzepte, die es ermöglichen aus Spezifikationen Strukturediteren zu generieren. Ich werde allerdings auf die Angabe ausführlicher Beispielspezifikationen verzichten, da dies bei Interesse bei Schmidt [Sch06] und Cramer [Cra10] nachgeschlagen werden kann. Zunächst werde ich den Prozess darstellen, mit dem DEViL Strukturediteren generiert. Danach stelle ich genauer visuelle Muster vor, die auch in dem Framework für 3D-Sprachen eine zentrale Rolle spielen werden. Abschließend werden Eigenschaften der generierten Strukturediteren hervorgehoben, auf die ich in Abgrenzung zu Editorfunktionen für 3D-Sprachen in Kapitel 6 wieder zu sprechen kommen werde.

#### Spezifikationsprozess

Einen Überblick über den Spezifikationsprozess mit DEViL liefert Abbildung 2.19. Die Grundlage einer jeden Sprachspezifikation ist die Angabe der abstrakten Struktur. Auf dieser Grundlage können eine Menge visueller Sichten und Codegeneratoren spezifiziert werden. Aus diesen Spezifikationen generiert DEViL einen Editor, mit dem Diagramme der visuellen Sprachen strukturiert konstruiert werden können.

Die abstrakte Struktur wird durch die textuelle Sprache *DSSL* (DEViL Structure Specification Language) beschrieben. Diese basiert auf objektorientierten Sprachkonzepten wie Klassen, Assoziationen, Aggregationen, Vererbung und Definition von Attributen. Ein Sprachkonstrukt der visuellen Sprache wird durch eine nicht abstrakte Klasse in der DSSL-Spezifikation repräsentiert. Die Attribute der Klassen repräsentieren primitive Werte verschiedenen Datentyps, Referenzen zu anderen Sprachkonstrukten oder Unterstrukturen.

Ein Sprachentwerfer kann mit DEViL beliebig viele visuelle Sichten für seine Sprache entwerfen, die alle auf der gleichen abstrakten Struktur basieren und

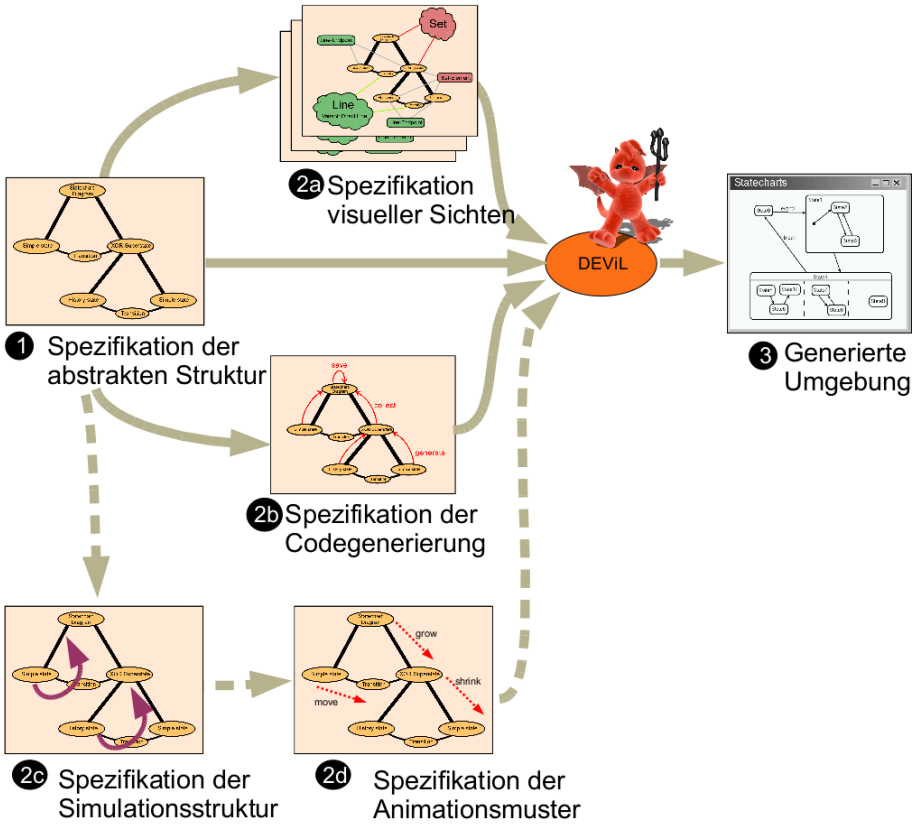



**Abbildung 2.19:** Der Spezifikationsprozess mit DEViL; Quelle: [Cra10].

diese auf unterschiedliche Weise visualisieren. Jede Sicht benötigt zur Definition der Sicht-Wurzel die Angabe eines Sprachkonstrukts der abstrakten Struktur. Eine Sicht kann dann alle Sprachkonstrukte, die sich unterhalb des Wurzelknotens befinden, darstellen. Weiterhin umfasst eine Sichtdefinition die Definition von Toolbar-Knöpfen, die es im generierten Editor erlauben neue Sprachkonstrukte in die Sicht einzufügen. Die Spezifikation der eigentlichen visuellen Darstellung wird durch die Attributierung einer kontextfreien Grammatik erreicht, die für jede Sicht und basierend auf den Sprachkonstrukten der abstrakten Struktur generiert wird. Die grafische Repräsentation der Sicht erreicht der Sprachentwerfer durch Dekoration visueller Muster an Symbole der attributierten Grammatik, die für individuelle Darstellungseigenschaften lediglich geeignet parametrisiert werden müssen. Auf visuelle Muster werde ich im nächsten Abschnitt detaillierter eingehen. Die automatisch aus der abstrakten Struktur generierte kontextfreie Grammatik genügt in einigen Fällen nicht den Erfordernissen. Dies ist der Fall, wenn semantische und editierbare Struktur voneinander abweichen, z. B. wenn zur visuellen Strukturierung zusätzliche Schachtelungsebenen benötigt werden, die die abstrakte Struktur

nicht widerspiegelt. Für solche Fälle kann in der Sichtspezifikation eine einfache Grammatik-Abbildung angegeben werden, die dann bei der Generierung der kontextfreien Grammatik berücksichtigt wird.

Ebenfalls auf Grundlage einer aus der abstrakten Struktur generierten Grammatik werden die Codegeneratoren spezifiziert. Um aus der attributierten Grammatik textuellen Zielcode zu generieren, kann der Sprachentwerfer erprobte Mechanismen des Eli-Systems verwenden. Dies umfasst neben der Spezifikation der attributierten Grammatik die Angabe von *PTG*-Mustern, die entsprechend attributiert und zusammengefügt werden, um den Zielcode zu generieren.

Die mit DEViL generierten Strukturediteren können so erweitert werden, dass die Instanzen der visuellen Sprache simuliert und animiert werden können (vgl. [CK09] und [Cra10]). Dabei ist unter Simulation eine Menge von Transformationen des semantischen Modells der visuellen Sprache zu verstehen, die von der grafischen Repräsentation abstrahiert. Die Aufgabe der Animation ist es dann, die diskreten Transformationen kontinuierlich zu visualisieren. Eine *lineare grafische Interpolation* transformiert dann die visuelle Repräsentation der Sprachkonstrukte zwischen den Simulationszuständen. Soll die Animation grafische Darstellungen umfassen, die keinen Repräsentanten in der abstrakten Struktur haben, können *dynamische Animationsobjekte* angegeben werden. In den meisten Fällen kann aus der Simulationsbeschreibung ohne weiteres Zutun des Sprachentwerfers die Animation generiert werden. Für benutzerdefinierte Anpassungen, zu denen auch die dynamischen Animationsobjekte zählen, stehen eine Vielzahl *animierter visueller Muster* zur Verfügung.

In Abschnitt 2.1 habe ich bereits den Sachverhalt angesprochen, dass Strukturediteren im Gegensatz zu Editoren, die visuelle Diagramme freihändig konstruieren, grundsätzlich syntaktische Korrektheit sicherstellen. Dies bedeutet allerdings nicht, dass die erstellten Diagramme auch semantisch korrekt sind. Um semantische Fehler abzufangen, kann der Sprachentwerfer Konsistenzprüfungen angeben, die auftretende Fehler identifizieren und dem Editornutzer anzeigen.

#### Visuelle Muster

Visuelle Muster [SK03] sind ein abstraktes Konzept, mit dem grafische Repräsentationen visueller Sprachen im DEViL-System spezifiziert werden. Sie werden auf Teile der abstrakten Struktur angewandt und legen fest, gemäß welchem Darstellungskonzept die Struktur visuell repräsentiert wird. Konzeptionell wird im DEViL-System zwischen „abstrakten“ visuellen Mustern und konkreten Implementierungsvarianten unterschieden. Abstrakte visuelle Muster sind ein Darstellungskonzept für eine bestimmte Struktur, wie z. B. eine „endliche Folge von Elementen“ oder „Menge von gleichartigen oder ähnlichen Elementen“. Erstere

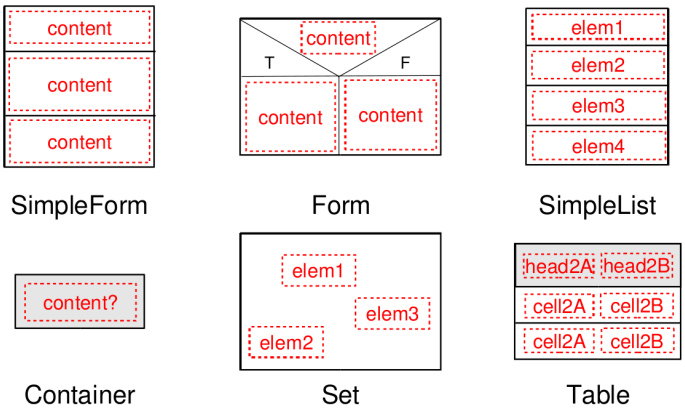



**Abbildung 2.20:** Auswahl visueller Muster, die das DEViL-System bereitstellt; Quelle: [Sch06].

könnten durch das Listen-Muster und letztere durch das Mengen-Muster dargestellt werden. Weitere Muster beschreiben Matrizen, Linienverbindungen, Graphen, Tabellen, Bäume, Container und Formulare. Abbildung 2.20 zeigt eine Auswahl schematischer Darstellungen von visuellen Mustern des DEViL-Systems.

Die Implementierungsvarianten der Muster konkretisieren die Darstellung und umfassen zusätzlich eine Layoutmethode und Interaktionsmechanismen. DEViL stellt eine Bibliothek an vorgefertigten Muster-Implementierungen bereit. Die Implementierung der Muster besteht aus einer Menge von *Berechnungsrollen*, die den aus der abstrakten Struktur abgeleiteten Grammatik-Symbolen zugeordnet werden. Die Berechnungsrollen der Muster implementieren bestimmte Schnittstellen und setzen andere Schnittstellen voraus, sodass sich Muster miteinander kombinieren lassen. Es gibt Muster, die eine Schnittstelle implementieren, die den Größenbedarf eines Objekts beschreiben und andere, die die Position eines Objekts beschreiben. Solche Muster können miteinander kombiniert werden, da zur Darstellung eines Sprachkonstrukts sowohl dessen Position als auch dessen Größe bekannt sein muss.

Eine Sonderrolle nimmt das *Formular-Muster* ein, dessen Aufgabe es ist, das visuelle Aussehen eines Sprachkonstrukts zu beschreiben. Dafür referenziert das Muster eine *generische Zeichnung*, die eine Vorlage für grafische Repräsentationen darstellt. Generische Zeichnungen werden in einem visuellen Editor spezifiziert und bestehen aus *grafischen Verzierungen*, einer Menge von *Containern*, einer Spezifikation des *Layoutverhaltens* sowie einer Menge *formaler Parameter*, die Darstellungseigenschaften festlegen. Bei der Instanziierung der Zeichnung können Container Unterstrukturen aufnehmen. Durch das Layoutverhalten wird bestimmt, wie sich die Zeichnung an die Größe der Unterstrukturen anpasst.

### Eigenschaften generierter Strukturediteren

Mit dem DEViL-System wurden eine Vielzahl von Editoren für ein breites Spektrum an visuellen Sprachen generiert: von Sprachen im industriellen Umfeld zur Steuerung von Robotern [CKK08] über regelbasierte Spiele [WCK11a] bis hin zu Sprachen klassischer Informatik-Kalküle wie Petri-Netze, erweitert um Simulationsunterstützung [CK09]. In diesem Abschnitt werde ich vor allem auf Eigenschaften der visuellen 2D-Strukturediteren eingehen, die konzeptionell auch für 3D-Editoren interessant sind sowie auf Sachverhalte, die sich bei 2D-Editoren als leicht zu realisieren erweisen, für 3D-Editoren allerdings aufwendiger umzusetzen sind. Auf einige der hier aufgegriffenen Punkte werde ich daher in Kapitel 6 wieder zu sprechen kommen. Eine Diskussion der mit DEViL generierten Editoren anhand des funktionalen Grundmodells (siehe Abschnitt 2.1) ist bei Schmidt [Sch06, S. 36f.] zu finden.

Alle diese Editoren besitzen gemeinsame Grundfunktionalitäten, die ohne Zutun des Sprachentwerfers zur Verfügung stehen: eine Multi-Fenster-Umgebung, das Laden, Speichern und Drucken von Diagrammen. Weiterhin basieren sie auf dem *Model-View-Control*-Muster, wobei die abstrakte Struktur das Modell repräsentiert, welches mit den verschiedenen Sichten gekoppelt ist. Grundsätzlich folgt die Interaktion mit dem Strukturediteur dem *direct-manipulation*-Paradigma von Shneiderman [Shn83]. Weiterhin stellt ein Editor sprachspezifische Funktionalität bereit, wie z. B. die Information, an welchen Stellen Sprachkonstrukte eingefügt werden können. Diese Funktionalität ist schon in den visuellen Mustern gekapselt, sodass der Sprachentwerfer sie nur anwenden und parametrisieren muss, wodurch ihm Grafikprogrammierung auf niedriger Ebene erspart bleibt.

Die Interaktion in den Strukturediteren ist davon geprägt, dass Sprachkonstrukte nur in einen für sie passenden Kontext eingefügt oder dahin verschoben werden können. Solche Einfügekontexte werden z. B. beim Klick auf einen Toolbar-Knopf zum Einfügen eines neuen Sprachkonstrukts angezeigt. Beim Berühren des Kontexts mit der Maus wird dieser hervorgehoben; ein Klick sorgt für das Einfügen des Sprachkonstrukts an der Stelle des Kontexts. Die Gestalt des Einfügekontexts ist an die grafische Repräsentation der beinhaltenden Sprachkonstrukte angepasst. Dies wird durch deren Kapselung in den visuellen Mustern sichergestellt. Prinzipiell lässt sich zwischen Einfügekontexten *mit* und *ohne Positionswahl* unterscheiden [Jun00, S. 144]. Bei Kontexten mit Positionswahl entscheidet der Benutzer über die endgültige Position des Sprachkonstrukts innerhalb der Zeichenfläche; bei Kontexten ohne Positionswahl hat der Benutzer keinen weitergehenden Einfluss auf die Position. Einfügekontexte für Listenelemente zählen zum letzten Fall. Sie werden als Linie zwischen den Listenelementen dargestellt. Zum Einfügen von Elementen einer Menge wird hingegen ein rechteckiger Einfügekontext bereitgestellt, inner-

halb dessen der Benutzer die Position für das Sprachkonstrukt bestimmen kann. Durch diese Art der Interaktion stellt der Editor sicher, dass die Diagramme keine inkonsistenten Zustände enthalten.

Die Navigation innerhalb der Struktureditoren ist sehr einfach. Zunächst kann zwischen den verschiedenen Sichtfenstern des Editors gewechselt werden. Die Größe des Editorfensters und der Sichtfenster kann beliebig angepasst werden und ist nur durch die Größe des Monitors beschränkt. Übersteigt die Größe der Sicht die Größe ihres Fensters, werden Scroll-Balken in zwei Dimensionen angeboten. Solch einfache Navigationsmechanismen werden für 3D-Editoren nicht mehr ausreichen, da die 3D-Szene an sich komplexer ist. Dieses Thema wird genauer in Kapitel 6 behandelt.

### 2.6.2 DiaGen/DiaMeta

DiaGen [Min01; Min02] bzw. DiaMeta [Min06] sind Rapid-Prototyping-Werkzeuge, die Diagrammeditoren generieren. DiaMeta ist der Nachfolger von DiaGen. Das Spezifikationskonzept beider Systeme ist konzeptionell sehr ähnlich, wesentlicher Unterschied ist, dass DiaGen *Hypergraphgrammatiken* zur Spezifikation der visuellen Sprache verwendet, wohingegen DiaMeta *Meta-Modelle* nutzt und auf dem *Eclipse Modeling Framework (EMF)* basiert. Die folgenden Beschreibungen beziehen sich meist konkret auf DiaMeta – da es sich dabei um das neuere System handelt – sind aber konzeptionell mit jenen zu DiaGen austauschbar.

Bei den von DiaGen/DiaMeta generierten Editoren handelt es sich grundsätzlich um Editoren mit denen visuelle Diagramme freihändig konstruiert werden. Optional kann durch Angabe weiterer Spezifikationen auch strukturiertes Editieren unterstützt werden, wodurch hybride Editoren generiert werden. Mit freihändigen Editoren ist allerdings nicht gemeint, dass der Editoranwender eine Diagrammkomponente – z. B. die kreisförmige Stelle eines Petri-Netzes – vollständig per Hand zeichnet. Stattdessen wird das visuelle Erscheinungsbild der Diagrammkomponente durch Angabe einer Spezifikation für die konkrete Syntax vom Sprachspezifizierer vorgegeben. Diese Komponenten lassen sich hingegen völlig frei und ohne Vorgabe einer bestimmten Struktur auf einer Zeichenfläche platzieren. Das vollständig freie Zeichnen von Diagrammen per Hand ist in der Literatur als *Sketching* bekannt. Auch hierzu gibt es im Umfeld des DiaMeta-Projekts Forschung [Bri09], die an dieser Stelle aber nicht weiter thematisiert werden soll.

Abbildung 2.21 stellt die Architektur eines mit DiaMeta generierten Editors dar. Die Rechtecke repräsentieren dabei funktionale Komponenten, die Ovale stellen Datenstrukturen dar. Die Verarbeitungskette eines frei konstruierten Diagramms startet beim *Modellierer* und endet beim *Model Checker*. Der Modellierer transformiert das Diagramm in ein internes Repräsentationsmodell. Dieses ist Eingabe

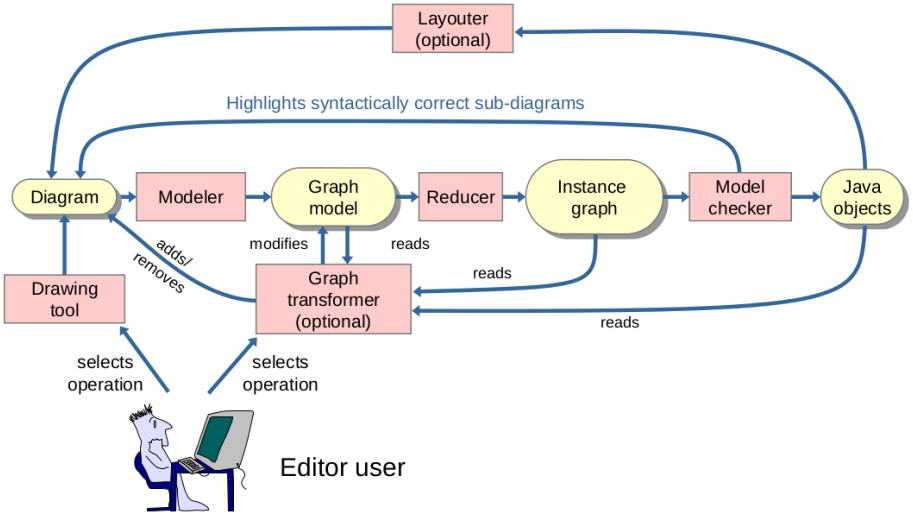



**Abbildung 2.21:** Architektur eines mit DiaMeta generierten Editors; Quelle: [Min06].

für den *Reduzierer*, der daraus den *Instanzgraphen* erstellt, der dann vom Model Checker analysiert wird. Dabei wird der maximale Teilgraph identifiziert, der syntaktisch korrekt ist. Die entsprechenden Diagrammkomponenten werden farbig hervorgehoben, sodass der Editornutzer eine visuelle Rückmeldung über die syntaktisch korrekten Teile des Diagramms erhält. Der Model Checker überprüft dabei nicht nur die abstrakte Struktur des Diagramms, sondern erstellt auch die Objektstruktur des syntaktisch korrekten Teildiagramms.

Vergleicht man diese Verarbeitungskette mit der von DEViL generierten Editoren, fällt auf, dass die Verarbeitungsschritte in umgekehrter Reihenfolge durchlaufen werden: aus einem konstruierten Diagramm wird eine semantische Struktur abgeleitet. Dies folgt direkt aus dem Ansatz freihändiger Editoren. Optional kann ein DiaMeta-Editor auch um Operationen zum strukturierten Editieren erweitert werden. Die dafür benötigten Editieroperationen – die Komponenten zum Diagramm hinzufügen bzw. entfernen – werden vom *Graphtransformierer* aufgerufen, der das Graphmodell entsprechend modifiziert. Für solch strukturiert editierbare Editoren kommt dann der *Layouter* ins Spiel, der das Layout der visuellen Darstellung bestimmt, indem er Attribute der Diagrammkomponenten modifiziert. In letzter Zeit wurde die Layoutfunktionalität von DiaMeta derart erweitert, dass verschiedene Layoutalgorithmen in einem Diagrammeditor miteinander kombiniert werden können [Mai12]. Weiterhin können mit DiaMeta generierte Sprachen auch simuliert und animiert werden [SM10].

Im Kontext der DiaGen- bzw. DiaMeta-Projekte gab es bisher im Rahmen von Diplomarbeiten zwei Anläufe, auch dreidimensionale Editoren zu generieren. Albe [Alb04] hat prototypisch für mit DiaGen generierte Editoren eine zusätzliche 3D-

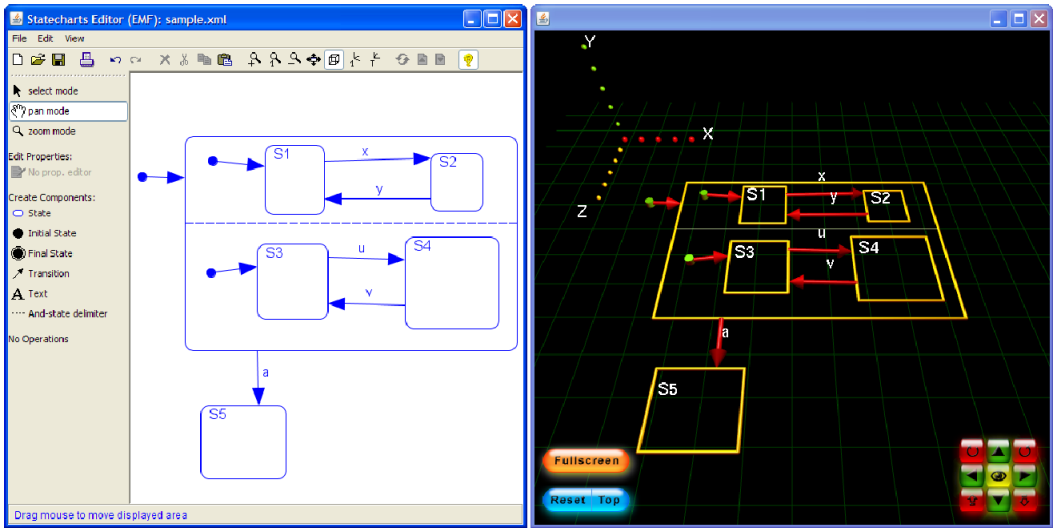



**Abbildung 2.22:** Auf 3D erweitertes Zustandsdiagramm generiert mit DiaMeta; Quelle: [Voß09, S. 81].

Zeichenschnittstelle implementiert. Leider vermittelt die Arbeit keinen Eindruck davon, welche Art 3D-Sprachen damit generiert werden können. Außerdem erscheint die Interaktion zum Einfügen von 3D-Objekten sehr aufwendig: Für ein einzufügendes Objekt müssen mehrere Punkte im dreidimensionalen Raum festgelegt werden. So muss für ein kugelförmiges Objekt der Mittelpunkt und deren Ausdehnung angegeben werden; zum Einfügen eines quaderförmigen Objekts wird die Angabe seiner Eckpunkte benötigt.

Voß [Voß09] hat den zweiten Anlauf unternommen, die mit DiaMeta generierten Editoren mit einer 3D-Darstellung auszustatten. Er präsentiert eine Reihe von generierten Editoren, die neben der 2D-Sicht nach seiner Erweiterung auch eine 3D-Sicht bereitstellen. Bei den meisten dieser Editoren ist allerdings ein gewinnbringender Einsatz der dritten Dimension nicht gegeben. Ein Beispiel hierfür ist das in Abbildung 2.22 dargestellte Zustandsdiagramm. Unter Anwendung der Klassifikation für dreidimensionale Sprachen aus Abschnitt 2.3 nutzen die meisten dieser Sprachen lediglich eine erweiterte 2D-Darstellung.

## 2.7 Zusammenfassung

Der erste Abschnitt dieses Kapitels gab einen Überblick über visuelle Sprachen im Allgemeinen und stellte Strukturediteren vor, mit denen visuelle Diagramme unter Einhaltung der syntaktischen Korrektheit strukturiert konstruiert werden. Danach wurden dreidimensionale visuelle Sprachen vorgestellt. Dafür wurden zahlreiche in der Literatur beschriebene Beispiele für 3D-Sprachen präsentiert und

beleuchtet, welchem Zweck die dritte Dimension dient. Der dritte Abschnitt ging systematisch der Frage nach dem Nutzen der dritten Dimension in visuellen Sprachen nach. Die meisten der vorgestellten 3D-Sprachen verwenden eine inhärent dreidimensionale Darstellung oder nutzen die dritte Dimension, indem ihr eine semantische Bedeutung zugeordnet wird. Solche Kriterien umfassen auch die präsentierte Klassifikation für 3D-Sprachen, die zusätzlich noch Kriterien einbezieht, die aus einer Klassifikation für 3D-Visualisierungen stammen.

Da die Strukturediteren, die mit dem Generatorsystem, welches in dieser Arbeit entwickelt wird, generiert werden, Techniken zur Interaktion und Navigation im 3D-Raum benötigen, wurden im vierten Abschnitt ausgewählte 3D-Editoren aus anderen Anwendungsbereichen vorgestellt und herausgearbeitet, welche Interaktionstechniken sie anbieten. Danach wurden 3D-Visualisierungen beschrieben und typische Konzepte zur Anordnung von 3D-Entitäten vorgestellt. Von besonderer Bedeutung sind außerdem die visuellen Variablen, mit denen sich visuelle Notationen charakterisieren lassen, und Tiefenhinweise, mit denen die Orientierung im 3D-Raum verbessert wird.

Der letzte Abschnitt stellte Generatorsysteme für visuelle Sprachen vor. Den größten Raum nahm die Beschreibung des DEViL-Systems ein, aus welchem wichtige Konzepte für ein Generatorsystem für 3D-Sprachen übernommen wurden. Ein zentraler Aspekt sind dabei die visuellen Muster, mit welchen ein Sprachentwerfer die visuelle Repräsentation einer visuellen Sprache auf hohem Niveau spezifizieren kann. Weiterhin wurde das DiaMeta-System vorgestellt, in dessen Kontext der bisher einzige Versuch unternommen wurde, generierte Editoren für visuelle Sprachen mit einer 3D-Darstellung auszustatten.

KAPITEL 3

# Ein Generatorsystem für dreidimensionale visuelle Sprachen

Die meisten in Abschnitt 2.2 präsentierten 3D-Sprachen wurden in der zweiten Hälfte der 1990er Jahre vorgestellt und prototypisch implementiert. Wie gut bedienbar die für die Sprachen entwickelten 3D-Editoren waren, lässt sich oftmals nicht genau nachvollziehen. Mark Najork, der Entwickler von Cube, gesteht jedenfalls ein, dass der entwickelte Editor nur von ihm selbst zu bedienen ist [Naj96, S. 238].

Die Anforderungen an die Entwicklung von 3D-Spracheditoren sind allerdings auch umfangreich und reichen von Techniken aus dem Übersetzerbau bis zu Anforderungen wie Interaktion und Konstruktion von 3D-Grafiken. Das macht den Aufwand zur Implementierung eines Editors für ein spezielles Anwendungsgebiet unangemessen hoch. Diese Hürde kann überwunden werden, wenn gut benutzbare 3D-Editoren von Generatorsystemen automatisiert hergestellt werden, so wie es für zweidimensionale Sprachen erfolgreich praktiziert wird (siehe Abschnitt 2.6).

In der vorliegenden Arbeit wurde ein Generatorsystem entwickelt, welches 3D-Strukturediteren generiert und Methoden und Werkzeuge zur Entwicklung dreidimensionaler visueller Sprachen wiederverwendbar kapselt. Als Vorbild dient dabei das Generatorsystem DEViL (siehe Abschnitt 2.6.1), welches Strukturediteren für zweidimensionale visuelle Sprachen generiert. Nach gründlicher Analyse konnten einige Spezifikationskonzepte von DEViL übertragen werden, um auch dreidimensionale Sprachen automatisch herzustellen. Dies gilt z. B. für das Konzept der visuellen Muster. Andere Aspekte wie die Konstruktion, Navigation und Interaktion mit 3D-Grafiken sind hingegen vollkommen neu.

Die DiaGen/DiaMeta-Systeme wurden so erweitert, dass die damit generierten Editoren das visuelle Diagramm auch in 3D darstellen können. Die so spezifizierten Sprachen verwenden allerdings überwiegend erweiterte 2D-Darstellungen. Mit DEViL3D lassen sich hingegen auch Sprachen mit inhärenter 3D-Darstellung spezifizieren, sodass DEViL3D das erste umfassende Generatorsystem von Strukturediteren für dreidimensionale Sprachen ist. Dieses Kapitel dient dazu, einen Überblick über das entwickelte Generatorsystem und dessen Konzepte zu geben. Zunächst werde ich dafür die Besonderheiten und Herausforderungen dreidimensionaler Sprachen beleuchten. Danach werde ich vorstellen, welche Schritte für einen Sprachentwickler notwendig sind, um mit DEViL3D eine 3D-Sprache zu spezifizieren. Das Kapitel endet mit einer Systemübersicht von DEViL3D.

## 3.1 Besonderheiten und Herausforderungen dreidimensionaler Sprachen

Wie bereits die Klassifikation von 3D-Sprachen in Abschnitt 2.3 aufgezeigt hat, ziehen 3D-Sprachen ihren Nutzen aus einer komplexen Darstellungsweise, die mit zweidimensionalen Sprachen nicht so einfach zu realisieren ist. 3D-Sprachen nutzen die dritte Dimension in bestimmter Weise und ordnen Sprachkonstrukte im 3D-Raum meist so an, dass deren Position eine semantische Bedeutung hat. Aufgrund der zusätzlichen Dimension bezeichnet Najork [Naj96] 3D-Sprachen im Vergleich zu 2D-Sprachen als syntaktisch reicher. Doch neben der Semantik von visuellen 3D-Sprachen gibt es darüber hinausgehende Besonderheiten und Herausforderungen der Sprachen bzw. der zur Konstruktion verwendeten Editoren.

Betrachtet man die Konstruktion von visuellen Diagrammen, so fällt auf, dass die Konstruktion von 2D-Diagrammen für den Editornutzer besonders deshalb einfach ist, weil er aus der dritten Dimension heraus ein Konstruktionswerkzeug auf der 2D-Zeichenfläche bedient. Bei der Konstruktion im 3D-Raum befindet sich der Konstrukteur in demselben 3D-Raum, in dem er 3D-Objekte konstruiert. Das wird besonders bei Verwendung von speziellen Ausgabegeräten wie *Head-Mounted-Displays*[1] offensichtlich. Zur Bewerkstelligung der größeren Freiheiten im 3D-Raum sind deshalb Hilfen zur Orientierung notwendig. 3D-Spracheditoren umfassen daher spezielle Navigationstechniken, mit deren Hilfe das 3D-Diagramm einfach aus verschiedenen Perspektiven betrachtet werden kann. Diese nutzen alle eine virtuelle Kamera, die vom Anwender gesteuert wird, um einen individuellen

[1] Head-Mounted-Displays werden am Kopf des Anwenders befestigt und koppeln Kopfbewegungen mit dem Blickwinkel der virtuellen Kamera, durch die die 3D-Szene betrachtet wird. Sie werden besonders in *virtuellen Realitäten* verwendet, um die Wahrnehmung der 3D-Szene zu verstärken (Beispiele für solche Systeme wurden von Osawa et al. [OASS01] und Biermann [Bie06] entworfen).

Blickwinkel auf die 3D-Szene festzulegen. Unterstützend dazu können zur Steigerung der 3D-Wahrnehmung stereoskopische Verfahren angewendet werden. Die Navigation in Editoren für 2D-Sprachen hingegen hat keine bedeutende Funktionalität und ist meist auf das Scrollen entlang der zwei Raumdimensionen limitiert. In Kapitel 6 wird genauer auf alle Aspekte der Navigation eingegangen.

Durch 3D-Sprachen lässt sich weiterhin das Bildschirmplatz-Problem [BBB+95] minimieren, welches gelegentlich bei visuellen 2D-Sprachen beobachtet wird und sich dadurch äußert, dass oftmals raumgreifende visuelle Diagramme nicht vollständig auf einem Bildschirm angezeigt werden können. Dieser Effekt wird auch durch das sogenannte *Deutsch-Limit* [McI98, Frage 12] beschrieben, welches aussagt, dass gleichzeitig nur 50 Sprachkonstrukte auf dem Bildschirm dargestellt werden können. Bei 3D-Sprachen können aufgrund der zusätzlichen Raumdimension mehr Sprachkonstrukte angezeigt werden. Dabei kann es allerdings schnell dazu kommen, dass sich Sprachkonstrukte gegenseitig verdecken, was durch die oben beschriebenen Navigationstechniken kompensiert werden kann.

Das kognitive System des Menschen kann dreidimensionale Darstellungen effizienter verarbeiten. Laut Van Reeth und Flerackers [RF93] hat das mentale Modell, das Entwickler von einem Diagramm und dessen Anwendungsgebiet haben, Verbindungen in die reale 3D-Welt. 3D-Sprachen können also diese Verbindung zusätzlich fördern; dies gilt insbesondere für inhärente 3D-Sprachen, die Objekte aus der realen Welt nutzen. Weiterhin haben Ware und Franck [WF94] ermittelt, dass aus der Betrachtung von Graphen in 3D mittels stereoskopischer Verfahren dreimal mehr Information gewonnen werden kann als aus einer 2D-Repräsentation des gleichen Graphen. Die Ergebnisse von Schoeffmann et al. [SAH14] zeigen, dass das Suchen in Fotosammlungen, die als 3D-Ring oder 3D-Globus dargestellt werden, weniger Zeit benötigt, als in einer 2D-Grid-Darstellung.

Der Aspekt der *Schachtelung* findet sich auch in textuellen Programmier- und Auszeichnungssprachen. Dort wird die Schachtelung durch Einsetzen von Textblöcken in umfassende Blöcke dargestellt, wobei die Blöcke durch eine Klammerung – dargestellt durch Spezialsymbole, Wortsymbole, Tags oder Einrückung – hervorgehoben werden. In 3D-Sprachen wird Schachtelung sehr natürlich durch vollständiges Umschließen eines Sprachkonstrukts durch ein anderes ausgedrückt. Dadurch wird meist eine Enthaltensein-Relation charakterisiert. Mithilfe spezieller Navigationstechniken können eingeschachtelte Objekte – wie Räume in der realen Welt – betreten werden. In Abschnitt 6.3 werden Interaktionstechniken für geschachtelte Strukturen vorgestellt.

Es gibt allerdings bei 3D-Sprachen auch neue Herausforderungen und Schwierigkeiten. So ist die Wiedererkennung von 3D-Objekten laut Tarr et al. [TWHG98] blickrichtungsabhängig. Das unterstreicht abermals die Wichtigkeit der Navigation innerhalb der verwendeten Strukturediteren. Weiterhin sind klassische Ausdru-

cke von 3D-Diagrammen auf Papier nur eingeschränkt möglich, da nur bestimmte Blickwinkel auf die 3D-Szene, aber nie die ganze Szene als solche abgebildet werden kann. Theoretisch sind allerdings auch Ausdrucke mit einem 3D-Drucker möglich. Inwieweit dies sinnvoll ist, ist eine andere Frage. Zum einen sind 3D-Diagramme nicht vollständig zusammenhängend und somit würde der Druck aus vielen unzusammenhängenden Sprachkonstrukten bestehen. Zum anderen lässt sich der Aspekt der Schachtelung durch einen solchen Druck nur schwer nachbilden.

Wie die meisten visuellen 2D-Sprachen werden auch 3D-Sprachen in domänenspezifischen Anwendungsbereichen sinnvoll eingesetzt. Ein gutes Beispiel liefern die Molekülmodelle, deren Darstellung in 3D deutliche Vorteile gegenüber einer auf 2D reduzierten Darstellung hat. Daher beschäftigt sich der nachfolgende Abschnitt mit der Spezifikation einer 3D-Sprache mit DEViL3D am Beispiel der Molekülmodelle.

## 3.2 Spezifikation einer 3D-Sprache am Beispiel Molekülmodelle

Dieser Abschnitt soll kurz die Benutzung von DEViL3D aus Sicht eines Sprachentwicklers skizzieren, der einen Struktureditor für die in Abschnitt 2.2.9 kennengelernten Molekülmodelle spezifizieren möchte. Für sie soll von DEViL3D ein Struktureditor generiert werden, der es ermöglicht, Kugel-Stäbchen-Modelle von Molekülen zu konstruieren. Der generierte Editor ist in Abbildung 3.1 zu sehen. Der Editornutzer kann damit Atome in den 3D-Raum einfügen und anschließend miteinander verbinden. Neben dem Einfügen von Sprachkonstrukten und Interagieren mit dem 3D-Diagramm stellt der Struktureditor auch Werkzeuge zur Navigation bereit, die es erlauben, die 3D-Szene aus verschiedenen Blickwinkeln zu betrachten (siehe Kapitel 6). Ist ein Molekül durch Einfügen der Atome und ihrer Bindungen strukturell konstruiert, ist der Editor in der Lage, das Layout des Moleküls zu berechnen, sodass es den chemischen Regeln genügt (solchen sprachspezifischen Layoutaspekten widmet sich Abschnitt 5.5).

Der Prozess zur Spezifikation einer 3D-Sprache mit DEViL3D ist in Abbildung 3.2 dargestellt. Zentral ist die Spezifikation der abstrakten Struktur, die das semantische Modell der Sprache definiert. Auf dieser baut die visuelle Darstellung sowie die Analyse und Codegenerierung auf. Im Folgenden werde ich Ausschnitte der Spezifikationen für die abstrakte Struktur und die visuelle Darstellung vorstellen. Eine Spezifikation für die Codegenerierung werde ich nicht extra angeben, da es sich dabei um eine klassische Anwendung des Eli-Systems mit Spezifikation eines Attributauswerters und Aufruf von PTG-Mustern handelt.

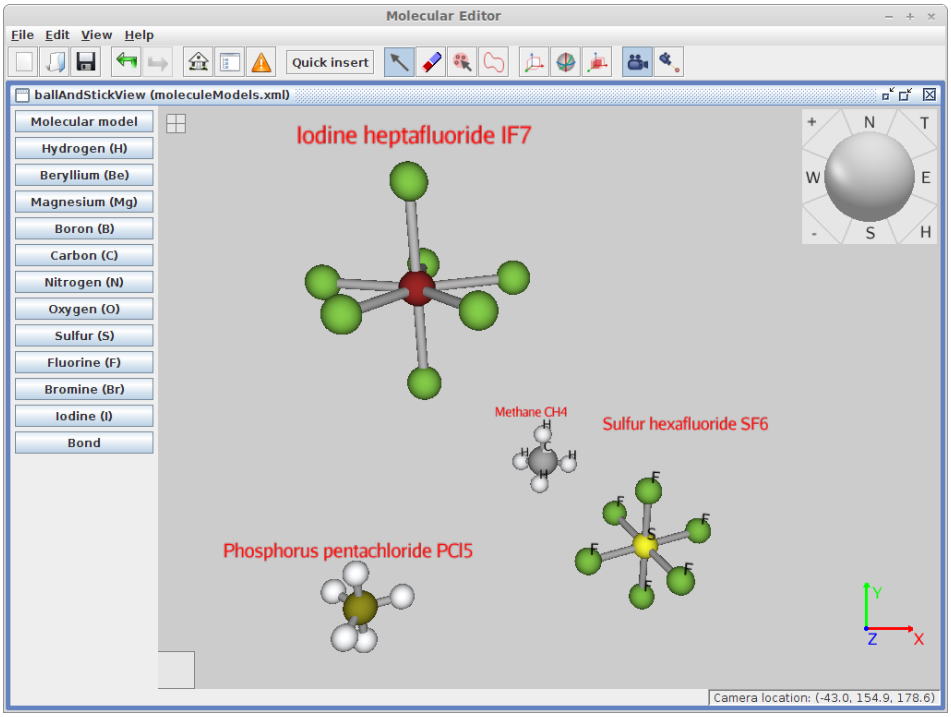



**Abbildung 3.1:** Der aus Spezifikationen generierte Moleküleditor.

Quelltext 3.1 zeigt einen Ausschnitt der abstrakten Struktur für Molekülmodelle. DEViL3D stellt dafür (wie bereits DEViL) die Spezifikationssprache *DSSL* (DEViL Structure Specification Language) bereit. DSSL ist objektorientiert und modelliert die Struktur der Sprache daher mit Klassen, die eine Menge von Attributen besitzen. Der spezifizierte Moleküleditor erlaubt die Angabe mehrerer Molekülmodelle in einem Diagramm (Zeile 2). Diese bestehen wiederum aus beliebig vielen Atomen und Bindungen (Zeilen 6, 7). Die SUB-Attribute definieren Baumkanten innerhalb des Strukturbaums und speichern Unterstrukturen bestimmten Typs. Verschiedene Arten von Atomen werden zu einer abstrakten Klasse zusam-

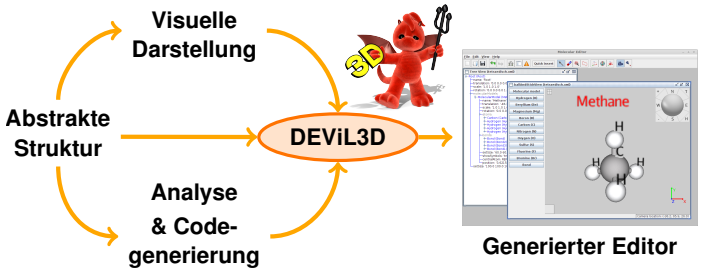



**Abbildung 3.2:** Spezifikationsprozess mit DEViL3D.

```
1  CLASS Root {
2    molecularModels: SUB MolecularModel*;
3  }
4
5  CLASS MolecularModel {
6    atoms: SUB Atom*;
7    bonds: SUB Bond*;
8  }
9
10 ABSTRACT CLASS Atom {
11   position: VAL VLVector3f;
12   freeValenceElectrons: VAL VLInt;
13   covalentRadius: VAL VLInt;
14 }
15
16 ABSTRACT CLASS AlkaliMetalAtom INHERITS Atom {
17   valenceElectrons: VAL VLInt INIT "1";
18 }
19
20 ABSTRACT CLASS CarbonGroupAtom INHERITS Atom {
21   valenceElectrons: VAL VLInt INIT "4";
22 }
23
24 CLASS Hydrogen INHERITS AlkaliMetalAtom {}
25
26 CLASS Carbon INHERITS CarbonGroupAtom {}
27
28 CLASS Bond {
29   from: REF Atom;
30   to: REF Atom;
31 }
```

**Quelltext 3.1:** Ausschnitt der abstrakten Struktur für Molekülmodelle.

mengefasst, die u. a. deren Position innerhalb des 3D-Raums enthalten. Die dafür zuständigen `VAL`-Attribute speichern primitive Werte verschiedener Datentypen, wie z. B. `VLVector3f` für einen dreidimensionalen Vektor. In der Struktur wird die Zugehörigkeit der Atome zu einer Gruppe des Periodensystems durch abstrakte Zwischenklassen, die die Anzahl der Valenzelektronen der Gruppe angeben, modelliert (Zeilen 16–22). Die Molekülbindungen umfassen zwei `REF`-Attribute, die Referenzen auf die zu verbindenden Atome besitzen.

Aus der abstrakten Struktur generiert DEViL3D eine kontextfreie Grammatik, die wiederum in eine attributierte Grammatik transformiert wird. Die Spezifikation der visuellen Darstellung basiert auf dieser attributierten Grammatik, die Berechnungen zur visuellen Darstellung initiiert und Sprachkonstrukte im 3D-Raum platziert. Quelltext 3.2 zeigt einen Ausschnitt der Spezifikation der visuellen Dar-

```
1 SYMBOL ballAndStickView_Hydrogen INHERITS VP3DForm,
      VP3DSetElement, VPConnectionEndpoint, showSymbolNames
2 COMPUTE
3   SYNT.depiction = DEPICTION(HydrogenDepiction);
4   SYNT.symbolName = "H";
5 END;
6
7 SYMBOL ballAndStickView_Bond INHERITS VPConnection
8 COMPUTE
9   SYNT.radius = 1;
10 END;
11
12 SYMBOL ballAndStickView_Bond_from INHERITS VPConnectionFrom
13 COMPUTE
14 END;
```

**Quelltext 3.2:** Spezifikations-Ausschnitt der visuellen Darstellung für Molekülmodelle.

stellung für Molekülmodelle in der Sprache *LIDO*[2]. Die Bezeichner der Symbolrollen setzen sich aus dem Namen der visuellen Sicht (im Beispiel `ballAndStickView`) und einem in der abstrakten Struktur definierten Sprachkonstrukt- bzw. Attribut-Literal zusammen. Die Symbolrollen erben alle Eigenschaften, die in den Berechnungsrollen der visuellen Muster definiert sind; so z. B. das Hydrogen-Atom von `VP3DForm` und `VP3DSetElement`. Das Formular-Muster ordnet einem Sprachkonstrukt eine visuelle Repräsentation in Form einer generischen 3D-Darstellung zu. Dies wird in Zeile 3 durch Überschreiben des Kontrollattributs `depiction` realisiert.

```
1 def check_Hydrogen(atom: StructureObject): String = {
2   var nbrFreeValEl: Integer = TreepathEvaluator.get(atom, "THIS.
        freeValenceElectrons.VALUE")
3   var nbrValEl: Integer = TreepathEvaluator.get(atom, "THIS.
        valenceElectrons.VALUE")
4   if(nbrFreeValEl == nbrValEl) {
5     return "Atom '" + atom.getName + "' is not connected with any
          other atom."
6   }
7   return ""
8 }
```

**Quelltext 3.3:** Konsistenzprüfung, ob es ungebundene Atome gibt.

[2]Die Sprache LIDO ist Teil des Eli-Systems und wird zur Spezifikation attributierter Grammatiken verwendet.

Die abstrakte Struktur und die visuelle Darstellung sind die wichtigsten Teile der Spezifikation für einen Strukturediteor, der automatisch Interaktions- und Navigationstechniken unterstützt. Darüber hinaus stellt DEViL3D weitere Spezifikationsmodule bereit, durch deren Anwendung der generierte Editor semantische Anforderungen der Sprache sicherstellt oder auf Verletzungen von Sprachanforderungen hinweist. In Quelltext 3.3 ist eine *Konsistenzprüfung* zu sehen, die prüft, ob es ungebundene Atome gibt. Eine Funktion zur Konsistenzprüfung beginnt mit `check_` gefolgt von dem Namen des entsprechenden Sprachkonstrukts. Der Funktion wird die zu prüfende Instanz des Sprachkonstrukts übergeben. Innerhalb des Strukturbaums lässt sich mithilfe von *Pfadausdrücken* navigieren (siehe Zeilen 2, 3). Neben den Konsistenzprüfungen können *Synchronisations-Funktionen* formuliert werden, die nach jeder Änderung am 3D-Diagramm aufgerufen werden und spezielle Abhängigkeiten zwischen verschiedenen Kontexten im Strukturbaum sicherstellen. Außerdem können *Initialisierungs-Funktionen* angegeben werden, die einmalig beim Instanziieren eines Sprachkonstrukts aufgerufen werden. Darüber hinaus können noch *benutzerdefinierte Funktionen* definiert werden, die im generierten Strukturediteor über das Kontextmenü eines Sprachkonstrukts aufgerufen werden können. Diese wurden beim Moleküleditor z. B. genutzt, um die Berechnung des sprachspezifischen Layouts anzustoßen (eine ausführliche Beschreibung dazu folgt in Abschnitt 5.5.1).

## 3.3 Systemübersicht

Nachdem der letzte Abschnitt die Anwendung von DEViL3D skizziert hat, wird nun eine Systemübersicht des entwickelten Generatorsystems vorgestellt. Konzeptionell ist DEViL3D an DEViL angelehnt, wurde aber von Grund auf neu implementiert. Im Vorhinein wurde gründlich analysiert, welche Teile wiederverwendet und angepasst werden können oder vollständig neu implementiert werden müssen.

Die Sprache DSSL zur Beschreibung der abstrakten Struktur konnte fast unverändert übernommen werden. Nur die Typen der `VAL`-Attribute mussten erweitert werden, um z. B. dreidimensionale Vektoren beschreiben zu können. Die Schnittstelle zur Spezifikation eines Codegenerators konnte ebenfalls übernommen werden. Alle weiteren Aspekte konnten höchstens konzeptionell, aber mit einigen bedeutenden Anpassungen übernommen werden. Die Berechnung der visuellen Darstellung basiert, wie bei DEViL, auf attributierten Grammatiken. Allerdings waren substanzielle Änderungen notwendig, um der Weiterentwicklung in die Dreidimensionalität Rechnung zu tragen. So wurden die visuellen Muster vollständig neu implementiert, um 3D-Darstellungskonzepte zu berücksichtigen (siehe Kapitel 4). Das Rahmenwerk der generierten Strukturediteoren, also das

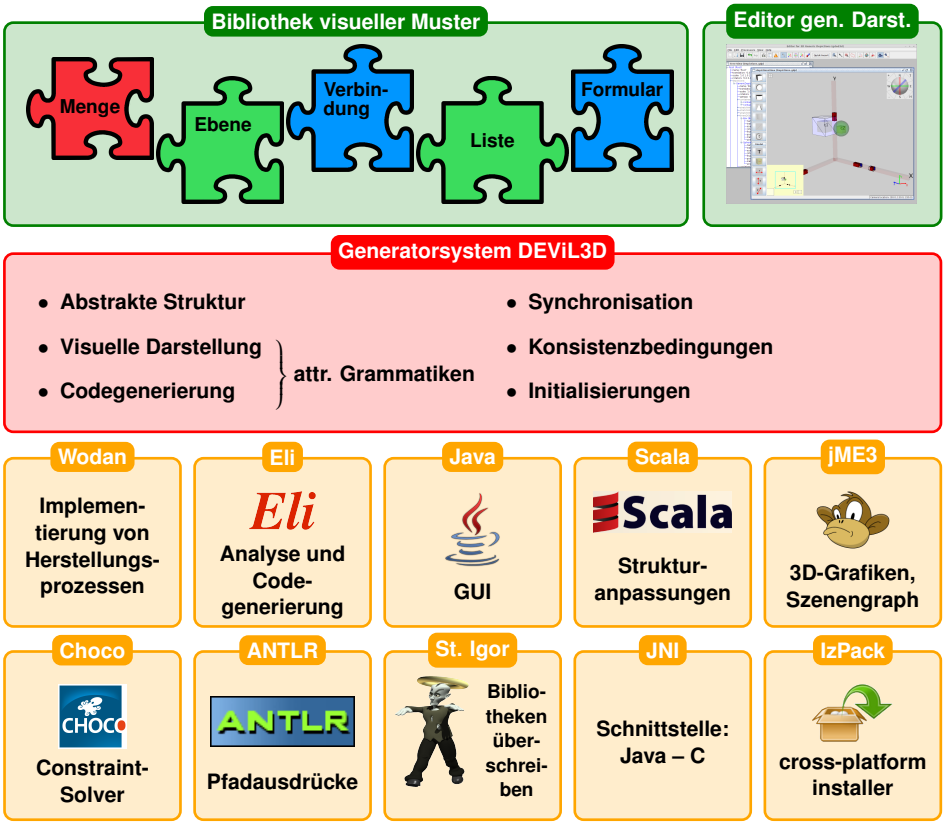



**Abbildung 3.3:** Aufbau des Generatorsystems DEViL3D.

Frontend einer 3D-Sprache, wurde grundlegend neu konzipiert. In diesen Bereich fällt die Darstellung von 3D-Szenen, mit denen der Editoranwender interagieren und darin navigieren kann (siehe Kapitel 6).

Abbildung 3.3 visualisiert den Aufbau des Generatorsystems DEViL3D in einer schichtenbasierten Darstellung. In der Mitte befindet sich das eigentliche Generatorsystem mit den wichtigsten Bestandteilen einer Sprachspezifikation. In der darüber liegenden Schicht sind bedeutende Bibliotheken und Werkzeuge skizziert, die wiederverwendbares Wissen kapseln und Sprachentwerfern die Spezifikation einer 3D-Sprache mit wenig Aufwand ermöglichen. Dies sind zum einen die Menge der visuellen Muster und zum anderen der Editor für generische 3D-Darstellungen, die das visuelle Aussehen der Sprachkonstrukte definieren. Auf unterster Ebene befinden sich Werkzeuge, Bibliotheken und Sprachen, die angewendet wurden, um DEViL3D zu implementieren. Deren Aufgabe und Verwendung werde ich im Folgenden kurz beschreiben:

**Wodan** ist eine von Carsten Schmidt in Python geschriebene Bibliothek zur Implementierung von Herstellungsprozessen. Damit hat es Gemeinsamkeiten

mit dem Standard-Unix-Werkzeug *Make*, welches zur Beschreibung von Abhängigkeiten und Generierungsschritten für konkrete Problem-Instanzen genutzt wird. Wodan wurde so konzipiert, um das Herstellungswissen zu einer Problem-Klasse allgemein zu kapseln. In DEViL3D ist Wodan dafür zuständig, aus der Menge der Sprachspezifikationen den 3D-Struktureditor zu generieren.

**Eli** [KPJ98] wird in DEViL3D in vielfältiger Weise genutzt: Die domänenspezifische Sprache zur Beschreibung der abstrakten Struktur ist mit Eli entworfen worden. Weiterhin sind alle im generierten Editor verwendeten Attributauswerter mit Eli spezifiziert. Der Sprachentwerfer nutzt die Sprache LIDO, die Teil des Eli-Systems ist, um die visuelle Darstellung und Codegenerierung zu spezifizieren. Für den letzten Aspekt wird auch das PTG-Werkzeug (*pattern-based Text Generator*) verwendet.

**Java** wird verwendet, um die GUI des Strukturediteors zu implementieren. Dies bildet ein gemeinsames Rahmenwerk eines jeden mit DEViL3D spezifizierten Editors. Ein fertig generierter Editor ist daher auch eine Java-Anwendung.

**Scala** ist eine objektorientierte und funktionale Programmiersprache, die in der *Java Virtual Machine* (JVM) läuft. Somit ist eine Integration in Java-Anwendungen einfach realisierbar. Sämtliche Funktionen zur Anpassung und Manipulation der Sprachstruktur (z. B. die Synchronisation) sind in Scala geschrieben. Für einige dieser Aufgaben waren Scalas funktionale Spracheigenschaften wie z. B. Funktionen höherer Ordnung besonders nützlich.

**jME3** (jMonkeyEngine 3) [Kus13] ist eine in Java geschriebene 3D-Grafik-API, mit der in DEViL3D alle Aspekte der 3D-Darstellung implementiert wurden. Insbesondere stellt jME3 einen *Szenengraphen* bereit, mit dem Objekte der 3D-Szene strukturiert werden können. jME3 ist schichtenbasiert und nutzt *OpenGL* zum Rendern der 3D-Szenen.

**Choco** ist eine Java-Bibliothek für Constraint-Programmierung.[3] Damit können Constraints formuliert und vom integrierten Constraint-Solver gelöst werden. Anwendung findet Choco für einige Layoutaufgaben (siehe Abschnitt 5.4).

**ANTLR** ist ein Parsergenerator, der in DEViL3D dafür eingesetzt wurde, die Pfadausdrücke zu modellieren, mit denen in der abstrakten Struktur navigiert werden kann.[4]

[3] `http://choco-solver.org/`
[4] `http://www.antlr.org/`

**St. Igor** ist eine Java-Bibliothek, die auf der *String Template Library* von ANTLR basiert. St. Igor wird in DEViL3D dafür genutzt, damit der Sprachspezifizierer möglichst einfach Funktionen zur Strukturanpassung überschreiben kann.

**JNI** (*Java Native Interface*) dient als Kommunikationsschnittstelle zwischen Programmen in der JVM und den Attributauswertern, die von Eli als C-Implementierung generiert werden.

**IzPack** ist eine Anwendung, mit der ein Installationsprogramm für einen mit DEViL3D generierten Editor erstellt werden kann, um diesen dann auf anderen Computern einfach installieren zu können.

KAPITEL 4

# Dreidimensionale visuelle Muster

Dreidimensionale visuelle Muster sind bei der Spezifikation einer 3D-Sprache mit DEViL3D die Schlüsselkomponente. Mit ihnen lässt sich auf Basis der vorher spezifizierten abstrakten Syntax sehr einfach das visuelle Frontend der Sprache spezifizieren. Dabei kapseln die visuellen Muster alle Aspekte, die für die visuelle Repräsentation der Sprache von Belang sind. Dies umfasst neben dem durch das Muster festgelegte Anordnungskonzept auch Layouteigenschaften sowie Mechanismen zur Interaktion mit der 3D-Darstellung.

Ziel war es für dreidimensionale Sprachen einen möglichst vollständigen Satz an visuellen Mustern zu identifizieren und anschließend zu implementieren, der in der Lage ist ein breites Spektrum visueller 3D-Sprachen syntaktisch zu beschreiben. Die einzelnen visuellen Muster wurden durch Untersuchung schon existierender 3D-Sprachen und durch Erweiterung von schon im DEViL-System existierenden Mustern für zweidimensionale visuelle Sprachen auf den dreidimensionalen Fall ermittelt. Die implementierten Muster stehen als Bibliothek von Berechnungsrollen zur Verfügung. Sprachentwickler können diese an Symbole der, aus abstrakter Struktur abgeleiteten, attributierten Grammatik dekorieren. Das macht den Spezifikationsaufwand für DEViL3D-Nutzer besonders einfach, da die Anwendung eines Musters automatisch sicherstellt, dass die ihr zugeordneten Sprachkonstrukte dem vom Muster definierten Darstellungskonzept folgen und dem Benutzer des Strukturediteors maßgeschneiderte Interaktionstechniken bereitstellen.

Entwickelt wurde das Konzept der visuellen Muster initial für VL-Eli [SS00; Jun00; SK03]. Für das DEViL-System wurde es um ein Modell zur Kombinierbarkeit der Muster sowie um weitere konkrete visuelle Muster erweitert [Sch06]. Für das DEViL3D-System habe ich das Konzept der visuellen Muster auf dreidimensionale visuelle Sprachen übertragen. Für alle drei Systeme sind die visuellen Muster ein

zentrales Konzept und eines ihrer Alleinstellungsmerkmale. Wie dieses Kapitel zeigen wird, kann das Muster-Konzept grundsätzlich auf 3D-Sprachen übertragen werden, wobei die konkreten Implementierungen natürlich neu entwickelt werden mussten.

Zwischen den konkreten visuellen Mustern und deren zugrundeliegenden *abstrakten Musterbeschreibungen* gibt es Unterschiede. Auf diese werde ich im nachfolgenden Abschnitt eingehen. Abschnitt 4.2 behandelt anschließend grundlegende Eigenschaften der konkreten visuellen Muster, u. a. auch Regeln zum Layout beim Zusammensetzen verschiedener Muster. In den danach folgenden vier Abschnitten stelle ich die wichtigsten Aspekte der in DEViL3D umgesetzten konkreten Muster vor. Abschnitt 4.7 beschreibt generische 3D-Darstellungen mit denen das visuelle Erscheinungsbild dreidimensionaler Sprachkonstrukte definiert wird. Der darauf folgende Abschnitt beschäftigt sich mit der Einbettung von 2D-Darstellungen in den 3D-Raum und den Zusammenhang zwischen visuellen Mustern für 2D- und 3D-Sprachen. Abschnitt 4.9 greift den Aspekt der Vollständigkeit auf und legt dar, dass die Menge der entwickelten visuellen Muster zur Beschreibung unterschiedlicher 3D-Sprachen vollständig ist. Das Kapitel endet mit der Vorstellung und Einordnung verwandter Arbeiten.

## 4.1 Abstrakte Musterbeschreibungen

In den bisherigen Erwähnungen visueller Muster in dieser Arbeit waren zumeist die konkreten visueller Muster gemeint, die Sprachentwickler anwenden, um die visuelle Darstellung ihrer 3D-Sprache zu spezifizieren (siehe insbesondere Abschnitt 3.2). Diese Muster sind allerdings bereits konkrete Ausprägungen einer abstrakten Musterbeschreibung. Die abstrakten Musterbeschreibungen beschreiben ein Darstellungskonzept für eine bestimmte Struktur (vgl. [Sch06, S. 48ff.]). Aus einem solchen abstrakten Darstellungskonzept lassen sich unterschiedliche konkrete visuelle Muster ableiten, die jeweils folgende Eigenschaften konkret festlegen:

- die konkrete Darstellung
- die Layouteigenschaften
- die Interaktionsmechanismen.

Abstrakte Musterbeschreibungen[1] wurden erstmals von Schmidt und Schindler [SS00] für visuelle zweidimensionale Sprachen konzipiert. Dafür wurde eine

[1] In den Arbeiten von Schmidt [SS00; Sch06] ist von „abstrakten visuellen Mustern“ statt von „abstrakten Musterbeschreibungen“ die Rede. Ich verwende hier den Begriff der Musterbeschreibungen, da damit Beschreibungen gemeint sind, aus denen sich visuelle Muster für 2D- und als auch 3D-Sprachen ableiten lassen.

Vielzahl zweidimensionaler visueller Sprachen auf Gemeinsamkeiten untersucht, um dann die gemeinsamen Eigenschaften der Repräsentation in visuellen Mustern zu kapseln. Bei dieser Untersuchung war die Anzahl der verwendeten Dimensionen kein Thema und es wurden explizit keine Überlegungen zu dreidimensionalen visuellen Sprachen angestellt. Das bedeutet aber nicht, dass die damals formulierten Musterbeschreibungen zur Beschreibung von 3D-Sprachen ungeeignet sind, sondern, dass die Beschreibungen zu sehr auf den zweidimensionalen Fall fokussiert sind. Ich habe die Musterbeschreibungen weiter generalisiert und verallgemeinert, sodass aus diesen sowohl konkrete visuelle Muster für 3D- als auch 2D-Sprachen abgeleitet werden können.

Insgesamt ergeben sich die folgenden abstrakten Musterbeschreibungen:

- Das *Mengen-Muster* visualisiert eine abstrakte Menge von gleichartigen oder ähnlichen Elementen, die innerhalb eines vorgegebenen Bereichs beliebig angeordnet werden können.

- Die *gitterbasierten Muster* visualisieren einen abstrakten Vektorraum, deren Elemente entlang der vorhandenen Raumdimensionen in gleichen Abständen zueinander aufgespannt werden.

- Durch das *Verbindungs-Muster* wird eine binäre Relation zwischen zwei Elementen durch eine Verbindungslinie zwischen ihnen visualisiert.

- Das *Graph-Muster* visualisiert einen abstrakten Graphen durch Kombination von Verbindungs- und Mengen-Muster.

- Das *Baum-Muster* visualisiert einen abstrakten gewurzelten Baum derart, dass entlang einer Dimension die Höhe der Baumknoten kodiert ist und die verbleibenden Dimensionen genutzt werden, um die Knoten mit gleicher Höhe darzustellen.

- Das *Formular-Muster* visualisiert ein abstraktes $n$-Tupel, indem die Tupelelemente in fester relativer Position zueinander angeordnet werden. Die Tupelelemente werden in eine grafische Repräsentation eingebettet, die einem Formular ein visuelles Erscheinungsbild verleihen.

Die hier formulierten abstrakten Musterbeschreibungen abstrahieren von ihrer konkreten Anwendung in zwei- oder dreidimensionalen visuellen Sprachen. So lassen sich aus vielen der obigen Beschreibungen verschiedene konkrete visuelle Muster ableiten, die sich in Abhängigkeit der zur Verfügung bestehenden Dimensionen noch unterscheiden können. In den Abschnitten 4.3 bis 4.6 werde ich auf die daraus abgeleiteten konkreten visuellen Muster eingehen.

Nicht unerwähnt lassen möchte ich, dass es mit dem *Registerkarten-* und *Stapel-Muster* (siehe [SS00, S. 33, 36]) auch zwei für 2D-Sprachen spezifische abstrakte Muster gibt, die in 3D-Sprachen keine direkte Berücksichtigung finden. Ausprägungen des Registerkarten-Musters sind aus GUI-Anwendungen bekannt und beschreiben ein abstraktes $n$-Tupel, dessen Elemente 2,5-dimensional übereinander gestapelt werden. Das Stapel-Muster visualisiert – wie das Listen-Muster – eine abstrakte Folge, allerdings mit dem Unterschied, dass immer nur das oberste Element sichtbar ist. Diese Anforderung ist in 3D-Sprachen nur noch eine Frage der Perspektive und kann durch Navigationsmechanismen (siehe Kapitel 6) realisiert werden, sodass in dem Fall das Stapel- mit dem Listen-Muster zusammenfällt.

Der letzte Absatz hat einige interessante Fragestellungen zum Verhältnis visueller Muster für 2D- und 3D-Sprachen aufgeworfen. So ergibt sich bei manchen 3D-Mustern nach Weglassen einer Dimension das korrespondierende 2D-Muster. Weiterhin können zwei Muster, die für 2D-Sprachen differenziert wurden, in 3D-Sprachen in einem Muster subsumiert werden. Außerdem ergeben sich durch Anordnung von 2D-Darstellungen im 3D-Raum ganz neue Möglichkeiten Zusammenhänge darzustellen. Alle diese Aspekte werden in Abschnitt 4.8 nach der Beschreibung der konkreten visuellen Muster behandelt.

Im Folgenden werde ich genauer auf grundlegende Eigenschaften und Konzepte von konkreten visuellen Muster eingehen und anschließend die wichtigsten Aspekte der umgesetzten Muster vorstellen. Spreche ich an nachfolgenden Stellen dieser Arbeit von visuellen Mustern, so sind damit meist die konkreten visuellen Muster gemeint.

## 4.2 Grundlegende Eigenschaften und Konzepte konkreter visueller Muster

Die Implementierungen der konkreten visuellen Muster wurden von mir als LIDO-Symbolberechnungen umgesetzt und stehen Sprachentwerfern in einer Bibliothek zur Verfügung. Jedes konkrete visuelle Muster umfasst eine Menge von Berechnungsrollen, die der Sprachentwickler Symbolen der Grammatik zuordnet. Das beim Moleküleditor angewandte Mengen-Muster (siehe Seite 55) umfasst z. B. die zwei Berechnungsrollen `VP3DSet` und `VP3DSetElement`. Ersteres wird dem Symbol zugeordnet, welches die Menge der Atome repräsentiert und letzteres den die Atome repräsentierenden Symbolen.

Die Anwendung eines Musters determiniert allerdings nicht vollständig die Darstellung der ihm zugeordneten Sprachkonstrukte. Eine Parametrisierung wird durch *Kontrollattribute* gewährleistet. Diese unterteilen sich nach Schmidt [Sch06, S. 145ff.] in *Layoutattribute* (z. B. der Abstand zwischen Listenelementen) und *Darstel-*

*lungsattribute* (z. B. das Aussehen visueller Verzierungen). In der Muster-Bibliothek sind die Kontrollattribute mit sinnvollen Startwerten definiert, die vom Sprachentwickler beliebig überschrieben werden können. Die Zuweisung von Werten an Kontrollattribute kann der Sprachentwickler auch an den Editornutzer delegieren (und so letztendlich endanwenderfreundlicher machen), indem Kontrollattribute mit Attributen der abstrakten Struktur gekoppelt werden.

Das für das DEViL-System entwickelte Konzept der Grammatik-Anpassung, welches erlaubt, die semantische Struktur auf Anforderungen der visuellen Repräsentation anzupassen, ist auch in DEViL3D in gleicher Weise anwendbar (für Details siehe [Sch06, S. 164ff.]).

In einigen 3D-Sprachen spielen auch textuelle Bestandteile eine wichtige Rolle (ein gutes Beispiel dafür sind die 3D-Klassendiagramme in Abbildung 7.1f auf Seite 136). Diese textuellen Elemente sind in speziellen Berechnungsrollen der Muster gekapselt, die an Blätter des zugrundeliegenden Strukturbaums zugewiesen werden können.

Das Prinzip der Kombination einzelner visueller Muster ist von besonderer Bedeutung, um 3D-Sprachen mit komplexem Darstellungskonzept zu realisieren. Das wird durch Kooperation der einzelnen Berechnungsrollen, die spezielle Schnittstellen anbieten oder erwarten, sichergestellt. Da dieses Verfahren vornehmlich Layoutaspekte tangiert, verweise ich diesbezüglich auf Kapitel 5. Dort beschreibe ich das Verfahren und alle weiteren in visuellen Mustern gekapselten Layouteigenschaften. Die ebenfalls an Muster gebundenen Interaktionsmechanismen werden in Kapitel 6 vorgestellt.

In den nachfolgenden Abschnitten werde ich die in DEViL3D umgesetzten visuellen Muster vorstellen. Zunächst folgt eine Übersicht über drei verschiedene Implementierungen des Mengen-Musters. Danach stelle ich in Abschnitt 4.4 konkrete Ausprägungen der gitterbasierten Muster vor. Danach folgen Abschnitte zu Relationen beschreibenden Mustern sowie zu den übrigen, unter keine dieser Kategorien fallenden, visuellen Muster.

## 4.3 Mengen-Muster

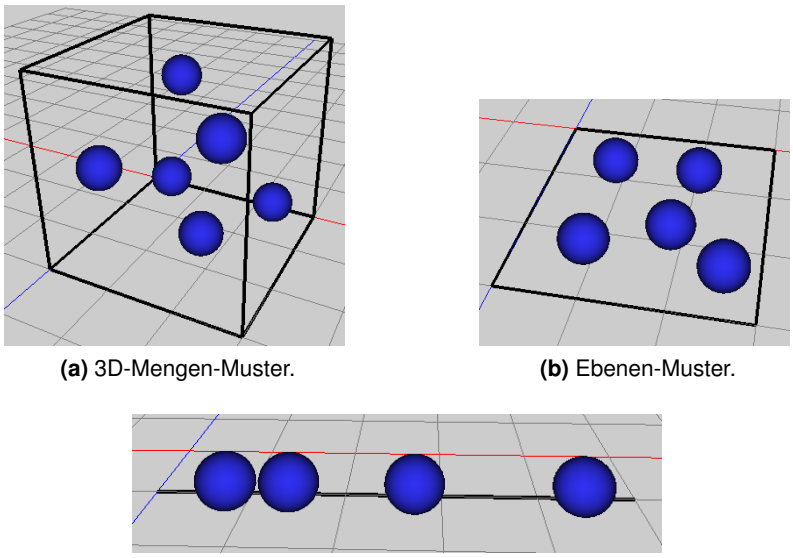
Die abstrakte Musterbeschreibung für Mengen beschreibt Elemente, die innerhalb eines vorgegebenen Bereichs vom Anwender beliebig angeordnet werden können. Ich habe drei konkrete visuelle Muster daraus abgeleitet, die Teil der Muster-Bibliothek von DEViL3D sind. Die drei konkreten Muster unterscheiden sich darin, innerhalb welchen Bereichs deren Elemente dargestellt werden.

Das wichtigste solche Muster für 3D-Sprachen ist das 3D-Mengen-Muster. Bei diesem werden die Mengenelemente innerhalb eines dreidimensionalen, quaderförmigen Bereichs angeordnet (siehe Abbildung 4.1a). Die vom Endanwender fest-

zulegende Position der Elemente wird in der editierbaren Struktur gespeichert. Interaktionsmechanismen, die das Einfügen und Verschieben innerhalb des vorgegebenen Bereichs ermöglichen, werden in Kapitel 6 vorgestellt. In den meisten Fällen soll jedes einzelne Element der Menge als solches zu erkennen sein, was impliziert, dass sich die Mengenelemente nicht gegenseitig durchdringen dürfen. Das konkrete visuelle Muster stellt dies automatisch sicher. Soll eine Durchdringung nicht ausgeschlossen werden, kann dies durch Überschreiben eines Kontrollattributs erlaubt werden.

Es wurden noch zwei weitere Implementierungen des Mengen-Musters umgesetzt, die sich hinsichtlich des Bereichs unterscheiden, in dem die Elemente platziert werden. Beim *Ebenen-Muster* werden die Elemente auf einer zweidimensionalen Ebene platziert (siehe Abbildung 4.1b). Beim *1D-Mengen-Muster* nehmen die Elemente kontinuierliche Positionen ein, die sich entlang einer Dimension unterscheiden (vgl. Abbildung 4.1c). Die Festlegung entlang welcher Dimension sich 1D-Mengenelemente aufreihen oder auf welcher Ebene sich Ebenenelemente befinden, kann durch Kontrollattribute festgelegt werden. Alle anderen Aspekte wie Nicht-Durchdringung oder darauf zugeschnittene Interaktionsmechanismen gelten analog zum 3D-Mengen-Muster.

Die drei Ausprägungen des Mengen-Musters wurden so gewählt, dass sie die im Dreidimensionalen möglichen Darstellungsvarianten gut abdecken. Sollen Elemente frei im Raum platziert werden, wird das vom 3D-Mengen-Muster unterstützt. In Sprachen, die Objekte aus der realen Welt anordnen, werden diese meist



**(a)** 3D-Mengen-Muster.

**(b)** Ebenen-Muster.

**(c)** 1D-Mengen-Muster.

**Abbildung 4.1:** Ausprägungen der drei Mengen-Muster.

auf einer Grundfläche positioniert, was durch das Ebenen-Muster realisiert wird. Weiterhin können auf solchen Ebenen aber auch zweidimensionale Diagramme dargestellt werden, die dann in den 3D-Raum eingebettet werden (siehe Abschnitt 4.8). Das 1D-Mengen-Muster reduziert die Freiheiten der Positionierung weiter und erlaubt die freie Anordnung von Elementen entlang einer Dimension.

Grundsätzlich ist das Ebenen-Muster mit dem Mengen-Muster im DEViL-System vergleichbar, welches Sprachkonstrukte auf einer 2D-Zeichenfläche darstellt. Einziger Unterschied ist, dass die Mengenelemente des Ebenen-Musters typischerweise ein dreidimensionales Volumen besitzen. Ein 1D-Mengen-Muster ist auch für zweidimensionale visuelle Sprachen vorstellbar, wurde im DEViL-System aber nicht explizit umgesetzt, sondern durch Überschreiben von Kontrollattributen des Mengen-Musters implizit realisiert. Bei 3D-Sprachen wird das 3D-Mengen-Muster häufig angewendet, wie z. B. für inhärent dreidimensionale Sprachen wie Molekülmodelle.

## 4.4 Gitterbasierte Muster

Aus der abstrakten Beschreibung für gitterbasierte Muster lassen sich mit dem Quader-, Matrix- und Listen-Muster drei konkrete visuelle Muster ableiten. Die drei Muster visualisieren einen Vektorraum, deren Elemente in gleichen Abständen zueinander aufgespannt werden. Dabei ist mit gleichem Abstand der Abstand zwischen dem Ende des ersten und dem Anfang des nächsten Elements gemeint. Das Quader-Muster visualisiert eine dreidimensionale Matrix, das Matrix-Muster mit einer zweidimensionalen Matrix einen Spezialfall davon und das Listen-Muster visualisiert eine eindimensionale Folge von Elementen. Die Sprachkonstrukte werden nach einem festen Schema auf einem Gitter angeordnet.

Die Elemente der einzelnen Muster werden immer parallel zu einer der drei Raumachsen aufgereiht. Im Gegensatz zu den frei positionierbaren Mengenelementen wird bei den gitterbasierten Mustern durch die Gleichmäßigkeit der Anordnung implizit eine feste Ordnung zwischen ihren Elementen definiert.

### 4.4.1 Quader-Muster

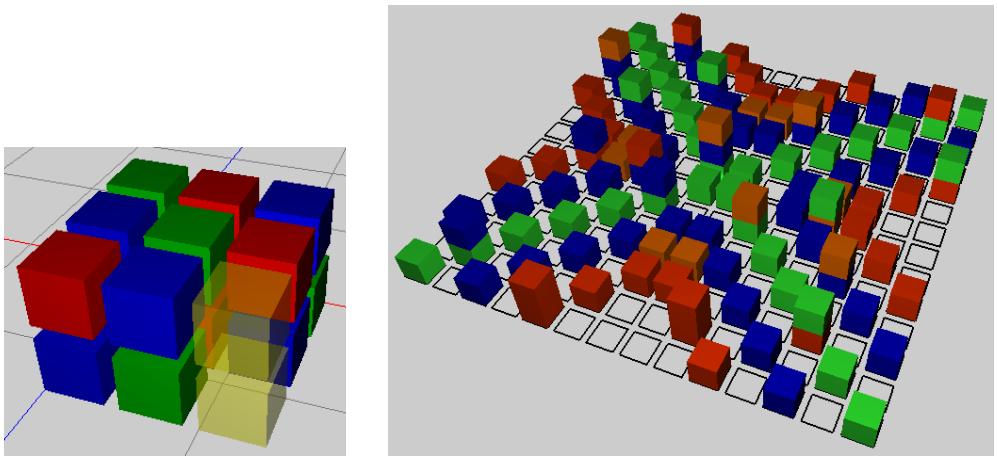
Das Quader-Muster ordnet Elemente innerhalb eines quaderförmigen Bereichs an, sodass diese durch Angabe der Ebene, Zeile und Spalte eindeutig identifizierbar sind. Zwischen den Ebenen, Zeilen und Spalten gibt es feste Abstände, sodass sich ein gitterbasiertes Schema ergibt. Für jede Position in dem Schema gibt es initial einen Container, der als Platzhalter für einzufügende Elemente fungiert. Ist ein Element größer als sein Platzhalter, werden automatisch auch die Container entsprechend vergrößert, die sich auf derselben Ebene, Zeile oder Spalte befinden.

Dadurch wird die gitterbasierte Anordnung zwar ungleichmäßig, aber die festen Abstände zwischen den Containern bzw. Elementen bleiben so erhalten.

Die durch das Quader-Muster dargestellte dreidimensionale Matrix kann durch Stapeln mehrerer zweidimensionaler Matrizen ausgedrückt werden. Dies wurde bei Umsetzung des Quader-Musters durch Kombination des Matrix- und Listen-Muster (siehe die folgenden zwei Abschnitte) realisiert. Über Kontrollattribute kann der feste Abstand zwischen den Ebenen, Zeilen und Spalten und die initiale Größe der Container festgelegt werden. In Abbildung 4.2a ist eine Anwendung des Matrix-Musters dargestellt. Die vorderen rechten Zellen sind leer, sodass dort transparent-gelbe Quader als Platzhalter angezeigt werden.

### 4.4.2 Matrix-Muster

Das Matrix-Muster ist konzeptionell ein Spezialfall des Quader-Musters bei dem dieses nur eine Ebene besitzt und somit eine zweidimensionale Matrix visualisiert. Wie auch beim Quader-Muster, ist jede Stelle innerhalb der Matrix gleichberechtigt und kann die gleiche Art Information aufnehmen. Die vom Matrix-Muster definierte Struktur kann als Folge von Folgen aufgefasst werden. Dies wird bei der Implementierung des Musters berücksichtigt, indem die Matrix aus einer Menge von Zeilen und Spalten besteht. Die Zellen der Matrix werden in der abstrakten Struktur als Liste in einer Matrix-Zeile gespeichert und besitzen jeweils eine Referenz auf die korrespondierende Spalte. Dem konkreten Muster liegt eine komplexe Struktur-Synchronisation zugrunde, um beim Einfügen oder Löschen einer Zeile oder Spalte automatisch die Matrix-Zellen konsistent zu halten.



**(a)** Anwendung des Quader-Musters. **(b)** Anwendung des Matrix-Musters im Musik-Editor.

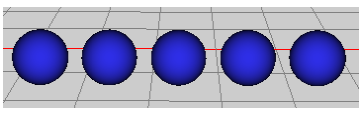
**Abbildung 4.2:** Ausprägungen des Quader- und Matrix-Musters.

Durch Kontrollattribute kann der feste Abstand zwischen den Zeilen und Spalten, die Größe der Matrix-Zellen, die initial durch Container repräsentiert werden, sowie die Beschaffenheit der grafischen Verzierungen festgelegt werden. Abbildung 4.2b zeigt eine Anwendung des Matrix-Musters im Musik-Editor. Dort enthält eine Zelle nicht nur einzelne Elemente, sondern eine Liste von Sprachkonstrukten, die entlang der $y$-Achse gestapelt werden.

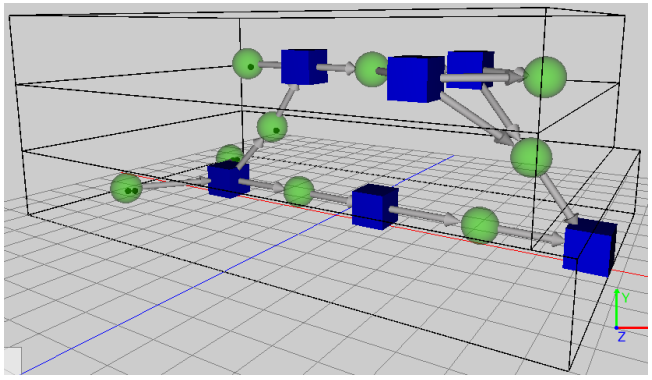
### 4.4.3 Listen-Muster

Das Listen-Muster visualisiert eine endliche Folge von Elementen, die entlang einer der drei Raumdimensionen aufgereiht wird. Die Ausbreitungsrichtung kann von Sprachentwerfern über ein Kontrollattribut festgelegt werden. Auch der Abstand zwischen den Listenelementen ist durch ein Kontrollattribut zu bestimmen.

Abbildung 4.3a zeigt die einfachste Listendarstellung gleichartiger Elemente. Eine komplexere Anwendung des Listen-Musters bei 3D-Petri-Netzen ist in Abbildung 4.3b dargestellt (vgl. Abschnitte 2.2.7 und 7.2.1). Jedes Listenelement besteht dabei aus einer Ebene, auf der Stellen und Transitionen platziert werden können. Die Listenelemente können also aus beliebig komplexen Strukturen bestehen. Dies ist aber keine spezielle Eigenschaft des Listen-Musters, sondern basiert auf der allen Mustern zugrundeliegenden Eigenschaft, diese miteinander zu kombinieren (siehe Abschnitt 5.1).



**(a)** Einfache Liste gleichartiger Elemente.




**(b)** Liste komplexer Strukturen.

**Abbildung 4.3:** Zwei verschiedene Ausprägungen des Listen-Musters.

## 4.5 Relationen beschreibende Muster

In diesem Abschnitt werde ich konkrete Ausprägungen des Verbindungs-, Graph- und Kegelbaum-Musters vorstellen, die primär Relationen zwischen Sprachkonstrukten visualisieren.

### 4.5.1 Verbindungs-Muster

Das Verbindungs-Muster visualisiert binäre Relationen zwischen zwei Sprachkonstrukten durch direkt zwischen diesen verlaufenden Linienverbindungen. Im Strukturbaum können sich die miteinander verbundenen Sprachkonstrukte an beliebigen Stellen befinden, sodass das Verbindungs-Muster meist *Querrelationen* zwischen verschiedenen Knoten im Strukturbaum repräsentiert.

Sprachentwickler haben durch verschiedene Kontrollattribute eine genaue Kontrolle über die Beschaffenheit der Verbindung. Es kann der Durchmesser der Linienverbindung sowie ihre Farbe oder Textur definiert werden. Weiterhin kann festgelegt werden, ob die Verbindung Pfeilspitzen umfassen soll, die in Größe und Form variieren können. Abbildung 4.4 zeigt die Bandbreite verschiedener Verbindungen im 3D-Klassendiagrammeditor.

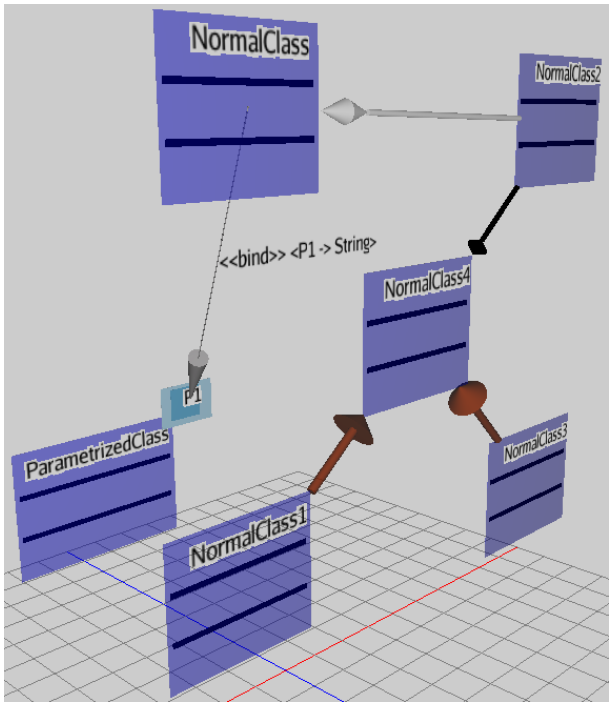



**Abbildung 4.4:** Verschiedene Verbindungen im 3D-Klassendiagrammeditor.

### 4.5.2 Graph-Muster

Basierend auf dem Verbindungs- und Mengen-Muster visualisiert das Graph-Muster einen Graphen $G = (V, E)$, dessen Knoten durch das Mengen- und dessen Kanten vom Verbindungs-Muster dargestellt werden. Charakteristisch für das Graph-Muster ist, dass spezielle Graph-Layoutalgorithmen zur Anordnung der Knoten verwendet werden können.

Dafür bieten sich *Kraft-gesteuerte Graphzeichnungen*[2] an, die auch gute Ergebnisse für im Dreidimensionalen dargestellte Graphen liefern. Die grundlegende Anforderung dieser Verfahren ist, dass sich die Knoten gleichmäßig über den Darstellungsbereich ausbreiten sollen, wobei Knoten, die miteinander verbunden sind, möglichst dicht nebeneinander platziert werden. Eades [Ead84] hat dafür ein Verfahren entwickelt, welches eine Analogie aus der Physik verwendet. Die Knoten des Graphen werden als kleine Kugeln mit elektrischer Ladung betrachtet, die sich gegenseitig abstoßen. Die Kanten fungieren als Federn, die mit Kugeln verbunden sind. Dadurch ergibt sich ein System, bei dem die Energie minimiert wird.

In DEViL3D ist beim Graph-Muster ein auf diesem Prinzip basierender einfacher Algorithmus umgesetzt. Es ließen sich aber zahlreiche weitere Algorithmen umsetzen, deren Auswahl durch Kontrollattribute festgelegt werden kann.

### 4.5.3 Kegelbaum-Muster

Das Kegelbaum-Muster ist eine konkrete Ausprägung der abstrakten Musterbeschreibung des Baum-Musters. Das Kegelbaum-Muster setzt das aus 3D-Visualisierungen bekannte Darstellungskonzept für hierarchische Strukturen von Robertson et al. [RMC91] um (siehe auch Abschnitt 2.5). Der Baum expandiert entlang der $y$-Achse und die Kinderknoten werden auf der $xz$-Ebene dargestellt. Die Kinderknoten werden auf einem Kreis angeordnet, der zusammen mit dem Vaterknoten als Spitze einen Kegel bildet (Abbildung 4.5 zeigt eine Instanz des Kegelbaum-Musters).

Anders als beim Verbindungs- und Graph-Muster visualisieren die Kanten zwischen Knoten im Kegelbaum keine Querrelationen im Strukturbaum. Die durch einen Kegelbaum dargestellte hierarchische Struktur liegt auch der Struktur der Sprache zugrunde.

Das Kegelbaum-Muster ist eine sehr spezielle Ausprägung der abstrakten Musterbeschreibung des Baum-Musters, da die Anordnung der Kinderknoten auf einem Kreis gefordert wird. Es sind auch weitere konkrete Ausprägungen des Baum-Musters vorstellbar, die die Kinderknoten z. B. an beliebigen Positionen auf einer Ebene anordnen.

[2] engl.: *force-directed graph drawing*

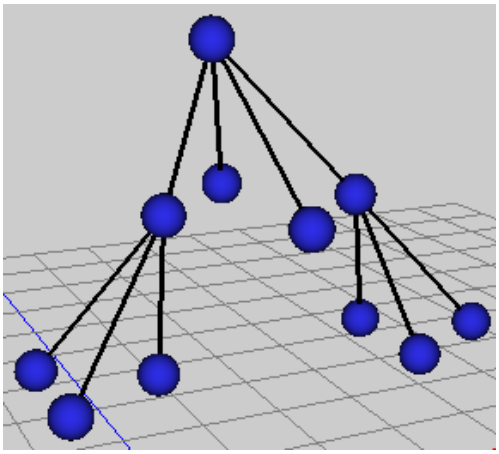


**Abbildung 4.5:** Instanz des Kegelbaum-Musters.

## 4.6 Weitere visuelle Muster

Zwei visuelle Muster lassen sich nicht direkt in das den letzten drei Abschnitten zugrundeliegende Klassifikationsschema einordnen, sodass ich diese hier beschreiben werde.

### 4.6.1 Container-Muster

Das Container-Muster ist das einfachste der implementierten Muster und stellt lediglich einen Platzhalter für genau ein Element dar. Es umfasst, wie z. B. die Mengen- oder das Listen-Muster, zwei Berechnungsrollen: `VPContainer` und `VPContainerElement`. Dem Knoten des Strukturbaums, dem die Rolle `VPContainer` zugeordnet wird, darf höchstens ein `VPContainerElement`-Knoten untergeordnet sein. Das Container-Muster wird häufig zusammen mit dem Matrix- oder Quader-Muster angewendet und stellt die in den Zellen befindlichen Sprachkonstrukte dar. Solange kein Containerelement vorhanden ist, wird ein transparent-gelber Quader als Platzhalter angezeigt (vgl. Abbildung 4.2a).

### 4.6.2 Formular-Muster

Das Formular-Muster beschreibt ein $n$-Tupel, dessen Elemente eine feste relative dreidimensionale Position zueinander haben. Außerdem wird durch das Formular-Muster die visuelle Repräsentation eines Sprachkonstrukts definiert. Die Stellen in Petri-Netzen werden z. B. als transparente grüne Kugeln dargestellt, die ein 1-Tupel, nämlich die Menge der im inneren befindlichen Token, beschreibt. Die Klassen-Darstellungen im 3D-Klassendiagrammeditor (siehe Abschnitt 7.2.1) sind ein 3-Tupel (Platzhalter für den Klassennamen im oberen Bereich, die Attribute

in der Mitte und Operationen im unteren Bereich) und werden als blaue Box mit geringer Tiefe durch das Formular-Muster dargestellt. Um das visuelle Aussehen und die Position der Tupel zu spezifizieren, werden sogenannte *generische 3D-Darstellungen* verwendet, die vom Formular-Muster durch das Kontrollattribut `depiction` referenziert werden. Die Unterelemente werden vom Kontrollattribut `formElementName` referenziert. Der nachfolgende Abschnitt stellt die generischen 3D-Darstellungen genauer vor.

## 4.7 Generische 3D-Darstellungen

Generische 3D-Darstellungen [Wol13a; Wol13b] sind ein abstraktes Konzept und beschreiben das visuelle Erscheinungsbild von Sprachkonstrukten als generische dreidimensionale Grafiken. Dieses Konzept wurde ursprünglich für den zweidimensionalen Fall im Rahmen des Vorgängersystems DEViL (siehe Abschnitt 2.6.1) entwickelt [Sch06, S. 172ff.], [Sch03a; Sch03b] und für diese Arbeit auf die dritte Dimension erweitert.

Generische 3D-Darstellungen sind formal ein 4-Tupel aus grafischen Primitiven, Darstellungseigenschaften, Containern und Dehnungsintervallen: $\mathcal{D} = (\mathcal{P}, \mathcal{R}, \mathcal{C}, \mathcal{I})$[3]. Diese Bestandteile spielen für die generischen Darstellungen folgende Rollen:

- Das Erscheinungsbild eines Sprachkonstrukts wird durch eine Menge von *grafischen Primitiven* $\mathcal{P}$ bestimmt. $\mathcal{P}$ ist die Vereinigung der Mengen folgender Primitive: $\mathcal{P}$ = Quader ∪ Kugel ∪ Kegel ∪ Zylinder ∪ Pfeil ∪ Linie ∪ Viereck ∪ Torus ∪ 3D-Modell ∪ Text. Durch 3D-Modelle lassen sich Objekte mit komplexerer Gestalt in die 3D-Darstellung integrieren.

- Die *Darstellungseigenschaften* $\mathcal{R}$ beschreiben Materialien wie Farben oder Texturen, die der Oberfläche von grafischen Primitiven $\mathcal{P}$ zugewiesen werden.

- Eine Menge von *Containern* $\mathcal{C}$ sind jeweils Platzhalter für weitere Darstellungsobjekte. Bei der Instanziierung der generischen 3D-Darstellung werden so weitere Sprachkonstrukte beliebiger Größe eingebettet. Dabei muss jeder Container einen eindeutigen Namen besitzen.

- Durch *Dehnungsintervalle* $\mathcal{I}$ wird das Layoutverhalten festgelegt, welches bestimmt, welcher Teil einer instanziierten 3D-Darstellung vergrößert wird, wenn die Größe von eingebetteten Objekten die aktuelle Container-Größe übersteigt.

[3] Die Abkürzungen beziehen sich auf die englischen Begriffe. $\mathcal{D}$: generic **D**epiction, $\mathcal{P}$: graphical **P**rimitives, $\mathcal{R}$: **R**epresentation properties, $\mathcal{I}$: stretch **I**ntervals.

Der beste Weg, visuelle Konstrukte zu spezifizieren, ist die Benutzung eines visuellen Editors. Ein solcher Ansatz stellt gemäß dem *Cognitive Dimensions Framework* [GP96] eine geringe Abbildungsnähe zwischen der Problemwelt und der Programmwelt sicher. Zu diesem Zweck wurde mit DEViL3D ein Editor spezifiziert mit dem Sprachentwerfer generische 3D-Darstellungen für die Konstrukte ihrer Sprache entwerfen können (siehe Abbildung 4.6). Der Editor selbst ist Teil des DEViL3D-Systems. In diesem stehen Interaktions- und Navigationstechniken zur Verfügung, die es erlauben im 3D-Raum zu konstruieren und auf die in Kapitel 6 genauer eingegangen wird.

Das Beispiel in Abbildung 4.6 zeigt eine 3D-Darstellung bestehend aus zwei grafischen Primitiven in Form eines Quaders und einer Kugel, aus den zwei Containern `c1` und `c2` sowie vier Dehnungsintervallen. Innerhalb der 3D-Darstellung haben die beiden Primitive eine feste relative Anordnung. Die Container befinden sich jeweils innerhalb der beiden Primitive. Die Dehnungsintervalle werden auf den hellroten Koordinatenachsen platziert, sind dadurch jeweils für eine Dimen-

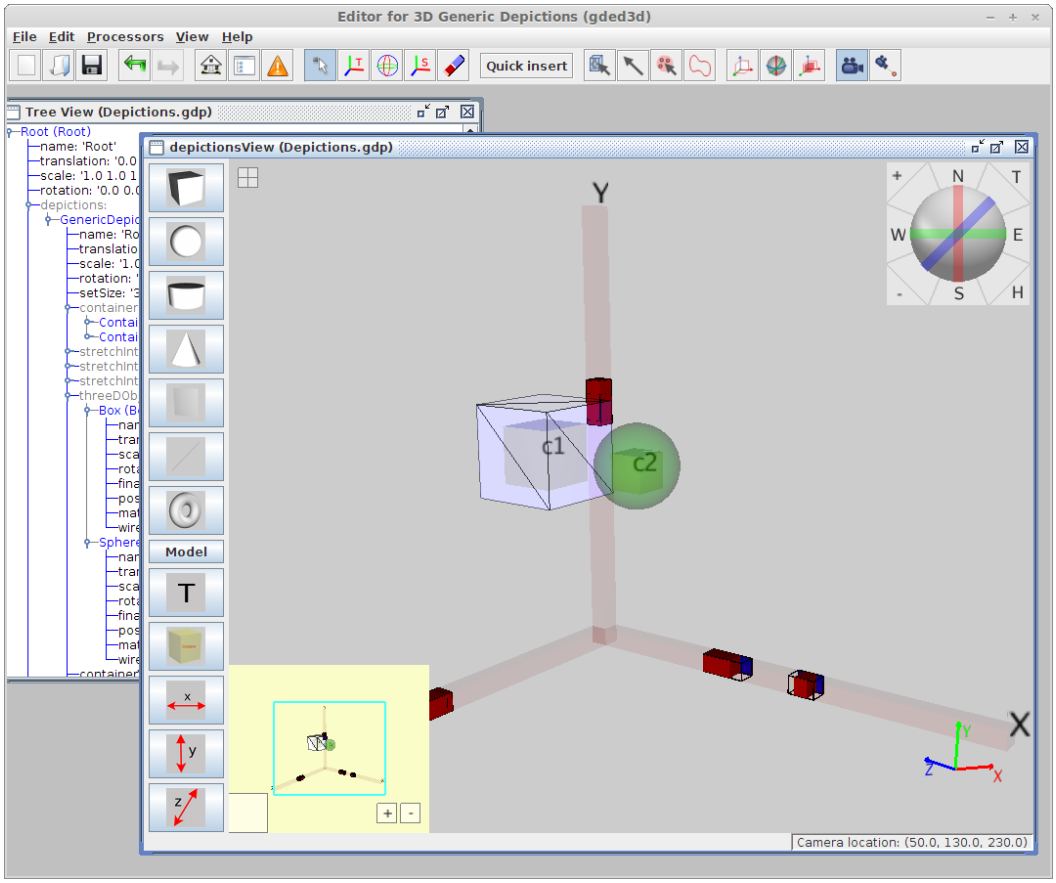



**Abbildung 4.6:** Bildschirmfoto des 3D-Editors für generische 3D-Darstellungen.

sion zuständig und beschreiben, welche Bereiche der instanziierten Darstellung bei wachsendem Platzbedarf vergrößert werden. Ein Beispiel für dieses Verhalten ist in Abbildung 5.3 auf Seite 90 dargestellt. Diesem Dehnungsverhalten liegt ein Algorithmus zugrunde, auf den genauer in Abschnitt 5.3.1 eingegangen wird.

Bei der Spezifikation des Editors wurde das 3D-Mengen-Muster verwendet, um die grafischen Primitive und Container darzustellen. Das 1D-Mengen-Muster kam für die Dehnungsintervalle zur Anwendung.

Die visuelle Gestalt der 3D-Darstellungen wird von den grafischen Primitiven $\mathcal{P}$ und den Darstellungseigenschaften $\mathcal{R}$ bestimmt. Durch diese beiden Bestandteile generischer 3D-Darstellungen werden die retinalen visuellen Variablen beschrieben, die in Abschnitt 2.5 vorgestellt wurden. Die Form, Ausrichtung und Größe wird durch die grafischen Primitive bestimmt, wobei die Größe dynamisch ist, sodass sie bei instanziierten 3D-Darstellungen noch variieren kann. Die Farbe, Helligkeit und Textur werden durch die Darstellungseigenschaften festgelegt. Diese umfassen entweder eine Farbe oder eine Textur, deren Helligkeit individuell festgelegt werden kann. So definierte Darstellungseigenschaften werden jedem grafischen Primitiv zugewiesen und bestimmen somit sein Erscheinungsbild.

Bei der Konstruktion der 3D-Darstellungen müssen folgende Bedingungen erfüllt sein: Jeder Container muss in jeder Dimension von mindestens einem Dehnungsintervall überdeckt werden. Andernfalls ist bei wachsendem Containerinhalt nicht erkenntlich, wie die 3D-Darstellung gedehnt werden soll. Außerdem dürfen sich Dehnungsintervalle nicht überlappen, da sonst nicht eindeutig zu bestimmen ist, welches der Intervalle für die Dehnung der Darstellung zuständig ist. Weiterhin muss ein Container einen eindeutigen Namen besitzen, da sie anhand ihres Namens in der Sichtspezifikation referenziert werden. Der 3D-Editor für generische 3D-Darstellungen weist Sprachentwickler während des Konstruktionsprozesses auf eventuelle Verletzungen dieser Anforderungen hin.

Die generischen 3D-Darstellungen werden im Spezifikationsprozess für die visuelle Darstellung der Sprache von der Berechnungsrolle `VP3DForm` des Formular-Musters referenziert. Das Sprachkonstrukt, welches die 3D-Darstellung referenziert, bekommt das durch sie definierte Aussehen. Unterelemente des Sprachkonstrukts erben von der `VP3DFormElement`-Rolle und werden anstelle des Containers in die Darstellung eingebettet, indem sie den Namen des Containers referenzieren. Die Container können als Schnittstelle von 3D-Darstellungen aufgefasst werden, da diese transparent ausgetauscht werden können, wenn sie in Anzahl und Benennung ihrer Container übereinstimmen.

## 4.8 Einbettung von 2D-Darstellungen in den 3D-Raum

In Abschnitt 4.1 wurde bei der Beschreibung der abstrakten visuellen Muster bereits erwähnt, dass aus diesen sowohl konkrete visuelle Muster für 3D- als auch 2D-Sprachen abgeleitet werden können. In diesem Abschnitt werde ich diese Zusammenhänge zwischen Mustern für 2D- und 3D-Sprachen näher beleuchten.

Die drei in Abschnitt 4.3 vorgestellten Mengen-Muster stammen alle aus derselben Abstraktion und unterscheiden sich nur dahingehend, in welchen der drei Dimensionen die Position eines eingefügten Elements wählbar ist. Für Mengenelemente des 3D-Mengen-Musters können Editoranwender die $x$-, $y$- und $z$-Position festlegen. Beim Übergang zum Ebenen-Muster ist die Position nur noch für zwei Dimensionen festzulegen und die dritte Dimension, die die Lage der Ebene beschreibt, ist bereits festgelegt. Ein ähnlicher Effekt liegt beim Übergang zum 1D-Mengen-Muster vor.

Das zweidimensionale Mengen-Muster aus dem DEViL-System ist aus dem gleichen abstrakten Muster abgeleitet, wie die in Abschnitt 4.3 vorgestellten Mengen-Muster für 3D-Sprachen. Das 2D-Mengen-Muster nutzt die zwei verfügbaren Dimensionen zur Platzierung von Sprachkonstrukten, wohingegen das Ebenen-Muster von vornherein von drei möglichen Dimensionen nur zwei nutzt. Damit

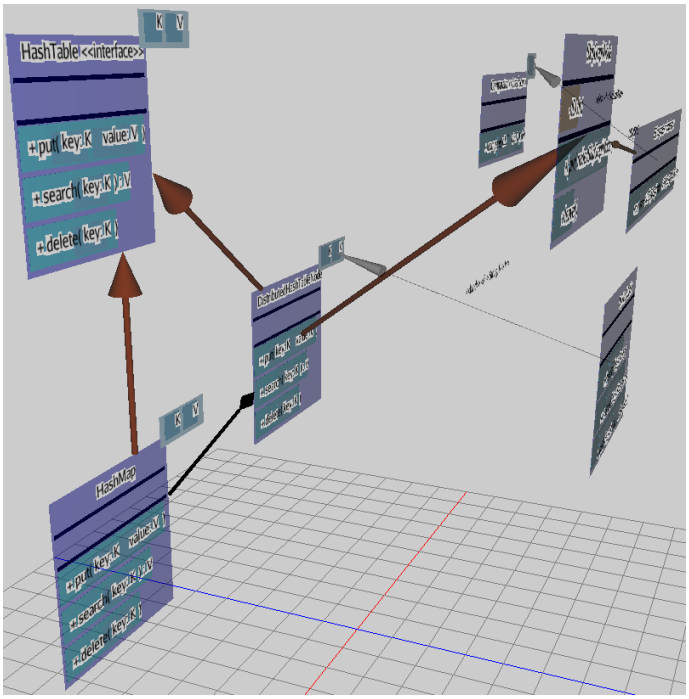



**Abbildung 4.7:** Einbettung von 2D-Klassendiagrammen in den 3D-Raum.

**Abbildung 4.8:** Zwei verschiedene Ansätze zur Einbettung einer 2D-Grid-Darstellung in den 3D-Raum; Quelle: [SAH14].

lassen sich mit diesen beiden konkreten Mustern vergleichbare Informationen darstellen.[4] Dadurch ergibt sich die Möglichkeit, Fragmente aus 2D-Sprachen auf Ebenen in den 3D-Raum einzubetten. Dieser Ansatz wird bei der 3D-Sprache für UML-Klassendiagramme [Sto14] genutzt (siehe auch Abschnitt 7.2.1). Dabei werden klassische 2D-Klassendarstellungen auf Ebenen im 3D-Raum platziert. Die Zuordnung der Klassen zu den Ebenen hat eine bestimmte Bedeutung, wodurch die Einbettung semantische Aussagekraft hat. Der spezifizierte Editor für 3D-Klassendiagramme hat zum Ziel, parametrisierte Klassen im 3D-Raum besser anzuordnen. Dazu umfasst jede Ebene die Menge der Klassen mit der gleichen Anzahl an Parametern (siehe Abbildung 4.7). Dieses Prinzip lässt sich auch zur Gruppierung anderer Aspekte – wie z. B. für Stereotype oder Paketzugehörigkeiten – erweitern.

Solche Möglichkeiten der Einbettung existieren auch für weitere Muster, wie z. B. das Listen- oder Matrix-Muster, die direkte Entsprechungen in DEViL und DEViL3D haben. Durch die Darstellung von Diagrammen einer mit DEViL spezifizierten 2D-Sprache im 3D-Raum, können z. B. semantische Sachverhalte zwischen den Diagrammen in den Vordergrund gerückt werden. Die Sprachspezifikation für die visuelle Darstellung muss aufgrund der Ähnlichkeit vieler Muster nur geringfügig angepasst werden. Das gilt natürlich nur bei unveränderter abstrakter Syntax der einzubettenden Sprachfragmente.

Weitere Ideen für Einbettungen von 2D-Darstellungen in den 3D-Raum sind bei Schoeffmann et al. [SAH14] zu finden. Sie haben eine Sammlung von Fotos, die typischerweise auf einem 2D-Grid angeordnet werden, in den 3D-Raum eingebettet, indem die Fotos auf einem 3D-Ring oder 3D-Kugel dargestellt werden (siehe Abbildung 4.8). Eventuell lassen sich solche Ideen auch auf 3D-Sprachen übertragen, sodass hauptsächlich zweidimensionale Darstellungen auf eine Kugel projiziert

[4] Natürlich haben die Sprachkonstrukte, die gemäß des Ebenen-Musters angeordnet werden, meist ein dreidimensionales Volumen, was beim 2D-Mengen-Muster nicht der Fall ist. Davon abstrahiere ich an dieser Stelle.

werden und diese besonders wichtige Teile des Diagramms in den Vordergrund rückt.

## 4.9 Beschreibungsvollständigkeit

In diesem Abschnitt möchte ich zeigen, dass die Menge der in den vorherigen Kapiteln vorgestellten visuellen Muster ausreicht, um die in Abschnitt 2.2 vorgestellten 3D-Sprachen syntaktisch zu beschreiben. Dabei handelt es sich teilweise um hypothetische Überlegungen, welche Muster angewendet werden würden, wenn eine bestimmte Sprache mit DEViL3D spezifiziert würde. Ein Großteil der anschließend diesbezüglich betrachteten 3D-Sprachen (sechs von zehn) wurde tatsächlich mit DEViL3D spezifiziert. Für eine genauere Beschreibung sowie Bildschirmfotos der generierten Strukturediteren verweise ich auf Abschnitt 7.2.1.

Tabelle 4.1 listet zehn 3D-Sprachen auf und gibt an, welche visuellen Muster für ihre Spezifikation mit DEViL3D verwendet werden müssen. Bei den ersten sechs Sprachen der Tabelle handelt es sich um die tatsächlich spezifizierten Sprachen. Die Anwendung visueller Muster für die letzten vier Sprachen stellt eine teilweise hypothetische Betrachtung dar. Zum einen kann sich bei der realen Spezifikation ein anderes Muster als bessere Alternative erweisen. Zum anderen – und dies ist viel entscheidender – bieten die Papiere, in denen die 3D-Sprachen vorgestellt wurden, nur einen sehr groben Überblick über die genaue Sprachsyntax. Die Entscheidung, welches visuelle Muster ich bei der Spezifikation nutzen würde, musste

| | Formular | 3D-Menge | Ebene | 1D-Menge | Liste | Container | Matrix | Quader | Verbindung | Kegelbaum | Graph |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Molekülmodelle | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |
| ToneCraft | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ |
| 3D-Petri-Netze | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |
| Gen. 3D-Darstellungen | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| 3D-Klassendiagramme | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |
| 3D-Sequenzdiagramme | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |
| Cube | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ |
| SAM | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| 3D-PP | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ |
| Lingua Graphica | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ |

**Tabelle 4.1:** Anwendung visueller Muster zur Spezifikation visueller 3D-Sprachen.

ich auf Grundlage grober Informationen treffen, die sich hauptsächlich auf die in den Papieren gezeigten Abbildungen der visuellen Sprachinstanzen stützen.

Bei der Spezifikation aller Sprachen kommt das Formular-Muster zur Anwendung, da dieses generische 3D-Darstellungen referenziert, ohne dass die Sprachkonstrukte keine visuelle Repräsentation hätten. Sehr häufig werden weiterhin das Verbindungs-, 3D-Mengen- und Listen-Muster angewendet. Einige Muster werden zur Spezifikation der zehn Sprachen nur sehr selten verwendet. So wird das 1D-Mengen-Muster nur bei der Sprache für generische 3D-Darstellungen zur Anordnung der Dehnungsintervalle verwendet. Das Matrix-Muster wird nur in der Sprache ToneCraft verwendet, um Musikinstrumente zu platzieren.

Besonders einfach ließen sich die Sprachen 3D-Petri-Netze und ToneCraft spezifizieren. Die visuelle Darstellung der 3D-Petri-Netze erfolgte unter Verwendung des Listen-, Ebenen-, 3D-Mengen- und Verbindungs-Musters. In der ToneCraft-Spezifikation wurde das Matrix-Muster in Verbindung mit dem Listen-Muster zum Stapeln der Instrumente verwendet; alternativ hätte allerdings auch das Quader-Muster verwendet werden können.

Aus Sicht der Muster-Anwendung ist auch die Sprache der Molekülmodelle sehr einfach zu spezifizieren: Neben der obligatorischen Anwendung des Formular-Musters wurde das 3D-Mengen-Muster zur Anordnung der Atome und das Verbindungs-Muster für die Bindungen zwischen den Atomen angewendet. Dadurch können aber noch nicht die, sich aus der Molekülstruktur ergebenden, speziellen Anforderungen zur Anordnung der Atome realisiert werden. Dafür müssen zusätzliche Layoutspezifikationen angegeben werden, auf die ich im nachfolgenden Kapitel eingehe. Ähnlich verhält es sich bei der Spezifikation der SAM-Sprache, bei der das Andocken von Nachrichten und Ports eine wichtige Rolle spielt (siehe Abschnitt 2.2.2). Zusätzlich zur Anwendung des 3D-Mengen-Musters muss auch hier die Layoutanforderung zum Andocken realisiert werden.

Die speziellen Layoutanforderungen dieser beiden Sprachen sind eher die Ausnahme, sodass zur visuellen Darstellung aller anderen Sprachen, die von DEViL3D bereitgestellten visuellen Muster ausreichen. Dies gilt auch für die 3D-Klassen- und 3D-Sequenzdiagramme, die spezielle semantische Sprachanforderungen besitzen, die durch Synchronisations-Funktionen sichergestellt werden; für die visuelle Darstellung sind die visuellen Muster hingegen ausreichend.

Es lässt sich zusammenfassen, dass die Bibliothek der visuellen Muster die Spezifikation aller zehn 3D-Sprachen unterstützt. Die zusätzlichen Layoutanforderungen sind zu speziell, als dass sie in einem zusätzlichen Muster implementiert werden sollten. Denn eine derartige Spezialisierung würde dazu führen, dass ein solch stark spezialisiertes Muster nur in genau einer Sprache zur Anwendung käme, was dem Muster-Konzept zuwider laufen würde.

## 4.10 Verwandte Arbeiten

In diesem Abschnitt werde ich verwandte Arbeiten zu den in diesem Kapitel präsentierten Themen vorstellen. Dies umfasst im Wesentlichen die visuellen Muster und die generischen 3D-Darstellungen. Da das Konzept der visuellen Muster in dieser Arbeit erstmals für dreidimensionale Sprachen weiterentwickelt wurde, gibt es keine verwandten Arbeiten, die sich auf 3D-Sprachen beziehen. Aus diesem Grund eruiert der nachfolgende Abschnitt Aspekte visueller Muster für zweidimensionale visuelle Sprachen.

### Visuelle Muster

Die hier vorgestellten visuellen Muster für 3D-Sprachen beruhen auf visuellen Mustern für 2D-Sprachen [SK03], die Sprachspezifizierern im VL-Eli- und DEViL-System als Bibliothek zur Verfügung stehen und initial von Schmidt und Schindler [SS00] konzipiert wurden. Die Anwendung der visuellen Muster erfolgt auch dort innerhalb der Spezifikation attributierter Grammatiken (siehe Abschnitt 3.2), mit deren Hilfe die visuelle Darstellung der Sprache berechnet wird.

Einen auf konzeptioneller Ebene in gewisser Weise vergleichbaren Ansatz stellt Tveit [Tve08] mit sogenannten *Diagramm-Mustern* vor. Sie hat neun Muster identifiziert, die sich in zwei Gruppen unterteilen lassen: Muster, die ein Diagramm-Token beschreiben und Muster, die Beziehungen zwischen Tokens beschreiben. Die erste Kategorie umfasst z. B. die *Compartment Pattern*, *Connection Pattern* und drei verschiedene Muster zur Beschreibung von Texten innerhalb eines Diagramms. Ersteres beschreibt eine Menge von Abteilungen innerhalb eines Diagramms. Dies würde in DEViL3D durch Container innerhalb einer vom Formular-Muster referenzierten generischen 3D-Darstellung realisiert werden. Das *Connection Pattern* realisiert die gleiche Funktionalität wie das Verbindungs-Muster in DEViL3D. Die Menge der Muster, die Beziehungen beschreiben, umfasst z. B. das *Inside Pattern*, welches ausdrückt, dass sich ein grafisches Objekt innerhalb eines anderen befindet. Solche Aspekte werden in DEViL3D auch mithilfe von Containern gelöst. Abschnitt 5.3 beschreibt dies ausführlich. Da Tveit die Muster nur theoretisch beschreibt, ohne diese implementiert und z. B. in einem Generatorsystem verfügbar gemacht zu haben, bleibt ihr Nutzen fraglich, insbesondere, da ähnliche Ansätze bereits u. a. im DEViL-System implementiert wurden.

Einen auf Meta-Modellen basierenden Ansatz zur Beschreibung, Klassifikation und Implementierung visueller Sprachen stellen Bottoni und Grau vor [BG04]. Auch dort gibt es wieder ein *Connection-based* Meta-Modell, welches für graphbasierte Sprachen essentiell ist und beschreibt, dass zwei Sprachkonstrukte miteinander verbunden werden. Auch Meta-Modelle für *Containment-based* Sprachen

werden vorgestellt, bei denen Sprachkonstrukte rekursiv ineinander geschachtelt sind.

Einen weiteren ähnlichen Ansatz verfolgen Maier und Minas mit sogenannten *Layoutpattern* [MM12a]. Da bei diesem der Aspekt des Sprachlayouts im Vordergrund steht, werde ich die Layoutpattern im nachfolgenden Layout-Kapitel in Abschnitt 5.7 genauer vorstellen und mit dem in DEViL3D verwendeten Ansatz in Beziehung setzen.

#### Generische 3D-Darstellungen

Ein zum Konzept der generischen 3D-Darstellungen sehr ähnlicher Ansatz wurde bereits von Chung et al. [CHM99] verfolgt, die ein Werkzeug namens *3DComposer* entwickelt haben, mit dem sogenannte *3Dvixels* konstruiert werden können, die als Bausteine für verschiedene 3D-Anwendungen (3D-Sprachen und 3D-Visualisierungen eingeschlossen) fungieren. Die Verwendung der 3Dvixels in verschiedenen Anwendungen wird durch die Generierung von wiederverwendbaren Softwarekomponenten in Form von *JavaBeans* ermöglicht. Die Konstruktion von beispielhaften 3D-Diagrammen erfolgt direkt durch den Endanwender in 3DComposer, also dem gleichen Werkzeug mit dem auch die 3Dvixel selber konstruiert werden. Es handelt sich somit nicht um ein Generatorsystem, welches zwischen Sprachentwickler und Sprachanwender unterscheidet. Daher benötigt 3DComposer keine Konzepte wie Container und Dehnungsintervalle.

Mit dem Editor für aufpumpbare Zeichnungen des AgentCubes-Systems [IRW09] (siehe Abschnitt 2.2.8) lassen sich 3D-Darstellungen aus zweidimensionalen Zeichnungen mittels einer diffusionsbasierten Aufpumpmethode herleiten. Dies ergibt 3D-Modelle mit stets flacher Grundfläche, was für deren Anwendungsgebiet im AgentCubes-System allerdings kein Nachteil ist.

Wie bereits oben erwähnt, gehen die generischen 3D-Darstellungen auf im DEViL-System verwendeten generischen Zeichnungen zurück [Sch06, S. 172ff.], [Sch03a; Sch03b]. Die auch dort verwendeten Dehnungsintervalle wurden vom *VPE-System* [Gra98] motiviert, welches ebenfalls Strukturediteren für zweidimensionale visuelle Spracheditoren generiert.

KAPITEL 5

# Layout

Zentraler Aspekt visueller Sprachen oder allgemein jedes visuell repräsentierten Diagramms ist dessen Layout. Das umfasst die Platzierung von Einzelkomponenten sowie mögliche Verbindungen zwischen diesen oder auf andere Weise ausgedrückte Relationen. Das Layoutergebnis bestimmt den visuellen Eindruck nach objektiven und subjektiven Kriterien. In manchen visuellen Sprachen trägt das Layout sogar semantische Information und ist in solchen Fällen von besonders zentraler Bedeutung.

Struktureditoren sollen Anwender beim Layout eines visuellen Diagramms unterstützen. Bei der Entwicklung von DEViL3D mussten für die generierten Struktureditoren für 3D-Sprachen verschiedene Layoutaspekte beachtet werden. Das Layout muss von zwei Seiten betrachtet werden: Zum einen müssen Sprachentwickler darin unterstützt werden, für ihre visuelle Sprache mit möglichst geringem Aufwand aber effizient ein Layout zu spezifizieren. Auf der anderen Seite sollen besonders dem Anwender des generierten Struktureditors möglichst viele Layoutentscheidungen abgenommen werden.

Viele aus dem Zweidimensionalen bekannte Layouttechniken sind auf die dritte Dimension erweiterbar und können daher übernommen werden. Neue Aspekte sind hingegen das Verdecken von Sprachkonstrukten, was besonders von Bedeutung ist, wenn Sprachkonstrukte ineinander geschachtelt werden. Das Andocken von Sprachkonstrukten ist vor allem bei Sprachen aus der Architektur-Domäne relevant, da es die Rolle der Gravitation übernehmen kann. Das Layout von planaren Graphen in 3D bietet weitere theoretische Vorteile, da deren Knoten so positioniert werden können, dass sich die Graphkanten nicht durchdringen.

Das Layout für visuelle Sprachen kann in vier Kategorien eingeteilt werden. Das *musterspezifische Layout* wird vom zugrundeliegenden visuellen Muster vor-

gegeben und kann mehr oder weniger restriktiv sein, was durch Layoutfreiheiten (siehe Abschnitt 2.1) bestimmt wird. So besitzt z. B. das Quader-Muster keine Layoutfreiheiten und gibt starre Regeln zur Anordnung von Objekten vor. Das 3D-Mengen-Muster hingegen erlaubt dem Anwender die Position eines Objekts in einem fest vorgegebenen Bereich zu definieren. Layouteigenschaften, die von Editornutzern bestimmt werden können, werden in der Kategorie *benutzerspezifisches Layout* subsumiert. Das *Layout allgemeiner Natur* gibt recht universelle Regeln vor, die sicherstellen, dass sich Sprachkonstrukte nicht durchdringen, in festem Abstand zueinander angeordnet sind oder aneinander andocken. Hat die betrachtete visuelle Sprache sehr individuelle Layoutvorgaben, die durch die anderen drei Layoutkategorien nicht abgedeckt werden, müssen *sprachspezifische Layoutregeln* definiert werden.

Dieses Kapitel geht zunächst auf Layouteigenschaften ein, die durch visuelle Muster vorgegeben werden. Danach wird der Aspekt der Layoutfreiheit genauer betrachtet. Eine zentrale Layoutmethode ist die Schachtelung, auf die ich in Abschnitt 5.3 eingehe. Auf Verfahren zur Nicht-Durchdringung von Sprachkonstrukten wird in Abschnitt 5.2 vorgestellt. Ein wichtiger Mechanismus zur Realisierung von Schachtelungen ist das Dehnungsverhalten der in Abschnitt 4.7 eingeführten generischen 3D-Darstellungen. Anschließend beschreibe ich constraintbasiertes Layout und skizziere, für welche Zwecke es in DEViL3D zur Anwendung kommt. Bevor ich am Ende des Kapitels verwandte Arbeiten vorstelle, zeige ich am Beispiel für Molekülmodelle, wie sich sprachspezifisches Layout realisiert lässt.

## 5.1 Layouteigenschaften der visuellen Muster

Die visuellen Muster sind das zentrale Konzept zur Festlegung eines Layouts von mit DEViL3D spezifizierten dreidimensionalen Sprachen. Die visuellen Muster kapseln parametrisierbare Layouteigenschaften, die vom Sprachspezifizierer an die Gegebenheiten seiner Sprache angepasst werden können. Das Layoutkonzept eines visuellen Musters ist für den Designer einer Sprache bei der Auswahl eines passenden Musters von zentraler Bedeutung. So können die Layoutanforderungen der zu entwickelnden Sprache mit dem Angebot visueller Muster in der Muster-Bibliothek abgeglichen werden. In Abschnitt 4.9 wird gezeigt, dass die zur Verfügung stehenden Muster ein sehr breites Spektrum visueller 3D-Sprachen abdecken. Häufig verwendete Muster sind die Mengen- und Listen-Muster.

Eine wichtige Layouteigenschaft der visuellen Muster ist deren Kombinierbarkeit. Eine visuelle Sprache kann nur in den wenigsten Fällen durch Verwendung eines einzelnen Musters realisiert werden. Ein gutes Beispiel für die Kombination einzelner Muster ist bei 3D-Petri-Netzen (siehe Abschnitt 2.2.7) zu sehen: Die Teilaspekte des Petri-Netzes werden durch Anwendung des Listen-Musters gestapelt,

das Ebenen-Muster erlaubt die Platzierung der Stellen und Transitionen und die Tokens innerhalb der Stellen sind Elemente des 3D-Mengen-Musters. Um visuelle Muster miteinander kombinierbar zu machen, bieten sie bestimmte Schnittstellen an und erwarten andere. Dieses Konzept wurde vom Vorgängersystem DEViL übernommen. Eine ausführliche Beschreibung ist bei Schmidt [Sch06, S. 149ff.] zu finden.

Darüber hinaus gibt es musterübergreifende Layoutstrategien, die Abhängigkeitsschemata zwischen Einzelentscheidungen im Layoutprozess definieren (siehe hierzu eine ausführlichere Darstellung bei [Sch06, S. 147ff.]). Die visuellen Muster in DEViL3D verwenden entweder das *GrowingBox-Layout* oder das *Layer-Layout* als musterübergreifende Layoutstrategie. Das GrowingBox-Layout wird verwendet, wenn geschachtelte Objekte eine Rolle spielen. Da der Größenbedarf eines Objekts von eingeschachtelten Objekten abhängen kann, wird in einer ersten Phase der Größenbedarf von innen nach außen propagiert. Der verfügbare Darstellungsraum wird in einer zweiten Phase von außen nach innen propagiert. Der verfügbare Raum kann dabei unter Umständen größer sein als der benötigte, sodass er an die Unterobjekte weitergegeben werden kann. Das Layer-Layout wird beim Layout von Verbindungen verwendet. Dabei werden die beteiligten Objekte, basierend auf ihren Klassen, in Ebenen aufgeteilt und schrittweise bearbeitet. In jedem Schritt wird das Layout einer Ebene berechnet, wobei Layoutinformationen der bereits bearbeiteten Ebenen verwendet aber nicht mehr geändert werden können. Die erste Ebene umfasst die zu verbindenden Objekte und die zweite Ebene die Verbindungen selber. Die Verbindungen haben allerdings keinen Einfluss auf die Positionierung der durch sie verbundenen Objekte.

Ein weiterer wichtiger Layoutaspekt der visuellen Muster sind die Layoutfreiheiten. Sie können entweder kontinuierlich oder diskret sein. Kontinuierliche Layoutfreiheiten liegen z. B. bei der Positionierung eines Sprachkonstrukts im Mengen-Muster vor. Dort kann der Editoranwender die Position der Mengenelemente innerhalb eines vorgegebenen Bereichs selbst bestimmen. Diskrete Layoutfreiheiten liegen z. B. beim Verbindungs-Muster vor und geben an, mit welchem Sprachkonstrukt eine Verbindung assoziiert ist. Je mehr Layoutfreiheiten eine Sprache aufweist, desto höher ist deren Viskosität [GP96], da Änderungen zeitaufwendig sind, wenn das Layout von Hand angepasst werden muss.

## 5.2 Durchdringungsfreiheit

Ist in zweidimensionalen Sprachen die Rede davon, dass sich Sprachkonstrukte überlappen, so spricht man im Dreidimensionalen von Durchdringen. Ein solcher Effekt ist meist nicht erwünscht, wenn es nicht gerade wie bei der Schachtelung (siehe folgenden Abschnitt 5.3) explizit gewünscht ist. Die Durchdringung von

Sprachkonstrukten ist möglich, wenn die ihnen zugeordneten visuellen Muster den Editoranwendern Layoutfreiheiten lassen, um die Position der Sprachkonstrukte in einem gewissen Rahmen selbst zu bestimmen, wie es bei den Mengen-Mustern der Fall ist.

In visuellen Mustern ohne Layoutfreiheiten wird in der Regel automatisch sichergestellt, dass sich Sprachkonstrukte nicht durchdringen. Zum Beispiel bei Listen hat der Editoranwender keine Möglichkeit, eine kontinuierliche Positionsänderung von Elementen der Liste vorzunehmen. Nur die diskrete Position innerhalb der Liste ist festzulegen. Zwischen den Listenelementen gibt es einen festen Abstand, der vom Editornutzer nicht verändert werden kann. Festgelegt wird dieser Abstand vom Sprachspezifizierer durch ein Kontrollattribut im visuellen Muster.[1] Ähnlich verhält es sich bei anderen Mustern, wie z. B. bei den Matrix-, Quader- oder Kegelbaum-Mustern.

Der folgende Abschnitt geht auf die Durchdringungskorrektur für Mengenelemente ein. Abschnitt 5.2.2 beschreibt Aspekte der Durchdringungsfreiheit von Graphkanten im Dreidimensionalen.

### 5.2.1 Sicherstellung der Durchdringungsfreiheit für Mengenelemente

Editoranwender können Mengenelemente innerhalb eines vorgegebenen Bereichs frei positionieren. Die Position und die Größe der Elemente werden persistent in der editierbaren Struktur der Sprache gespeichert. Beim Einfügen oder auch beim nachträglichen Verschieben[2] können sich Elemente gegenseitig durchdringen. Da dies meist nicht gewollt ist, stellen die Mengen-Muster automatisch einen Algorithmus bereit, der sicherstellt, dass Elemente durchdringungsfrei angeordnet werden. Sollte in gewissen Situationen eine Durchdringung doch gewünscht sein, kann der Algorithmus per Kontrollattribut deaktiviert werden.

Der Algorithmus verfolgt eine deterministische Strategie zum Auflösen der Durchdringung, in dessen Verlauf Elemente verschoben werden. Um das Layout des Diagramms nur minimal zu verändern, werden die Positionsänderungen möglichst gering gehalten. Durchdringen sich zwei Elemente, kann dies durch Verschieben eines der Elemente entlang einer der drei Dimensionsachsen aufgelöst werden. Es wird ermittelt, entlang welcher Dimension die Durchdringung minimal ist, um dann eines der beteiligten Elemente entlang dieser Dimension zu verschieben. Durch das Verschieben von Elemente kann es zu erneuten Durchdrin-

[1] Genau genommen kann das Kontrollattribut für den Abstand der Listenelemente mit einem negativen Wert überschrieben werden, was eine gleichmäßige Durchdringung der Listenelemente zur Folge hätte. Allerdings wäre dies eine dem Konzept einer Liste zuwider strebende Benutzung.

[2] In den Abschnitten 6.2 und 6.4 wird beschrieben, welche Techniken ein Strukturediteor zum Einfügen und Verschieben anbietet.

gungen kommen. Daher wird das beschriebene Verfahren so lange angewendet, bis insgesamt keine Durchdringungen mehr existieren.

Ein gutes Beispiel dafür, dass die Durchdringungseigenschaft Einfluss auf die Semantik einer Sprache haben kann, ist bei der Sprache für generische 3D-Darstellungen (siehe Abschnitt 4.7) zu finden, bei der sich Dehnungsintervalle nicht überlappen dürfen. Das stellt sicher, dass eindeutig definiert ist, welches Stretchintervall für das Stretchverhalten (vgl. Abschnitt 5.3.1) welches Containers zuständig ist. Durch den Algorithmus für die Durchdringungsfreiheit wird dies automatisch sichergestellt.

### 5.2.2 Durchdringungsfreiheit von Graphkanten

Dieser Abschnitt skizziert einen Vorteil von 3D-Darstellungen beim Layout von Graphen. Jeder Graph hat eine 3D-Repräsentation[3], bei der Graphkanten durchdringungsfrei[4] dargestellt werden. Die Durchdringungsfreiheit ist eine zentrale Qualitätsmetrik für Graphen. Schneiden sich Kanten, beeinflusst dies die Lesbarkeit eines Graphen negativ [PCJ97]. Außerdem ist ein Graph für Menschen umso schwieriger zu verstehen, je mehr seiner Kanten sich überschneiden.

Einen guten Überblick über 3D-Graphkonstruktionen geben Dujmović und Whitesides [DW13]. Ein *Fáry-Graph*[5] [Fár48] ist planar, besitzt also keine sich schneidenden Kanten, und kann mit geradlinigen Kanten gezeichnet werden. Die Knoten in dreidimensionalen Fáry-Gitter-Graphen haben eindeutige Positionen mit Integer-Koordinaten. Die Kanten dürfen sich nur an gemeinsamen Endpunkten schneiden und eine Kante schneidet ausschließlich die zwei Knoten, die ihre Endpunkte sind (siehe Abbildung 5.1).

Pach et al. [PTT97] haben bewiesen, dass jeder $r$-färbbare (mit $r > 2$) Graph mit $n$ Knoten eine dreidimensionale Fáry-Gitter-Repräsentation besitzt. Alle Knoten, die einer gemeinsamen Farbklasse angehören, bekommen die gleiche $x$-Position. Die Durchdringungsfreiheit der Graphkanten wird dann durch die geeignete Wahl der $y$- und $z$-Koordinaten der Knoten sichergestellt.

Selbst wenn ein 3D-Graph keine sich schneidenden Kanten besitzt, wird er zur Darstellung auf einem Monitor auf eine 2D-Fläche projiziert. In Abhängigkeit des Blickwinkels hat es den Anschein, dass sich die Graphkanten schneiden, auch wenn dies nur in der 2D-Projektion der Fall ist. Eades et al. [EHW97] und Bose

[3] In der Graphentheorie wird die Darstellung eines Graphen auf einer bestimmten Oberfläche (engl. *surface*) als *Einbettung* bezeichnet. Die 3D-Repräsentation eines Graphen ist eine Einbettung in $\mathbb{R}^3$.

[4] Im Zweidimensionalen ist von *Überschneidungsfreiheit* die Rede. Da der Fokus meiner Betrachtungen sich auf den dreidimensionalen Raum bezieht, spreche ich konsistent von Durchdringungsfreiheit.

[5] In der Literatur werden diese Graphen häufig auch als *Fáry-Gitter-Graphen* oder *Gitter-Zeichnungen* (engl.: *grid drawing*) bezeichnet.

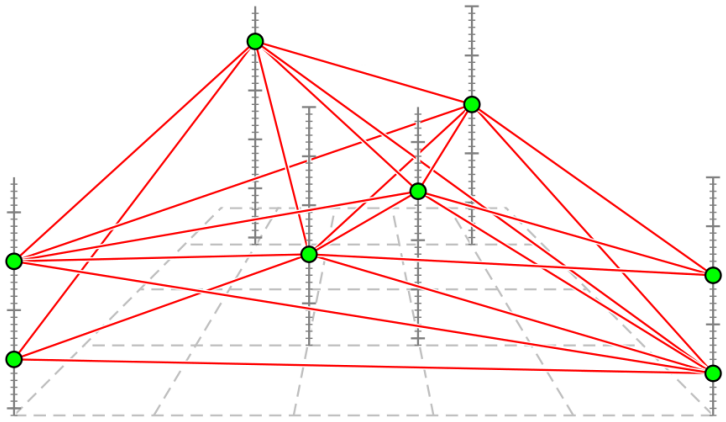


**Abbildung 5.1:** Repräsentation eines dreidimensionalen Fáry-Gitter-Graphen; Quelle: [DW13].

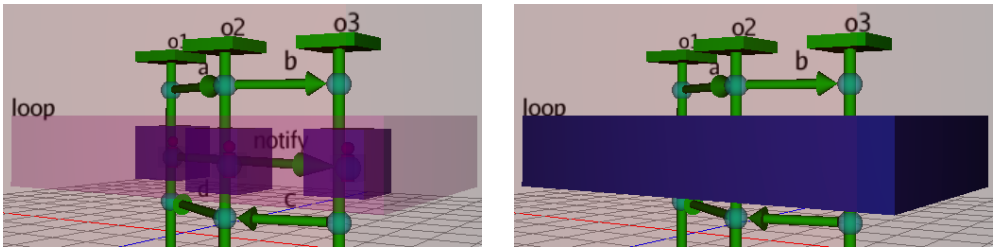
et al. [BGRT99] haben allerdings Methoden entwickelt, wie schlechte Blickwinkel auf 3D-Graphen bei deren Projektion in 2D vermieden werden können. Ein Blickwinkel ist z. B. schlecht, wenn die 3D-Position zweier Graphknoten auf die gleiche 2D-Position abgebildet wird. Bei den Artikeln [EHW97] und [BGRT99] handelt es sich allerdings um theoretische Arbeiten, die eine statische Projektion eines 3D-Graphen auf eine zweidimensionale Zeichenfläche betrachten. In der Praxis eines 3D-Strukturediteors kann der Blickwinkel auf eine 3D-Szene durch geeignete Navigationsmöglichkeiten (siehe Kapitel 6) variiert werden.

## 5.3 Schachtelung

Das Ineinanderschachteln von dreidimensionalen Sprachkonstrukten ist ein natürlicher Weg, um eine Enthaltensein-Relation auszudrücken, bei der innen Befindliches dem Äußeren zugehörig ist. Ein anderer Weg, eine dadurch modellierte hierarchische Struktur darzustellen, sind Kegelbäume (siehe Kapitel 4). Beiden gemein ist die zugrundeliegende baumartige abstrakte Struktur.

Eine Schachtelung im Dreidimensionalen ist prinzipiell mit russischen Matrjoschka-Puppen vergleichbar: Eingeschachtelte Sprachkonstrukte sind vollständig von ihren Eltern-Konstrukten umschlossen. Es gibt also eine physische Inklusion, was bedeutet, dass die inneren Objekte ohne die äußeren nicht existieren können. Auf struktureller Ebene ist dies bei Schachtelung in zweidimensionalen visuellen Sprachen ebenfalls so. Der große Unterschied ist, dass im Dreidimensionalen die inneren Objekte räumlich umschlossen werden.

Das räumliche Umschließen von Sprachkonstrukten bringt einen weiteren bedeutenden Faktor ins Spiel: das visuelle Erscheinungsbild bzw. die Transparenz der in einer Schachtelung beteiligten Sprachkonstrukte. Sind die umgebenden Sprach-




**(a)** Transparentes Sprachkonstrukt. **(b)** Opakes Sprachkonstrukt.

**Abbildung 5.2:** Prinzip der Sichtbar- und Unsichtbarmachung eingeschachtelter Sprachkonstrukte.

konstrukte opak, verdecken sie ihren Inhalt. Daraus kann allerdings auch ein Layoutprinzip abgeleitet werden, welches einem Strukturedi tor-Anwender ermöglicht, temporär nicht im Fokus stehende Sachverhalte auszublenden und bei Bedarf wieder sichtbar zu machen. Abbildung 5.2 zeigt eine Anwendung dieses Prinzips aus einem mit DEViL3D generierten Strukturedi tor für UML-Sequenzdiagramme, in dem der Inhalt kombinierter Fragmente ausgeblendet werden kann.

Werden viele Sprachkonstrukte ineinander geschachtelt, ergeben sich schnell weitere Probleme hinsichtlich der Übersichtlichkeit der entstehenden Darstellung. Bei der Entwicklung des 3D-Sequenzdiagrammeditors ergaben sich schnell Hierarchien mit einer Schachtelungstiefe von mehr als vier Sprachkonstrukten. Um diese tiefen Schachtelungen zu vermeiden, wurden weitere eingeschachtelte kombinierte Fragmente ausgelagert, um die Szene nicht zu unübersichtlich werden zu lassen (siehe [Stü14, S. 26ff.]).

Die Entscheidung, wie Schachtelung in einer 3D-Sprache eingesetzt wird, obliegt dem Sprachentwerfer. Eine Schachtelungstiefe von mehr als drei Sprachkonstrukten ist meist nicht sinnvoll. Das Schachteln von Sprachkonstrukten selber wird mithilfe generischer 3D-Darstellungen realisiert. Der nachfolgende Abschnitt widmet sich diesem Aspekt. Auf die Herausforderungen, die sich hinsichtlich der Interaktion mit geschachtelten hierarchischen Strukturen ergeben, wird in Abschnitt 6.3.2 eingegangen.

### 5.3.1 Dehnungsverhalten generischer 3D-Darstellungen

Die generischen 3D-Darstellungen können, wie in Abschnitt 4.7 beschrieben, Container umfassen, die es erlauben beliebige Unterelemente aufzunehmen, um diese damit in die 3D-Darstellung einzuschachteln. Benötigen die Unterelemente mehr Platz als ein Container aktuell bietet, müssen die grafischen Primitive und die Container der 3D-Darstellung an den erhöhten Platzbedarf angepasst, also ausgedehnt werden. Verantwortlich dafür ist ein Dehnungsalgorithmus, der 3D-Darstellungen

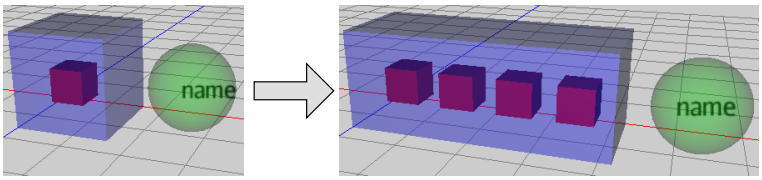



**Abbildung 5.3:** Eine instanziierte 3D-Darstellung vor und nach Ausdehnung.

automatisch ausdehnt, sobald der Platz für Unterelemente nicht mehr ausreicht [Wol13a]. Dies erfolgt unabhängig vom visuellen Muster, nach dem die Unterelemente repräsentiert werden.

Bei Instanziierung der generischen 3D-Darstellung aus Abbildung 4.6 auf Seite 74 werden in den ersten Container rote Würfel eingefügt, die als Liste angeordnet werden, die entlang einer Dimension wächst. Abbildung 5.3 zeigt beispielhaft, wie bei wachsender Anzahl von Listenelementen der den Listen-Container umfassende blaue Quader ausgedehnt wird und die grüne Kugel entsprechend aufrückt. Um dieses Ergebnis zu erzielen, arbeitet der Dehnungsalgorithmus wie folgt: Der Algorithmus iteriert für jede der drei Raumdimensionen über alle Container der 3D-Darstellung. Dabei wird geprüft, ob die benötigte Größe des Containers, die durch den Platzbedarf der eingeschachtelten Elemente bestimmt wird, die aktuelle Größe übersteigt. Ist dies der Fall, werden alle Dehnungsintervalle berechnet, die den Container schneiden. Danach werden die Teile des Containers oder eines grafischen Primitives, welche vom Dehnungsintervall geschnitten werden, linear ausgedehnt.

Abbildung 5.4 skizziert das Vorgehen schematisch. Dargestellt ist die generische 3D-Darstellung aus Abbildung 4.6 reduziert auf die $x$-Achse, um das Vorgehen des Algorithmus für eine Dimension zu zeigen. Initial ist die Größe des Containers `c1` $a$. Der benötigte Platz der eingeschachtelten Elemente ist mit $a + b$ größer. Somit muss der Container ausgedehnt werden, um sich dem Platzbedarf anzupassen. Der dafür notwendige Ausdehnungsprozess kann dadurch veranschaulicht werden, wenn man sich vorstellt, die Container und die grafischen Primitive wären auf ein elastisches Gummi gedruckt. Durch Ziehen an den Rändern der

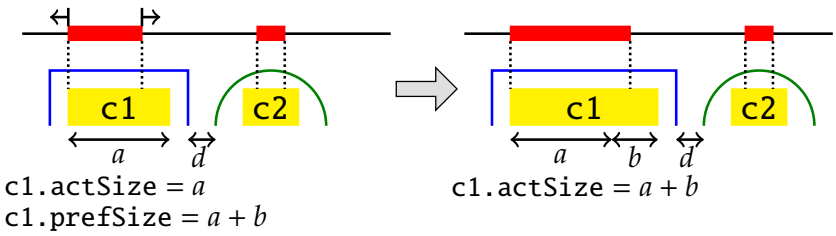



**Abbildung 5.4:** Verhalten des Algorithmus zur Dehnung der 3D-Darstellungen.

Dehnungsintervalle dehnt sich die 3D-Darstellung bis die notwendige Größe erreicht ist. Bereiche, die nicht von einem Dehnungsintervall geschnitten werden, werden somit nicht ausgedehnt. Der Abstand zwischen den einzelnen Bestandteilen der 3D-Darstellung bleibt bei diesem Verfahren unverändert. So z. B. auch der Abstand *d* zwischen dem Quader und der Kugel. Die Kugel rückt samt des in ihr enthaltenen Containers `c2` nach rechts auf.

Der Dehnungsalgorithmus ist ins DEViL3D-System integriert und wird automatisch aufgerufen, wenn neue Elemente in einen Container eingefügt werden. Daher muss sich der Sprachentwickler über all diese Aspekte keine Gedanken machen und kann sicher sein, dass sich die 3D-Darstellungen seines spezifizierten Editors jederzeit an eingeschachtelte Unterelemente anpassen. Für den Anwender des Editors stellt die automatische Anpassung der 3D-Darstellungen das natürlich erwartete Verhalten sicher.

## 5.4 Constraintbasiertes Layout

Beziehungen zwischen Entitäten lassen sich oftmals durch Constraints, also Einschränkungen, beschreiben. Bei diesem deklarativen Ansatz werden Beziehungen durch eine Menge von Gleichungen und Ungleichungen ausgedrückt. Die Berechnung von Layouts mit diesem Verfahren ist besonders prädestiniert, da Gleichungen und Ungleichungen zwischen für das Layout relevanten Eigenschaften, wie Größe und Position von Objekten, einfach beschrieben werden können. Das Lösen dieses Gleichungssystems obliegt dann einem sogenannten *Constraint-Solver*, der für alle im Gleichungssystem vorkommenden Variablen Werte aus einem vorgegebenen Wertebereich finden muss, die das Gleichungssystem erfüllen. Findet der Constraint-Solver für alle Variablen eine gültige Belegung ist das Gleichungssystem lösbar. Allerdings ist in den meisten Fällen die Lösung nicht eindeutig, da es viele verschiedene Lösungen gibt. In solchen Fällen ist das Gleichungssystem bzw. sind die Constraints *unterspezifiziert*. Sind die Constraints *überspezifiziert*, lässt sich keine Lösung des Gleichungssystems finden.

Bei unterspezifizierten Constraints obliegt es dem Constraint-Solver, aus der Menge korrekter Lösungen eine auszuwählen. Zu diesem Zweck gibt es zahlreiche Vorgehensweisen und Heuristiken, diejenige Lösung zu ermitteln, die den Erwartungen des Benutzers am besten entspricht und ihn am wenigsten überrascht. Weiterhin benötigen Constraint-Solver zum Finden einer Lösung für ein größer werdendes Gleichungssystem oftmals viel Zeit. Aus diesem Grund sieht Minas [Min01, S.136] die Anwendung für constraintbasiertes Layout höchstens für Rapid-Prototyping von Spracheditoren. Jung hat constraintbasiertes Layout in VL-Eli für Sprachkonstrukte mit Layoutfreiheiten eingesetzt. Für Sprachkonstruk-

te ohne Layoutfreiheiten sieht er die Layoutberechnungen durch den Attributauswerter als effizienter an.

Diesem Ansatz folgend wird in DEViL3D constraintbasiertes Layout nur punktuell für Sprachkonstrukte mit Layoutfreiheiten verwendet. Dieser constraintbasierte Ansatz ist nicht als Konkurrenz zu visuellen Mustern zu verstehen, sondern zur Lösung von speziellen Aufgaben. Die Sprachkonstrukte, die als Elemente eines der drei Mengen-Muster dargestellt werden, weisen Layoutfreiheiten auf. Für diese wurden auf Constraints basierende Alternativen für die Durchdringungsfreiheit (siehe Abschnitt 5.2) und die Anordnung auf einem Raster implementiert.

Zur Lösung der Constraints wurde der Constraint-Solver *Choco* verwendet. Mit Choco lassen sich verschiedenartige Constraints modellieren, die anschließend vom integrierten Constraint-Solver gelöst werden. Zur Modellierung der Constraints stehen Variablen verschiedenen Typs zur Verfügung; neben Integer- und Real-Variablen auch Mengen-Variablen. Diese können durch arithmetische, boolsche oder tabellenbasierte Constraints in Beziehung gesetzt werden.

### 5.4.1 Formulierung der Durchdringungsfreiheit mit Constraints

Basierte die händische Implementierung der Durchdringungsfreiheit noch auf einer speziellen Strategie, wie eine Durchdringung von zwei Objekten aufzulösen ist, müssen beim constraintbasierten Ansatz nur eine Menge von Constraints und deren disjunktive oder konjunktive Verbindung definiert werden. Für Sprachkonstrukte $o_1$ und $o_2$ in einem 3D-Mengen-Muster, die sich in allen drei Dimensionen nicht durchdringen sollen, ergibt sich folgendes Ungleichungssystem:

$$
\begin{aligned}
&\Big( \quad pos_d^{o_1} + size_d^{o_1} \leq pos_d^{o_2} \\
&\quad \vee \; pos_d^{o_2} + size_d^{o_2} \leq pos_d^{o_1} \Big) \\
\wedge \quad & pos_d^{o_1} \geq 0 \\
\wedge \quad & pos_d^{o_2} \geq 0 \qquad \forall d \in \{x, y, z\}
\end{aligned}
$$

Wobei $size_d^{o_1}$ und $size_d^{o_2}$ Konstanten sind und $pos_d^{o_1}$ und $pos_d^{o_2}$ Variablen, für die der Constraint-Solver Belegungen finden muss.

Werden dem Constraint-Solver diese Constraints für alle Kombinationen zwischen im Mengen-Muster vorhandenen Sprachkonstrukten hinzugefügt, zeigt sich ein auf den ersten Blick unerwartetes Verhalten: Auch in Situationen, in denen sich keine Sprachkonstrukte durchdringen, variiert die Position der Sprachkonstrukte beim Aktualisieren der 3D-Szene leicht. Dies hängt damit zusammen, dass von dem verwendeten Constraint-Solver unabhängig von einer möglichen Durchdringung das obige Ungleichungssystem gelöst wird. Grund dafür ist, dass die Menge

der Constraints stark unterspezifiziert sind, was zu einem nichtdeterministischen – oder nach Schmidt [Sch06, S. 152] sogar zu „unberechenbarem“ – Verhalten führt.

Da ein solches unvorhersehbares Verhalten für Editornutzer schwer nachvollziehbar und sicherlich nicht gewollt ist, werden die Constraints nur für Sprachkonstrukte eingeführt, die mit anderen durchdrungen sind. Dadurch behalten die meisten Sprachkonstrukte ihre Positionen bei und nur diejenigen, die an einer Kollision beteiligt sind, werden verschoben. Durch das Auflösen einer Kollision kann allerdings eine neue Kollision zwischen anderen Sprachkonstrukten entstehen, wenn diese durch das Auflösen der ersten Kollision ineinandergeschoben werden. Dies wird durch das Lösen eines erneuten Constraintsystems aufgelöst und zwar so lange, bis alle Elemente durchdringungsfrei angeordnet sind.

Der Sprachspezifizierer kann mithilfe eines Kontrollattributs zwischen dem constraintbasierten und von Hand implementierten Algorithmus zur Sicherstellung der Durchdringungsfreiheit wählen. Vergleicht man die Ergebnisse beider Verfahren, fällt auf, dass beim von Hand implementierten Verfahren oft nur ein Sprachkonstrukt verschoben wird, wohingegen sich im constraintbasierten Ansatz meist alle an einer Durchdringung beteiligten Sprachkonstrukte, wenn auch nur minimal, bewegen.

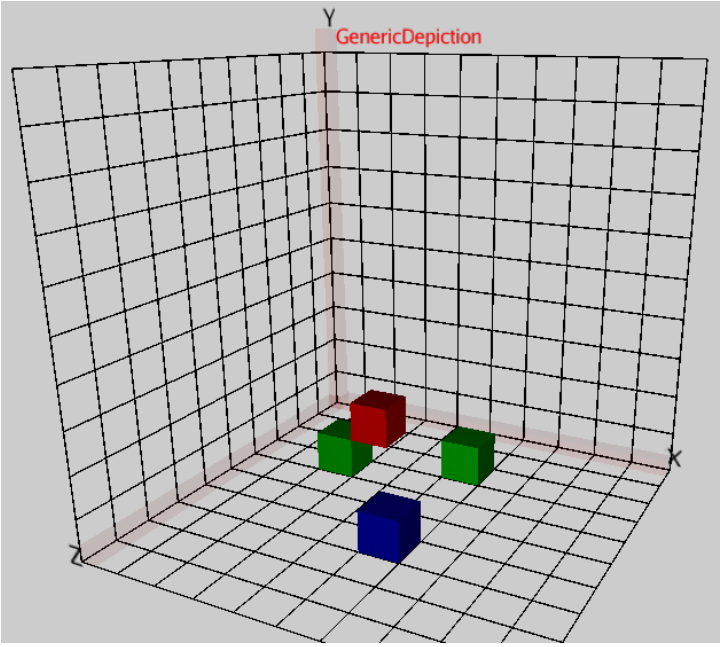



**Abbildung 5.5:** Sprachkonstrukte werden auf einem gleichmäßigem Raster angeordnet.

### 5.4.2 Anordnung von 3D-Mengenelementen auf einem Raster

Wenn 3D-Mengenelemente innerhalb des vorgegebenen Bereichs in regelmäßigen Abständen angeordnet werden sollen, bietet sich dafür ein Raster an. Das 3D-Mengen-Muster unterstützt diese Layouterweiterung, die durch ein Kontrollattribut aktiviert werden kann. Ein Beispiel dafür ist in Abbildung 5.5 zu sehen, in dem im Editor für generische 3D-Darstellungen die Sprachkonstrukte auf einem Raster mit einer Schrittweite von zehn Pixeln angeordnet werden.

Realisiert wird die Anordnung mit einer Menge von Constraints, die von der Schrittweite des Rasters *stepSize* und der ursprünglichen Position des Sprachkonstrukts $objectPosOld_d$ abhängen. Die Constraints beschreiben das Problem so genau, dass das Ungleichungssystem weder über- noch unterspezifiziert ist, sondern immer genau eine Lösung berechnet.

$$\begin{aligned} objectPosNew_d &= factor \cdot stepSize \\ 2 \cdot (objectPosNew_d - objectPosOld_d) &\leq stepSize \\ 2 \cdot (objectPosOld_d - objectPosNew_d) &\leq stepSize \end{aligned}$$

Dabei sind *stepSize* und $objectPosOld_d$ Konstanten sowie $objectPosNew_d \in \mathbb{R}$ und $factor \in \mathbb{N}_0$ Variablen, für die der Constraint-Solver Belegungen finden muss.

## 5.5 Sprachspezifisches Layout

Sprachspezifisches Layout beschreibt im Allgemeinen dasjenige Layout, welches zur korrekten Darstellung einer visuellen Sprache notwendig ist. In den meisten Fällen lässt sich die Darstellung und somit das Layout einer Sprache durch Anwendung von visuellen Mustern und deren Parametrisierung erreichen. So lässt sich z. B. das Layout der 3D-Sprache ToneCraft (siehe Abschnitt 2.2.10) zur Komposition von Musik durch Anwendung des Matrix- und Listen-Musters vollständig spezifizieren.

In seltenen Fällen sind die Layoutanforderungen der Sprache so speziell, dass sie nicht vollständig durch Muster-Anwendungen und deren Parametrisierung erreicht werden können. So verhält es sich z. B. bei Molekülmodellen, bei denen die Anordnung der Atome von chemischen Regeln abhängig ist. In solchen Fällen müssen für ein korrektes Layout zusätzliche sprachspezifische Constraints[6] hinzugefügt werden, die sich nicht automatisch durch Parametrisierung mittels Kontrollattributen in visuellen Mustern realisieren lassen.

[6] Mit Constraints sind hier allgemein weitere layoutspezifische Anforderungen gemeint, die nicht zwangsläufig durch Constraint-Programmierung (siehe Abschnitt 5.4) gelöst werden.

Wenn in dieser Arbeit von sprachspezifischem Layout die Rede ist, so ist der zweite Fall gemeint, bei dem das Layout nicht ausschließlich durch Anwendung visueller Muster erreicht werden kann. Im Folgenden werde ich ausführlich das anspruchsvolle Layout für Molekülmodelle vorstellen und skizzieren, wie ein Spracheentwickler mit DEViL3D solche Layoutanforderungen umsetzen kann.

### 5.5.1 Berechnung des Layouts für Molekülmodelle

Wie bereits in Abschnitt 2.3 erwähnt, kann das Kugel-Stäbchen-Modell für Molekülmodelle (siehe Abschnitt 2.2.9) als inhärent dreidimensionale Sprache aufgefasst werden. Die Dreidimensionalität erlangt das Modell durch die Positionierung der Atome im 3D-Raum in Abhängigkeit abstoßender Kräfte zwischen Elektronenpaaren. Darüber, wie die Position der Atome und die daraus resultierende *Molekülstruktur*[7] genau zu berechnen ist, gibt es in der Chemie verschiedene Theorien. Eine weithin akzeptierte Theorie ist die Berechnung der Molekülstruktur durch das VSEPR-Modell[8] [GR05], welches 1957 von Gillespie und Nyholm entwickelt wurde. Es sagt die Molekülstruktur auf Grundlage der Abstoßung von Elektronenwolken vorher, die um die bei der Atombindung beteiligten Elektronen entstehen. Die Elektronenpaare in der *Valenzschale*[9] eines Atoms stoßen sich gegenseitig ab und nehmen eine Position ein, die die Abstoßung minimiert und die Distanz zwischen den Atomen maximiert. Elektronenpaare in der Valenzschale können entweder gebunden oder ungebunden sein, wobei die ungebundenen Elektronenpaare mehr Platz beanspruchen. Als Ausgangspunkt benötigt das VSEPR-Modell das *Zentralatom*, welches im Zentrum des Moleküls positioniert ist. Die Molekülstruktur kann dann anhand der Anzahl und des Typs (gebunden oder ungebunden) der Elektronenpaare der Valenzschale, die sich um das Zentralatom herum anordnen, vorhergesagt werden.

Mit DEViL3D wurde ein 3D-Spracheditor spezifiziert, der es erlaubt, eine Auswahl der Atome der Periodentabelle und Bindungen zwischen diesen in die 3D-Szene einzufügen. Die visuelle Darstellung der Atome erfolgt mithilfe des 3D-Mengen-Musters, die der Bindungen durch das Verbindungs-Muster. Ein so spezifizierter Editor erlaubt beliebige Positionierungen der Atome und berücksichtigt in keiner Weise die Molekülstruktur, da dies die generisch definierten visuellen Muster nicht leisten können. Allerdings bietet DEViL3D Sprachspezifizierern Erweiterungsstellen zur Spezifikation von sprachspezifischem Layout an [Wol14]. Abbildung 5.6 zeigt, wie das Methan-Molekül ($CH_4$) im mit DEViL3D generierten

[7] Die Molekülstruktur ist der Fachbegriff, der die geometrische, räumliche relative Anordnung der Atome in einem Molekül bezeichnet.

[8] Das VSEPR-Modell lässt sich allerdings nicht auf alle erdenklichen Moleküle anwenden; so lassen sich z. B. keine Metalle oder Ionenkristalle beschreiben.

[9] Die Valenzschale ist die äußerste mit Elektronen besetzte Schale in der Elektronenhülle.

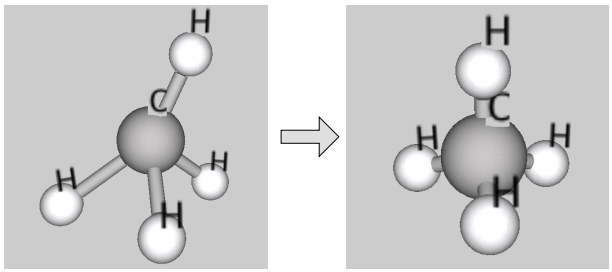



**Abbildung 5.6:** Anwendung des Layouts auf das Methan-Molekül.

Moleküleditor seine Molekülstruktur erlangt: Der Benutzer des Editor konstruiert das Molekül zunächst mit Fokus auf deren strukturelle Aspekte (Anzahl von Atomen und deren Bindungen), ohne auf die exakte Positionierung der Atome zu achten. Danach kann eine Funktion aufgerufen werden, die das Layout des Moleküls berechnet und die Atome umpositioniert. Im Folgenden werde ich zunächst genauer auf das VSEPR-Modell eingehen, welches zur Layoutberechnung genutzt wird. Abschließend werde ich kurz skizzieren, welche Erweiterungsstellen zur Spezifikation des Layouts genutzt werden.

Das VSEPR-Modell benötigt grundlegende Informationen des Moleküls zur Berechnung dessen Layouts. Grundlegend ist die Anzahl der Valenzelektronen eines Atoms. Diese ist in der Periodentabelle der Elemente notiert und in der abstrakten Struktur der Sprache modelliert. Zentrales Verfahren innerhalb des VSEPR-Modells ist die *AXE-Methode*, mit deren Hilfe die an der Bindung beteiligten Elektronen gezählt werden. Das A repräsentiert dabei das Zentralatom eines Moleküls. Die Menge der Atome X, die direkt mit A verbunden sind, werden als *Liganden* bezeichnet. E ist die Anzahl der ungebundenen Valenzelektronenpaare in der Umgebung des Zentralatoms, die häufig auch *Pseudoliganden* genannt werden.

| **Molekültyp** | **sterische Zahl** | **Struktur** | **Winkel** | |
|---|---|---|---|---|
| | | | $\theta$ | $\varphi$ |
| $AX_2$ | 2 | linear | 180° | 90° |
| $AX_3$ | 3 | trigonal-planar | 120° | 90° |
| $AX_4$ | 4 | tetraedrisch | 109,5° | 120° |
| $AX_3E$ | | | 107° | 120° |
| $AX_2E_2$ | | | 104,5° | 180° |
| $AX_5$ | 5 | trigonal-bipyramidal | 90° | 120° |
| $AX_6$ | 6 | oktaedrisch | 90° | 90° |
| $AX_7$ | 7 | pentagonal-bipyramidal | 90° | 72° |

**Tabelle 5.1:** Tabelle zur Vorhersage der Molekülstruktur auf Grundlage der sterischen Zahl.

Die Summe aus X und E ist als *sterische Zahl* bekannt. Das Kohlenstoffatom *C* z. B. besitzt vier Valenzelektronen, das Wasserstoffatom *H* hingegen besitzt nur ein Valenzelektron. Das Methan-Molekül $CH_4$, welches sich aus diesen beiden Atomen zusammensetzt, wird durch die AXE-Methode wie folgt repräsentiert: $AX_4E_{4-4\cdot1} = AX_4E_0$. Beim Wasser-Molekül $H_2O$ gehen nur zwei der sechs Valenzelektronen des Sauerstoffatoms eine Bindung mit den zwei Wasserstoffatomen ein. Die übrigen vier bilden zwei Elektronenpaare, was nach der AXE-Methode so repräsentiert wird: $AX_2E_{\frac{6-2}{2}} = AX_2E_2$. Die zwei Elektronenpaare nehmen etwas mehr Platz in Anspruch, sodass sich der Winkel zwischen den Wasserstoffatomen verringert.

Auf Basis der Informationen der AXE-Methode kann nun das VSEPR-Modell die Molekülstruktur mit den folgenden vier Schritten berechnen:

1. Bestimme das Zentralatom A.
2. Berechne die Menge der Liganden X, die mit A verbunden sind.
3. Bestimme die sterische Zahl und lies mit dieser die Molekülstruktur aus Tabelle 5.1 ab.
4. Berechne die neuen Positionen für die Liganden gemäß der bestimmten Molekülstruktur.

Die Berechnung der Positionen der Liganden aus Schritt 4 erfolgt mithilfe von *Kugelkoordinaten*. Diese geben einen Punkt durch seinen Abstand zum Ursprung sowie zwei Winkeln an. Kugelkoordinaten lassen sich in kartesische Koordinaten umrechnen. Glaister [Gla93] hat Kugelkoordinaten genutzt, um das Layout für Moleküle mit tetraedrischer Struktur (wie z. B. Methan) zu berechnen. Dieser Ansatz wurde für den mit DEViL3D spezifizierten Moleküleditor verallgemeinert, sodass alle in Tabelle 5.1 aufgelisteten Molekülstrukturen berechnet werden können.[10] Für diesen Ansatz wird das Zentralatom als Ursprung des Kugelkoordinatensystems angenommen (siehe Abbildung 5.7) und für jeden Liganden die Kugelkoordinate $(r, \theta, \varphi)$ in kartesische Koordinaten umgerechnet. Der Abstand $r$ zwischen dem Zentralatom und dem Liganden wird durch den *Kovalenzradius* der beiden beteiligten Atome bestimmt, der ebenfalls in der abstrakten Struktur der Sprache vorgehalten

[10] Der Übersichtlichkeit halber listet Tabelle 5.1 nicht alle Molekültypen mit Beteiligung von Pseudoliganden auf. Beispielhaft sind zwei Molekültypen tetraedrischer Struktur mit Pseudoliganden aufgeführt. Dabei fällt auf, dass mit steigenden Pseudoliganden bzw. Valenzelektronenpaaren der Winkel $\theta$ zwischen den bindenden Atomen kleiner wird, da die Elektronenpaare mehr Platz beanspruchen.

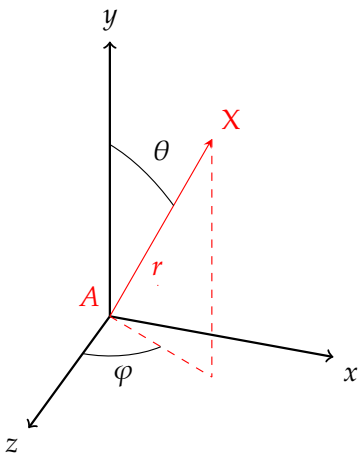



**Abbildung 5.7:** Kugelkoordinaten werden genutzt, um die Position der Liganden zu berechnen.

wird. Die Winkel $\theta$ und $\varphi$ werden aus Tabelle 5.1 abgelesen. Die Umrechnung ins kartesische Koordinatensystem erfolgt mit folgenden Formeln:

$$x = r \cdot \sin\theta \cdot \cos\varphi$$
$$y = r \cdot \sin\theta \cdot \sin\varphi$$
$$z = r \cdot \cos\theta$$

In Abbildung 5.8 sind einige Molekülstrukturen zu sehen, die von dem Moleküleditor berechnet wurden. Eingezeichnet sind die Winkel $\theta$ und $\varphi$, die zu den Angaben in Tabelle 5.1 korrespondieren.

Diese Layoutberechnungen wurden mithilfe der in Kapitel 3 erwähnten und vom DEViL3D-System bereitgestellten Erweiterungsstellen implementiert. Synchronisations-Funktionen für die Atome sorgen dafür, dass nach jedem Editierschritt die Anzahl der gebundenen und freien Valenzelektronen des betreffenden Atoms berechnet werden. Die eigentliche Berechnung des Layouts gemäß der oben vorgestellten vier Schritte wurde in einer benutzerdefinierten Funktion umgesetzt, die aufgerufen wird, sobald der Anwender des Editors diese über das Kontextmenü eines Sprachkonstrukts auswählt.

### 5.5.2 Alternative Layouts

In manchen Sprachen gibt es mehrere Repräsentationen des gleichen Modells. So ist es auch für Molekülmodelle, bei denen die zuvor beschriebenen Kugel-Stäbchen-Modelle nur eine von vielen Repräsentationen sind. Ein weiteres räumliches Modell für Moleküle ist das *Kalottenmodell*, welches den Molekülaufbau plastisch dar-

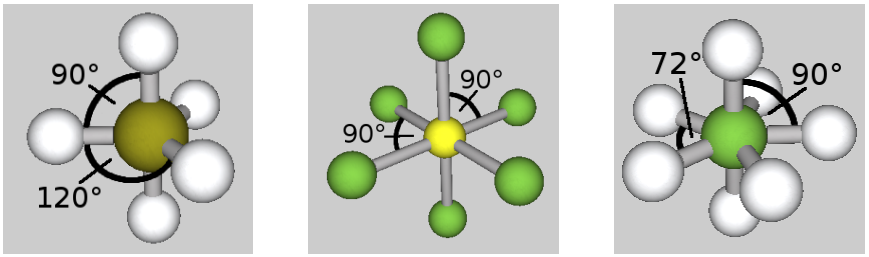
stellen soll. Die Atome werden durch Kugelausschnitte, den sogenannten Kalotten, repräsentiert. Abbildung 5.9 zeigt das Kalottenmodell des Methan-Moleküls in dem von DEViL3D generierten Moleküleditor. Dieses alternative Molekül-Layout wird in einer zusätzlichen vom Editor bereitgestellten Sicht dargestellt.

## 5.6 Zusammenfassende Betrachtungen

Das Kapitel hat verschiedene Techniken vorgestellt, durch die das Layout einer mit DEViL3D spezifizierten Sprache definiert wird. Durch sinnvolle Anwendung der Techniken durch den Sprachentwerfer werden Editoren mit sehr guten Layouteigenschaften generiert. Diese Erkenntnisse werde ich hier kurz resümieren und mit meinen Erfahrungen als Sprachentwerfer untermauern.

Bei der Spezifikation der meisten 3D-Sprachen ist die Anwendung visueller Muster und deren geschickte Parametrisierung ausreichend, um das beabsichtigte Layout für eine Sprache zu erreichen. Das hat bereits Abschnitt 4.9 gezeigt und dabei aufgeführt, welche visuellen Muster die einzelnen Sprachen nutzen. Durch die Muster werden automatisch die Durchdringungsfreiheit sowie die Schachtelung von Sprachkonstrukten sichergestellt. Ein gutes Beispiel dafür ist die Spezifikation der 3D-Sequenzdiagramme, die hierarchisch geschachtelte Strukturen beinhalten. Allerdings ist es dabei die Aufgabe des Sprachentwerfers, die Schachtelungstiefe geeignet zu begrenzen, da sonst die Darstellung unübersichtlich wird.

Für alle Sprachen, die darüber hinausgehende Layoutanforderungen besitzen, stellt DEViL3D Erweiterungsstellen zur Spezifikation benutzerdefinierten Layouts bereit. Für die Sprache der Molekülmodelle wurde dies angewendet, um die Anordnung der Atome gemäß den chemischen Regeln zu realisieren. Alternative Darstellungen derselben Struktur können durch Definition einer weiteren Sicht realisiert werden. So wurde auch die zusätzliche Darstellung der Moleküle als Kalottenmodell realisiert. Unabhängig von der Hauptsicht kann das Layout der




**(a)** Trigonal-bipyramidal. **(b)** Octahedral. **(c)** Pentagonal-bipyramidal.

**Abbildung 5.8:** Beispiele einiger vom Moleküleditor berechneter Molekülstrukturen.

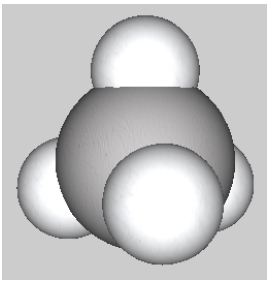


**Abbildung 5.9:** Kalottenmodell des Methan-Moleküls.

alternativen Darstellung bestimmt werden, entweder durch Anwendung anderer Muster oder auch einer zusätzlichen benutzerdefinierten Layoutspezifikation.

## 5.7 Verwandte Arbeiten

Ein sehr ähnliches Vorgehen beim Layout ist im DEViL-System zu finden [Sch06, S. 147ff.]. Die Layoutverfahren werden ebenfalls in visuellen Mustern gekapselt, wodurch ein bedeutender Teil des Layouts und deren Kombination automatisch sichergestellt wird. Die Layoutverfahren in DEViL sind teilweise vom *GIGAS*- und *VPE-System* inspiriert. So werden mit dem GIGAS-System [Fra89] *Layout-Grammatiken* spezifiziert, die Schachtelungsstrukturen für Rechtecke beschreiben. Das VPE-System [Gra98] stellt ein automatisches Layout sicher und inspirierte die DEViL-Entwickler zum Entwurf generischer Vektorgrafik-Zeichnungen, die die Grundlage für DEViL3D's generische 3D-Darstellungen sind. Weitere Details zu den beiden Systemen sind auch in Schmidts Dissertation [Sch06, S. 54ff.] zu finden.

Das Ineinanderschachteln von Objekten ist ein besonders bei 3D-Darstellungen weit verbreitetes Prinzip. Einige 3D-Visualisierungen, wie z. B. beim Informationswürfel, der auf Seite 35 vorgestellt wird, nutzen dieses Prinzip, welches meist eine Enthaltensein-Relation ausdrückt. Mit ähnlicher Semantik findet es auch Anwendung in historischen 3D-Sprachen, wie z. B. Cube, 3D-PP oder SAM.

### Layout-Muster in DiaMeta

In diesem Kontext sind auch die Arbeiten von Maier und Minas interessant [MM07; MM08; MM09; MM10; MM12a; MM12b; Mai12]. Sie haben ein Konzept entwickelt, welches Layoutverfahren in Mustern kapselt und diese miteinander kombinierbar macht. Sie kommen innerhalb von mit DiaMeta (siehe Abschnitt 2.6.2 auf Seite 44) generierten Spracheditoren zum Einsatz. Die Layoutverfahren werden in den Editoren im Zweidimensionalen angewendet, funktionieren konzeptionell aber für beliebig viele Dimensionen.

Es werden 18 Layout-Muster vorgestellt, die jeweils ein spezifisches Layoutproblem lösen. Beispiele sind das *Node Overlap Removal Pattern* oder *Rectangular Containment Pattern*. Ersteres sorgt dafür, dass sich Knoten eines Graphen nicht überlappen und leistet damit ähnliches, wie das in Abschnitt 5.2 beschriebene Verfahren zur Nicht-Durchdringung von 3D-Objekten in DEViL3D. Mit dem zweiten Layout-Muster lassen sich rechteckige Objekte in beliebiger Rekursionstiefe in ein Eltern-Objekt einschachteln. Dies wird im DEViL3D-System für den dreidimensionalen Fall durch Container in generischen 3D-Darstellungen und deren automatischem Dehnungsverhalten (vgl. Abschnitt 5.3.1) realisiert. Weitere Layout-Muster stellen z. B. die gleichmäßige horizontale oder vertikale Ausrichtung von Objekten oder deren Anordnung in einer Zeile oder Spalte sicher.

Jedes der Layout-Muster basiert auf einem von fünf *Pattern-Specific Meta-Models* (PMMs). So gibt es z. B. das *List PMM*, auf dem folgende Layout-Muster basieren: *Equal Horizontal/Vertical Distance Pattern*, *Align in a Row/Column Pattern* sowie *List Pattern*. Vergleicht man diesen Ansatz mit dem Konzept von DEViL3D, so sind die PMMs mit den visuellen Mustern vergleichbar, da sie ein komplexes Darstellungskonzept bestehend aus feingranularen Layoutentscheidungen definieren. Die Layout-Muster sind in gewisser Weise mit den von visuellen Mustern bereitgestellten Kontrollattributen verwandt. So gibt es im Listen-Muster in DEViL3D ein Kontrollattribut, welches den Abstand der Listenelemente festlegt. Ein Algorithmus, der die Nicht-Durchdringung von 3D-Objekten sicherstellt, kann ebenfalls durch ein Kontrollattribut des Mengen-Musters festgelegt werden.

### Constraintbasierte Ansätze

Das erste constraintbasierte visuelle System war Sutherland's *Sketchpad* [Sut64]. Ein mit Constraints arbeitendes System neueren Datums ist *Dunnart* [DMW09], welches von Dwyer und Marriot entwickelt wurde und einen Editor bereitstellt, mit dem Diagramme durch Angabe zahlreicher Constraints gelayoutet werden können.

Bei VL-Eli [Jun00] – dem Vorgänger von DEViL – wurde der Constraint-Solver *Parcon* [GLM+96] verwendet, um das Layout visueller Sprachen zu erstellen. Bei der Entwicklung von DEViL wurde allerdings davon Abstand genommen, da durch Weiterentwicklung der visuellen Muster das gleiche Ergebnis ohne einen Constraint-Solver, der zusätzlich aufwendig zu warten ist, erzielt werden konnte. In DEViL3D wird constraintbasiertes Layout nur punktuell als alternative Implementierung zur Sicherstellung der Durchdringungsfreiheit von Sprachkonstrukten sowie Andocken an ein Gitter oder an andere Objekte angewendet.

Das bereits in Abschnitt 4.10 vorgestellte Werkzeug 3DComposer [Chu99; CHM99], mit dem 3Dvixel als Bausteine für 3D-Sprachen oder 3D-Visualisierungen erstellt

werden können, ermöglicht die Erstellung binärer Constraints zwischen genau zwei 3Dvixeln. So können z. B. Constraints definiert werden, die feste Abstände zwischen 3Dvixeln sicherstellen und auch komplexere Darstellungen wie Kegelbäume realisieren.

KAPITEL 6

# Interaktion und Navigation

Dieses Kapitel behandelt Interaktions- und Navigationsaspekte der mit DEViL3D generierten 3D-Strukturediteren, die das Frontend einer 3D-Sprachimplementierung sind. Mit diesen Strukturediteren konstruieren Endanwender Diagramme einer dreidimensionalen Sprache im 3D-Raum. Daher müssen die Interaktions- und Navigationstechniken die sich daraus ergebenden Anforderungen erfüllen.

Die Interaktion mit 3D-Editoren ist verglichen mit 2D-Editoren deutlich komplexer, was sich u. a. an den sechs *Freiheitsgraden* (Translation und Rotation in allen drei Dimensionen und damit doppelt so viele wie im 2D-Raum) ausdrückt, die die Möglichkeiten der Bewegung im Raum beschreiben. Dies bedarf speziellen Techniken, die z. B. das Einfügen und Verschieben von Sprachkonstrukten auch an im 3D-Raum weiter entfernten Positionen ermöglichen. Betrachtet man Navigationsoperationen in 2D-Anwendungen, so fällt auf, dass diese überwiegend auf das Scrollen entlang der zwei Dimensionsachsen limitiert sind, da der Anwender stets mit einer zweidimensionalen Ebene interagiert, deren sichtbarer Ausschnitt auf dem Monitor höchstens verschoben werden kann. Handelt es sich hingegen um einen 3D-Editor, so wird der Anwender in gewisser Weise Teil der 3D-Szene, was zusätzlicher Techniken bedarf, damit sich der Anwender in der 3D-Szene nicht verliert, er aber trotzdem jede Position im Raum erreichen und das konstruierte Diagramm aus verschiedenen Blickwinkeln betrachten kann.

Für 3D-Anwendungen gibt es zahlreiche spezialisierte Ein- und Ausgabegeräte, die die Orientierung im 3D-Raum erleichtern sollen. Da das DEViL3D-System von mehreren Anwendern und Entwicklern genutzt wird, insbesondere von Studenten, die z. B. für ihre Abschlussarbeit eine 3D-Sprache spezifizieren, wurde entschieden, die generierten Editoren zunächst mit herkömmlichen Ein- und Ausgabegeräten bedienbar zu machen. Weiterhin sollen die Editoren auf den drei am weitesten

verbreiteten Betriebssystemen Linux, Mac OS X und Windows lauffähig sein, was durch Festlegung auf ein spezielles Ein- oder Ausgabegerät unter Umständen zu Einschränkungen aufgrund von Treiber-Inkompatibilitäten geführt hätte. Aus diesem Grund habe ich mich zunächst auf die klassische Maus und Tastatur als kleinsten gemeinsamen Nenner zur Interaktion mit dem Editor festgelegt. Im Laufe der Arbeit wurde allerdings noch Spezialhardware angeschafft, die es ermöglicht, die 3D-Szene im Struktureditor stereoskopisch darzustellen (siehe Abschnitt 6.8).

Die hier vorgestellten Interaktions- und Navigationstechniken sind nicht alle von Grund auf neu konzipiert worden, sondern sie basieren auf Techniken, die in etablierten 3D-Editoren (siehe Abschnitt 2.4) bereits erprobt sind. Dafür wurde analysiert, welche der Techniken auch für 3D-Spracheditoren relevant ist. Diese Techniken wurden dann generisch verfügbar gemacht, sodass sie für eine große Bandbreite von 3D-Sprachen anwendbar sind. Insgesamt ist das Ziel, dass die Funktionalität wie man sie von guten manuell entwickelten Editoren kennt, auch von den mit DEViL3D generierten Editoren geboten wird.

Einige der in diesem Kapitel präsentierten Interaktions- und Navigationstechniken wurden in Rahmen einer Masterarbeit[RR13] prototypisch implementiert. Zur Konzeption dieser Techniken und deren Kapselung in visuelle Muster wurde in den Papieren [WK14] und [WK15] berichtet.

Dieses Kapitel ist wie folgt strukturiert: Zunächst werde ich einen Überblick über 3D-Interaktionstechniken geben und sie klassifizieren. Die drei nachfolgenden Abschnitte folgen einem typischen Interaktionsablauf in einem 3D-Struktureditor: Einfügen, Selektieren und anschließend Manipulieren von Sprachkonstrukten. Anschließend folgt ein Abschnitt, der die 3D-Navigation kurz klassifiziert und sie in die zwei Bereiche Travelling und Wayfinding unterteilt. Die Abschnitte 6.6 und 6.7 widmen sich diesen Themen genauer. Der darauf folgende Abschnitt beschreibt stereoskopische Verfahren und deren Einbindung in DEViL3D-Editoren. In Abschnitt 6.9 gehe ich auf Gemeinsamkeiten und Unterschiede der hier beschriebenen Techniken mit jenen, wie sie in etablierten 3D-Editoren verwendet werden, ein. Das Kapitel endet mit der Vorstellung verwandter Arbeiten, die kurz vorgestellt und der eigenen Arbeit gegenübergestellt werden.

## 6.1 Übersicht und Klassifikation der 3D-Interaktion

An dieser Stelle werde ich einen groben Überblick über 3D-Interaktion im Allgemeinen geben. Dabei beziehe ich mich grundlegend auf Publikationen von Bowman et al. [BKLP04] und Dachselt [Dac04]. Dort wird die 3D-Interaktion in *Selektion* und *Manipulation* untergliedert. Dies und die weitere Untergliederung der einzelnen Interaktionstechniken ist in Abbildung 6.1 zu finden. Da im Hinblick auf den in

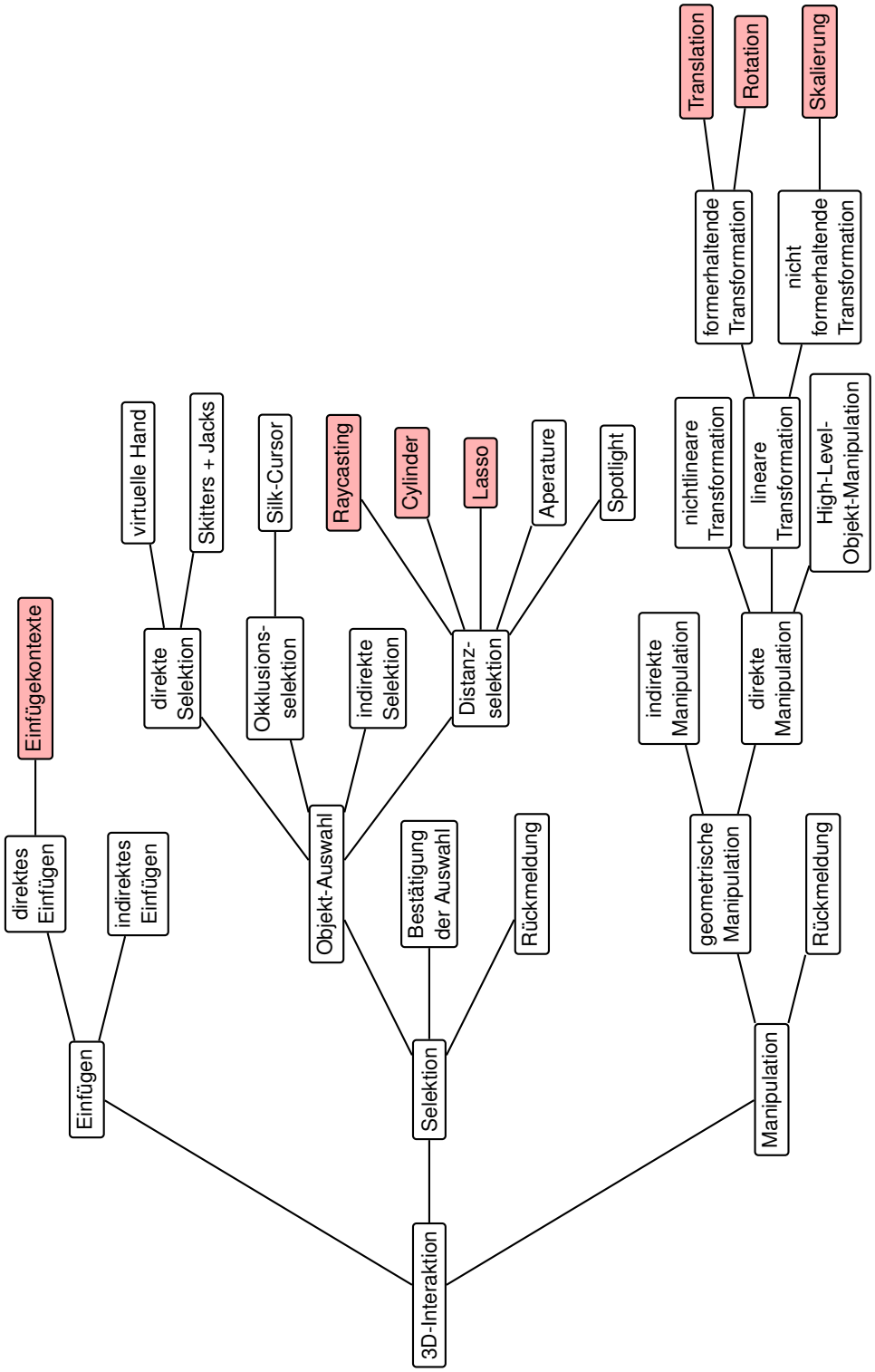



**Abbildung 6.1:** Klassifikation von 3D-Interaktionstechniken.

dieser Arbeit betrachteten Anwendungsfall – Editoren für dreidimensionale Sprachen – der Aspekt des *Einfügens* von Sprachkonstrukten in der Standard-Literatur fehlt, habe ich meine Klassifikation dahingehend erweitert. Abbildung 6.1 ist also die Vereinigung von aus der Literatur bekannten Klassifikationsaspekten und eigenen im Rahmen dieser Arbeit relevanten Aspekten.

An den Blättern der baumartigen Klassifikation in Abbildung 6.1 stehen konkrete Techniken oder *Metaphern*, mit denen die entsprechende Interaktion durchgeführt wird. Es gibt zahlreiche Metaphern für 3D-Interaktionen [PD10, S. 89ff.]. Ein Beispiel ist das *Raycasting*, welches beschreibt, wie ein Strahl in die 3D-Szene geschossen wird, um Objekte auszuwählen. Eine Metapher steht in unmittelbarem Zusammenhang mit der korrespondierenden Technik, die auch konkrete Hardware- und Softwarekomponenten mit einbezieht. Nicht alle der dort aufgeführten Techniken sind für 3D-Strukturediteren von Relevanz; sie sind hier lediglich mit aufgeführt, um auch den Anspruch einer allgemeinen Klassifikation für 3D-Interaktionstechniken zu erfüllen. Die für Strukturediteren relevanten Techniken und Metaphern sind rot eingefärbt. Auf einige der anderen werde ich im Abschnitt 6.10 über verwandte Arbeiten kurz eingehen.

Die Interaktion lässt sich generell dahingehend unterscheiden, ob sie *direkt* oder *indirekt* erfolgt. Direkt bedeutet, dass eine Operation unmittelbar an einem 3D-Objekt vorgenommen wird, es z. B. „angefasst“ wird, um es zu verschieben. Eine indirekte Interaktion liegt vor, wenn eine Änderung an einem Objekt über einen Mittler vorgenommen wird. Dies kann z. B. ein Dialogfenster sein, in das eine neue Position für das Objekt eingetragen wird. Diese Unterscheidung ist in allen drei Kategorien Einfügen, Selektion und Manipulation zu finden.

Beim Einfügen von Sprachkonstrukten spielen sogenannte *Einfügekontexte* eine zentrale Rolle, die es ermöglichen, Objekte nur an validen Positionen einzufügen. Die Selektion gliedert sich nach [BKLP04] in die drei Phasen Objekt-Auswahl, Bestätigung der Auswahl und Rückmeldung (vgl. Abbildung 6.1). Die Auswahl eines Objekts kann nach [Dac04] auf vier verschiedene Arten erfolgen. Für 3D-Strukturediteren ist hier aber primär die *Distanzselektion* und sekundär die *indirekte Selektion* von Belang. Die Manipulation besteht aus der eigentlichen geometrischen Manipulation und einer nachfolgenden Rückmeldung über die durchgeführte Manipulation. Die direkte lineare Transformation ist die relevanteste Manipulationsart, die entweder formerhaltend oder nicht formerhaltend ist. Die nachfolgenden drei Abschnitte folgen diesem Schema und beschäftigen sich mit dem Einfügen, der Selektion sowie der Manipulation von Sprachkonstrukten. Die hier nur kurz erwähnten Begrifflichkeiten der Klassifikation werden in den nachfolgenden Abschnitten genauer erläutert.

## 6.2 Einfügen von Sprachkonstrukten

Strukturediteren stellen sicher, dass Sprachkonstrukte strukturiert und an für sie vorgesehenen Stellen im Diagramm platziert werden können (vgl. Abschnitt 2.1). Wie schon bei von DEViL generierten Editoren für 2D-Sprachen gibt es zum Einfügen von Sprachkonstrukten in ein in der 3D-Szene dargestelltes Diagramm *Einfügekontexte*. Die Einfügekontexte markieren Positionen, an denen entsprechende Sprachkonstrukte eingefügt werden können. Bricht man diese Darstellung auf die Grammatik-Ebene einer Sprache hinunter, markieren Einfügekontexte Positionen von Nichtterminalen auf der rechten Seite einer Produktion. Eine Einfügeoperation würde dann die Ableitung des Nichtterminals an die entsprechende Position in der Produktion einfügen.

Diesem Prinzip folgen auch die Einfügekontexte in den 3D-Editoren. Sie sind mit Kontexten im von der abstrakten Struktur der Sprache definierten Strukturbaum assoziiert und repräsentieren diese in der 3D-Darstellung des Diagramms. Durch Betätigung des Einfügekontexts (wie dies genau geschieht, werde ich in den nächsten Abschnitten beschreiben) wird ein neues Sprachkonstrukt in den assoziierten Kontext des Strukturbaums eingefügt, woraufhin nach dem Model-View-Control-Prinzip die 3D-Darstellung neu berechnet und somit das neue Sprachkonstrukt auch dort dargestellt wird. Die Einfügekontexte sind in den visuellen Mustern (siehe Kapitel 4) gekapselt, die vom Sprachentwickler den Symbolen der aus der abstrakten Struktur generierten Grammatik zugeordnet werden. Dieses Vorgehen stellt sicher, dass sich der Sprachentwickler nicht um die Spezifikation der Einfügekontexte kümmern muss, sondern sie nach erfolgter Spezifikation der visuellen Repräsentation automatisch in die 3D-Szene eingefügt werden. Dort werden sie

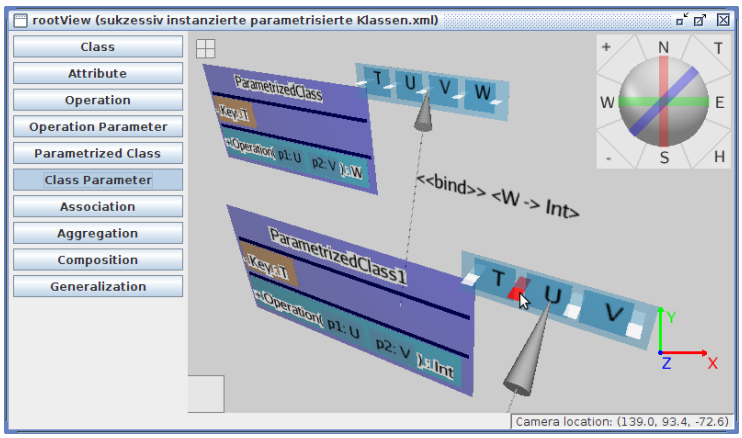
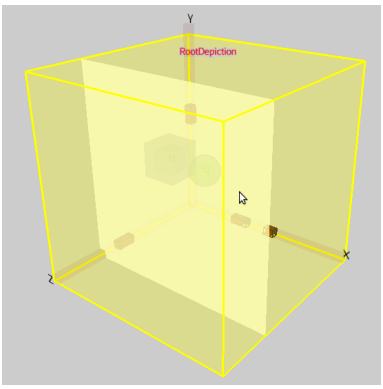
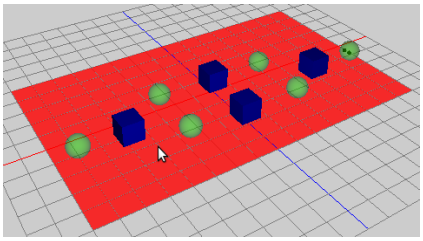


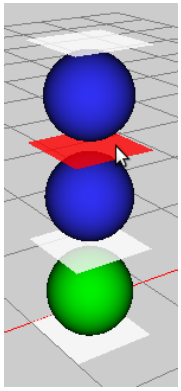

**Abbildung 6.2:** Zum Einfügen eines Sprachkonstrukts werden alle validen Einfügepositionen durch Einfügekontexte hervorgehoben.



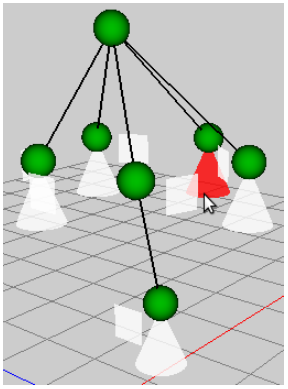

**(a)** Einfügekontext für Element einer 3D-Menge.



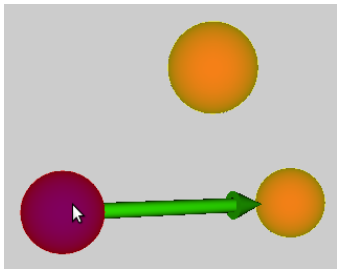
**(b)** Einfügekontext für auf einer Ebene angeordnete Elemente.



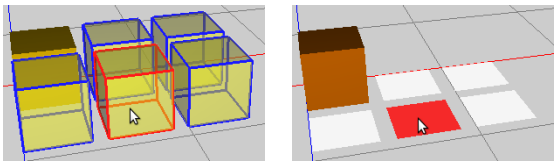
**(c)** Einfügekontext für Listenelemente.



**(d)** Einfügekontexte für Knoten in einen Kegelbaum.



**(e)** Einfügekontext für Verbindungen.



**(f)** Einfügekontexte für ein Containerelement.

**Abbildung 6.3:** Einfügekontexte für verschiedene visuelle Muster.

angezeigt, wenn der Editoranwender auf einen Einfüge-Button an der linken Seite der Sicht klickt (siehe Abbildung 6.2).

Die Beschaffenheit und Form eines Einfügekontexts variiert in Abhängigkeit des visuellen Musters und ist an dessen Repräsentationseigenschaften angepasst. Insgesamt gibt es drei verschiedene Grundformen (Quader, Ebene und Kegel) der Einfügekontexte, die je nach visuellem Muster in bestimmter Weise für das Einfügen der Sprachkonstrukte sorgen. Abbildung 6.3 zeigt Einfügekontexte für sechs visuelle Muster.

Der Einfügekontext, mit dem Sprachkonstrukte einfügt werden können, die als Elemente einer 3D-Menge repräsentiert werden, besteht aus einem transparenten Quader, der in seinem Inneren eine Ebene enthält (siehe Abbildung 6.3a). Diese Ebene kann entlang der $z$-Achse verschoben werden. Durch Klick auf die Ebene kann für das einzufügende Sprachkonstrukt eine dreidimensionale Position innerhalb eines vorgegebenen Bereichs bestimmt werden: Die $x$- und $y$-Koordinate ergibt sich aus den Bildschirmkoordinaten projiziert auf die Ebene innerhalb des Quaders und die $z$-Koordinate aus der Position dieser Ebene. Da der Benutzer die konkrete Einfügeposition eines Sprachkonstrukts selbst bestimmen kann, ist dies ein Einfügekontext mit Positionswahl (vgl. Abschnitt 2.6.1). Als Einfügekontext mit Positionswahl fungiert ebenfalls der ebenenförmige Einfügekontext (Abbildung 6.3b) für Elemente des Ebenen-Musters, bei dem die kontinuierliche Position des Sprachkonstrukts auf der Ebene bestimmt wird.

Zum Einfügen von Listenelementen werden ebenfalls Einfügekontexte in Form von Ebenen verwendet. Dabei sind die diskreten Positionen für Elemente der Liste durch Ebenen markiert (Abbildung 6.3c). Der Benutzer kann durch Anklicken der Ebene die diskrete Position eines neuen Elements innerhalb der Liste bestimmen. Auf gleiche Weise sind Einfügekontexte beim Matrix- und Quader-Muster für neue Zeilen, Spalten und Ebenen beschaffen. Hierbei handelt es sich um Einfügekontexte ohne Positionswahl.

Zum Einfügen neuer Knoten in einen Kegelbaum gibt es zwei Einfügekontexte (siehe Abbildung 6.3d): Um auf einer schon existierenden Baum-Ebene Knoten einzufügen, werden durch ebenförmige Einfügekontexte – wie bei der Liste – die Zwischenpositionen angezeigt. Eine neue Ebene wird durch einen kegelförmigen Einfügekontext hinzugefügt. Um Containerelemente einzufügen, kann zwischen einem quader- oder ebenenförmigen Einfügekontext ausgewählt werden (siehe Abbildung 6.3f). Beim Einfügen von Verbindungen fungieren die zu verbindenden Elemente als Einfügekontexte und werden gelb hervorgehoben. Sukzessives Klicken auf zwei dieser Elemente fügt zwischen diesen eine Verbindung ein (Abbildung 6.3e).

Tabelle 6.1 fasst die Eigenschaften der Einfügekontext für die einzelnen visuellen Muster zusammen. Darüber hinaus ist aufgeführt, ob der Einfügekontexte

| Vis. Muster | Einfügekontext(e) | Positionswahl | EK parametrisierbar |
|---|---|---|---|
| 3D-Menge | Quader mit Ebene | mit | Ausrichtung Einfügeebene |
| Ebene | Ebene | mit | ✗ |
| 1D-Menge | Ebene | mit | ✗ |
| Liste | Ebene | ohne | Dicke der EK-Ebene |
| Container | Quader \| Ebene | ohne | Wahl des EK |
| Kegelbaum | Kegel + Ebene | ohne | ✗ |
| Verbindung | zu verbindende Obj. als EK | ohne | ✗ |
| Matrix | Ebene | ohne | ✗ |
| Quader | Ebene | ohne | ✗ |

**Tabelle 6.1:** Eigenschaften der Einfügekontexte für die einzelnen Muster.

eines visuellen Musters parametrisiert werden kann. So können Sprachentwickler z. B. für Containerelemente zwischen zwei Einfügekontexten wählen (siehe Abbildung 6.3f) oder beim Einfügekontext für 3D-Mengenelemente festlegen, orthogonal zu welcher der drei Achsen die Einfügeebene verschiebbar ist.

## 6.3 Selektion

Nachdem ein Sprachkonstrukt eingefügt wurde, muss der Editornutzer die Möglichkeit haben, beliebige Sprachkonstrukte auszuwählen, um z. B. Operationen zur Manipulation des Sprachkonstrukts anzuwenden. Der Ablauf der Selektion besteht aus der Objekt-Auswahl und der nachfolgenden Bestätigung der Auswahl, die durch Anzeige einer Bounding-Box die ausgewählten Objekte markiert. Ist die Selektion von Objekten in einem zweidimensionalen Editor noch verhältnismäßig einfach, da die Position des Mauscursors direkt mit der Position von Objekten auf der Zeichenfläche verglichen werden kann, so müssen bei 3D-Editoren erweiterte Techniken angewendet werden. Dafür gibt es verschiedene Techniken, die sich unter dem Sammelbegriff Distanzselektion subsumieren. In diesem Abschnitt werde ich die Raycasting-Technik vorstellen, die die Grundlage für alle Selektionen im 3D-Raum bietet. Danach gehe ich auf die Selektion von Objekten in geschachtelten Strukturen ein bevor danach die Mehrfachselektion Thema ist.

### 6.3.1 Raycasting

Die Raycasting-Technik ermöglicht die Auswahl von Sprachkonstrukten aus der 3D-Szene mit einer herkömmlichen Maus, wie es aus 2D-Editoren bekannt ist. Anders als in 2D-Editoren befinden sich die Objekte nicht direkt auf der Monitor-

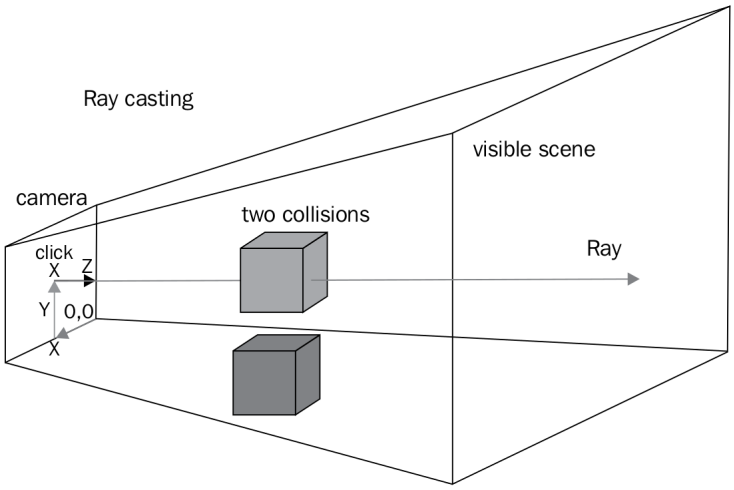



**Abbildung 6.4:** Die Raycasting-Technik; Quelle: [Kus13, S. 68].

Ebene, sondern an beliebigen Positionen im virtuellen 3D-Raum. Dies zeigt auch Abbildung 6.4, in der sich die Monitor-Ebene links befindet. Die sichtbare 3D-Szene, die sich ausgehend von der Monitor-Ebene erstreckt, enthält zwei Würfel, die entlang der $z$-Achse verschoben sind. Mit der von einer 2D-Maus zu bestimmenden 2D-Position kann ein solches Objekt nicht ohne weiteres ausgewählt werden. Es muss also ein Weg gefunden werden, eine 2D-Position auf der auf dem Monitor sichtbaren Projektionsebene der 3D-Szene in 3D-Koordinaten zu übersetzen.

Die Raycasting-Technik bedient sich dafür eines Strahls, der ausgehend vom Mauscursor auf der Projektionsebene orthogonal in die 3D-Szene „geschossen“ wird. Dieser Strahl liefert eine Liste von Kollisionen mit 3D-Objekten zurück. Die zur Implementierung der 3D-Szenen genutzte jMonkeyEngine unterstützt bereits diesen Ansatz, liefert allerdings in der Kollisionsliste sowohl den Ein- als auch den Austrittspunkt des Strahls in oder aus einem Objekt (siehe Abbildung 6.4). Diese beiden Kollisionspunkte wurden in den von DEViL3D generierten 3D-Editoren zu einer Kollision zusammengefasst, um damit genau ein Sprachkonstrukt auswählen zu können.

Die Raycasting-Technik wird nicht nur zur Selektion von Sprachkonstrukten eingesetzt, sondern kommt universell bei der Interaktion mit beliebigen Objekten in der 3D-Szene zum Einsatz. So auch bei der Benutzung der oben beschriebenen Einfügekontexte. Werden in einem 3D-Diagramm Sprachkonstrukte ineinander geschachtelt (siehe Abschnitt 5.3), basiert die Interaktion mit ihnen auch auf Raycasting, bedarf allerdings spezieller Anpassungen. Diesem Aspekt widmet sich der nachfolgende Abschnitt.

### 6.3.2 Selektion in geschachtelten Strukturen

Würde man das Raycasting-Verfahren ohne weitere Anpassungen verwenden, um Sprachkonstrukte zu selektieren, die in andere eingeschachtelt sind, könnte nur das äußere Objekt ausgewählt werden. Es wurden verschiedene weitere Ansätze entwickelt. Ein Ansatz zur Selektion eingeschachtelter Objekte beruht darauf, das äußere Objekt auszublenden und so das darin enthaltene Objekt freizugeben. Dies ist natürlich kein praktikabler Weg, da er erstens aufwendig ist und zweitens das Erscheinungsbild des 3D-Diagramms stark beeinflusst.

Der zweite Ansatz beruht auf der indirekten Auswahl eines Sprachkonstrukts über ein Kontextmenü. Klickt der Benutzer auf ein Sprachkonstrukt, in dessen Inneren sich weitere Objekte befinden, erscheint ein Kontextmenü, welches alle Sprachkonstrukte auflistet, die von dem ausgehenden Raycasting-Strahl getroffen wurden (siehe Abbildung 6.5). Die Auswahl erfolgt anschließend durch Anklicken eines der Kontextmenü-Einträge.

Die dritte zur Verfügung stehende Selektionstechnik wählt immer von den vom Strahl getroffenen Objekten dasjenige aus, welches sich am tiefsten innerhalb der hierarchisch geschachtelten Struktur befindet. Zwischen diesem und der indirekten Selektion per Kontextmenü kann in einem 3D-Editor jederzeit gewechselt werden. Diese Selektionstechniken sind in allen mit DEViL3D generierten Strukturediteren automatisch anwendbar und der Sprachentwickler kann festlegen, welche der Selektionstechniken in seinem Editor der Standard sein soll.

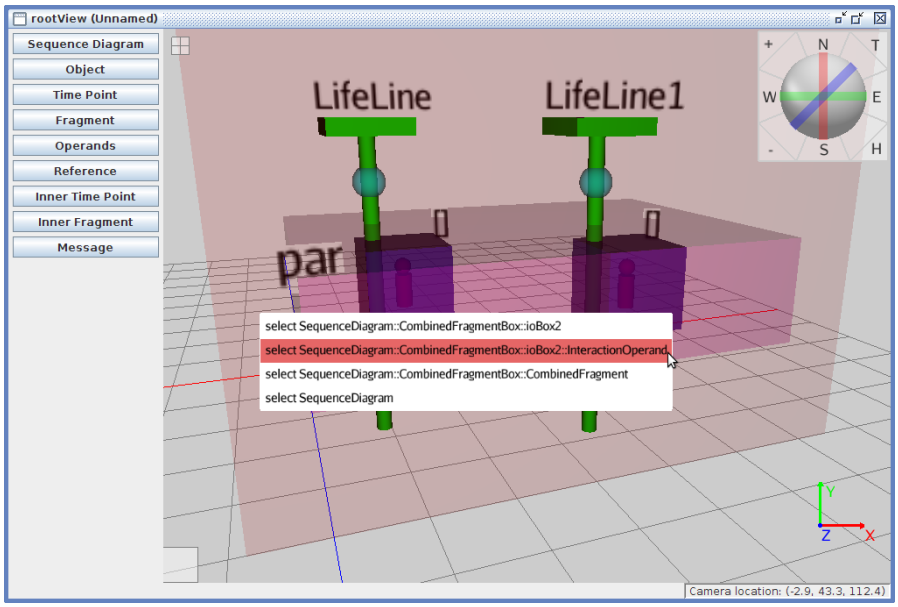


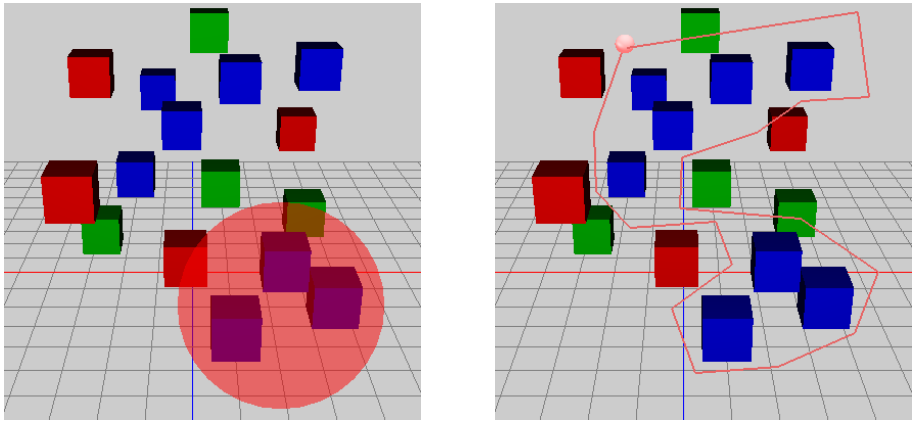

**Abbildung 6.5:** Indirekte Auswahl geschachtelter Sprachkonstrukte über das Kontextmenü.

### 6.3.3 Mehrfachselektion

Der Schwerpunkt der bisher vorgestellten Selektionsoperationen ist das Auswählen einzelner Sprachkonstrukte. Die mit DEViL3D generierten Editoren erlauben allerdings auch die gleichzeitige Auswahl mehrerer Sprachkonstrukte, was z. B. notwendig ist, um eine Gruppe von Objekten gleichzeitig zu bearbeiten, z. B. zu verschieben. Die einfachste und auch aus 2D-Editoren bekannte Vorgehensweise ist die sukzessive Auswahl einzelner Objekte bei gedrückt gehaltener Kontrolltaste. Die generierten Editoren bieten allerdings unter Zuhilfenahme der *Zylinder-* und *Lasso-Metapher* zwei intuitivere und direktere Methoden an.

Bei Auswahl der Zylinder-Technik erscheint eine transparente rote Zylindergrundfläche (siehe Abbildung 6.6a) auf der 2D-Projektionsfläche der 3D-Szene. Sie kann vom Editornutzer auf dieser Ebene beliebig verschoben und in seiner Größe angepasst werden. Durch Betätigen der linken Maustaste werden alle Objekte ausgewählt, die vollständig von der Zylinder-Grundfläche umschlossen sind. Dafür expandiert die Zylinder-Grundfläche, vorstellbar als ein Bündel von Strahlen wie beim Raycasting, in die 3D-Szene. Es werden alle Objekte ausgewählt, die sich dann innerhalb des daraus entstehenden Zylindervolumens befinden, also vollständig von der Mantelfläche des Zylinders umschlossen werden.

Auf diesem Ansatz baut die Lasso-Technik auf und erlaubt dem Anwender, ein beliebiges Polygon auf der Projektionsfläche zu erstellen, welches dann ebenfalls in die 3D-Szene expandiert. Die Lasso-Technik ist evtl. etwas flexibler einsetzbar, da ein benutzerdefiniertes Polygon erstellt werden kann. Abschnitt 7.3.7 stellt ein Experiment vor, welches die beiden Techniken anhand von zwei Benutzergruppen



**(a)** Zylinder-Metapher. **(b)** Lasso-Metapher.

**Abbildung 6.6:** Zwei Wege zur simultanen Auswahl mehrerer Objekte.

miteinander vergleicht. Auch diese beiden Techniken sind in allen mit DEViL3D generierten Editoren ohne Zutun des Sprachspezifizierers anwendbar.

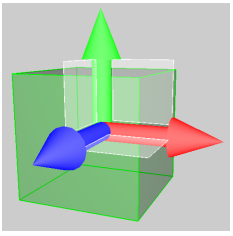
## 6.4 Manipulation von Sprachkonstrukten

Die Manipulation eines Sprachkonstrukts hat als Vorbedingung deren Selektion und wird allgemein in direkte und indirekte Manipulation untergliedert. Die direkte Manipulation ist hier die deutlich interessantere, da sie unter Zuhilfenahme bestimmter Werkzeuge direkt in der 3D-Szene stattfindet. Die indirekte Manipulation ändert Eigenschaften eines Sprachkonstrukts über Menüs oder spezielle Dialoge.

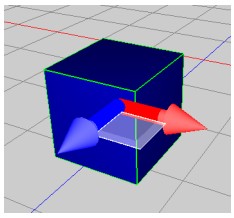
### 6.4.1 Direkte lineare Transformation

Die direkte lineare Transformation von Sprachkonstrukten unterteilt sich in formerhaltende und nicht formerhaltende Transformation (vgl. Abbildung 6.1). Eine formerhaltende Transformation ist mit der *Translation*, also Verschiebung, und der *Rotation* eines Sprachkonstrukts gegeben. Die *Skalierung* hingegen ist eine nicht formerhaltende Transformation.

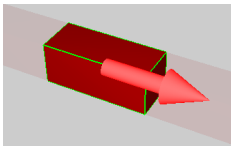
Ein mit DEViL3D generierter Editor stellt sogenannte *Widgets* bereit, die an Sprachkonstrukte angehängt werden können, um eine der drei Transformations-Aufgaben zu erledigen. Die Form und Funktionsweise der Widgets variiert in Abhängigkeit des visuellen Musters des zu transformierenden Sprachkonstrukts. Abbildung 6.7 zeigt Translations-Widgets für Elemente der drei verschiedenen Mengen-Muster. Gemäß der Layoutfreiheiten der einzelnen Muster erlauben die Widgets eine Verschiebung entlang drei, zwei oder einer Dimension. Bei dem Widget für 3D-Mengen- und Ebenenelemente gibt es jeweils zwischen zwei Pfeilen noch kleine Ebenen, um das Sprachkonstrukt gleichzeitig entlang zweier Dimensionen kontinuierlich zu verschieben.



**(a)** Entlang dreier Dimensionen verschiebbar.



**(b)** Entlang zweier Dimensionen verschiebbar.



**(c)** Entlang einer Dimension verschiebbar.

**Abbildung 6.7:** Translations-Widgets für die Elemente der drei verschiedenen Mengen-Muster.

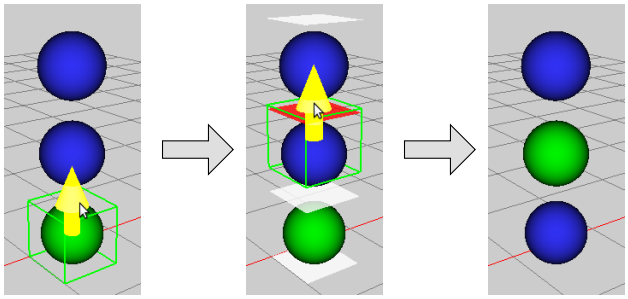


**Abbildung 6.8:** Translation eines Listenelements.

Das Widget zur Transformation von Listenelementen besteht, ähnlich wie das Widget für 1D-Mengenelemente, aus einem Pfeil, der sich in die Richtung erstreckt, in die auch die Liste wächst. Allerdings kann das Sprachkonstrukt bei Betätigung des Widgets nur an diskrete Positionen zwischen andere Listenelemente verschoben werden und nicht an beliebige kontinuierliche Positionen wie bei Mengenelementen. Um diese Zwischenpositionen auszuwählen, werden die Einfügekontexte der Liste angezeigt, wie in Abbildung 6.8 dargestellt. Das zu verschiebende Objekt kann dann auf dem rot hervorgehobenen Einfügekontext losgelassen werden, um es an diese Position zu verschieben.

Analog zu den Translations-Widgets bieten die Editoren weitere Widgets zur Rotation und Skalierung an. Das Widget zum Skalieren ist fast identisch mit jenem zur Translation. Ein Objekt an welches das Skalierungs-Widget angehängt ist, kann ebenfalls separat entlang einer Dimension oder gleichzeitig entlang zweier Dimensionen skaliert werden. Das Widget zur Rotation besteht aus drei Tori, die um das zu rotierende Objekt angeordnet sind. Die Rotation damit kann allerdings nur um eine Dimension zur gleichen Zeit stattfinden.

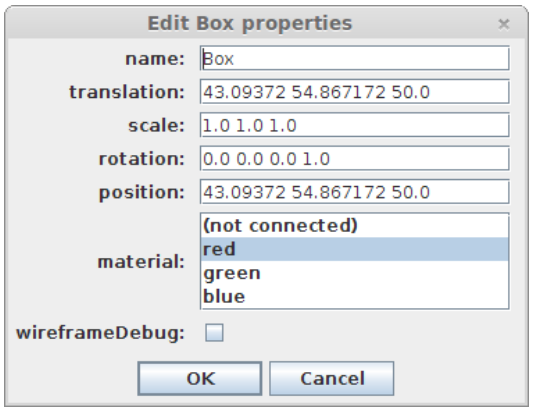



**Abbildung 6.9:** Indirekte Manipulation mittels Eigenschaftendialog.

### 6.4.2 Indirekte Manipulation

Indirekte Manipulation von Sprachkonstrukten erfolgt nicht unmittelbar in der 3D-Sicht, sondern mit Mitteln der 2D-GUI-Elemente des Editors. So können über das Kontextmenü eines Sprachkonstrukts Objekte gelöscht, kopiert oder eingefügt werden. Außerdem steht in jedem 3D-Spracheditor automatisch ein Eigenschaftendialog zur Verfügung, der alle editierbaren Attribute des Sprachkonstrukts umfasst (siehe Abbildung 6.9). Dort können neben sprachspezifischen Attributen auch die Position des Objekts in der 3D-Szene verändert werden. Sprachentwickler können durch Angabe einer zusätzlichen Spezifikation die Eigenschaftendialoge an kontextspezifische Anforderungen anpassen.

## 6.5 Übersicht und Klassifikation der 3D-Navigation

Die Navigation in einer 3D-Szene ist eine zentrale Komponente, die es ermöglicht alle relevanten Bereiche der Szene zu erreichen und von verschiedenen Positionen

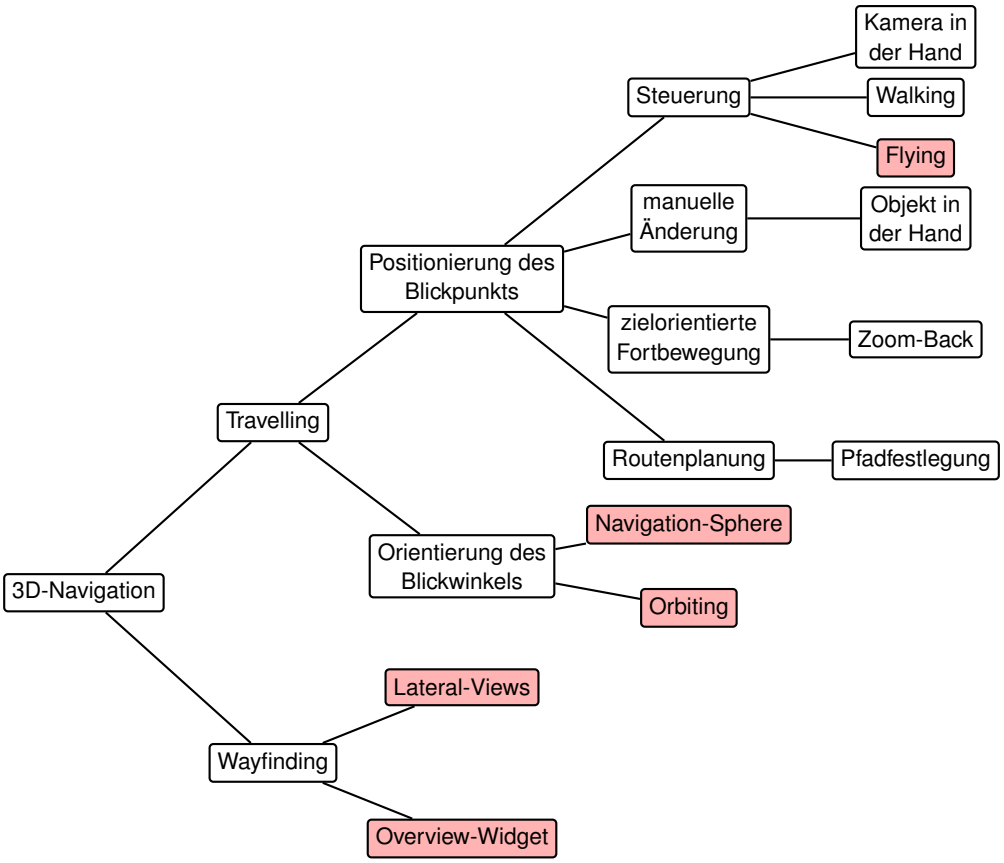



**Abbildung 6.10:** Klassifikation von 3D-Navigationstechniken.

aus zu betrachten. Diese Aufgabe bedarf besonderer Unterstützung durch den 3D-Editor, da der Benutzer des Editors selbst Teil der 3D-Szene ist, wohingegen er bei der Benutzung von 2D-Editoren aus der dritten Dimension heraus agiert und mit Objekten und Komponenten auf einer ebenen Fläche interagiert, die höchstens entlang zweier Dimensionen verschoben werden kann.

Bowman et al. [BKLP04] unterteilen die Navigation in 3D-Anwendungen in *Travelling*- und *Wayfinding*-Komponenten (siehe Abbildung 6.10). Travelling ist dabei die Bewegungskomponente der Navigation und umfasst die Kontrolle über die Blickrichtung eines Anwenders in einer 3D-Szene. Wayfinding hingegen ist eine kognitive Komponente des Anwenders, die diesen unterstützt, ein mentales Modell der 3D-Szene zu erstellen und bei der Orientierung darin unterstützende Hilfsmittel anbietet.

Die Travelling-Komponente untergliedert sich wiederum in die *Positionierung des Blickpunkts* und die *Orientierung des Blickwinkels* [BKLP04]. Erstere kümmert sich um Positionsänderungen der Kamera im 3D-Raum, wohingegen letztere die Kamera schwenkt, ohne aktiv ihre Position zu verändern. Das Ändern der Kamera-Position kann auf verschiedene Weise erfolgen: als bewusste Steuerung oder als manuelle Änderung, wie bei der *Objekt in der Hand* Metapher, bei der die Position eines Objekts fest ist und sich die Position relativ zu diesem ändert. Eine zielorientierte Fortbewegung unterstützt das Zoomen hin zu einem Objekt und bei der Routenplanung werden feste Pfade festgelegt, entlang derer die 3D-Szene anschließend immer wieder durchlaufen werden kann.

Die folgenden zwei Abschnitte gehen genauer auf die beiden Komponenten Travelling und Wayfinding ein und beschreiben jeweils die Techniken, die mit DEViL3D generierte Editoren dafür bereitstellen.

## 6.6 Travelling

Die zentrale Komponente der Navigation ist das Travelling, welches Bewegungseigenschaften wie die Festlegung des Blickwinkels auf die 3D-Szene festlegt. Bowman et al. unterteilen die Aufgabe des Travelling [BKH97] weiter in *Richtung/Zielauswahl*, *Geschwindigkeit/Beschleunigung* und *Eingabebedingungen*. Der erste Aspekt umfasst die Methode, mit der der Benutzer die Richtung seiner Bewegung festlegt, z. B. mit einer 2D-Maus als Eingabe oder auch mittels *Gaze-directed steering* in Virtual-Reality Umgebungen.[1] Der zweite Aspekt umfasst die Möglichkeiten, mit denen der Anwender die Geschwindigkeit seiner Fortbewegung festlegen kann.

[1] Bei Verwendung der Travelling-Technik *Gaze-directed steering* in Virtual-Reality Umgebungen wird die Position des Kopfs eines Anwenders verwendet, um die Richtung der Fortbewegung zu bestimmen.

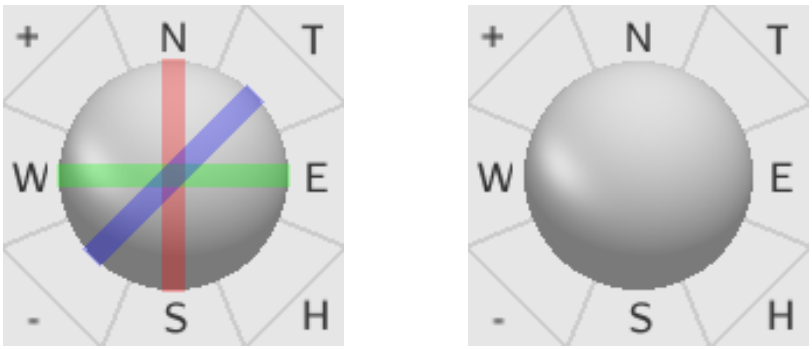
Der letzte Aspekt beschreibt Bedingungen, mit denen die Navigation startet, endet oder ob währenddessen kontinuierlich ein Eingabegerät betätigt werden muss.

Im Folgenden werde ich zwei verschiedene Wege beschreiben, wie die Kamera in der 3D-Szene gesteuert werden kann. Danach werde ich die *Orbiting*-Technik vorstellen.

### 6.6.1 Steuerung der Kamera

Jede 3D-Szene eines 3D-Spracheditors umfasst eine Kamera mit *First-Person-View*. Der Anwender kann die Position der Kamera im Raum festlegen sowie zusätzlich den Blickwinkel der Kamera anpassen. Dies erfolgt ähnlich wie bei vielen Computerspielen: Die Position der Kamera wird mit ausgewählten Tasten der Tastatur festgelegt und der Blickwinkel kann parallel dazu durch Bewegen der Maus mit gedrückter Maustaste verändert werden.

Eine weitere Möglichkeit zur Steuerung der Kamera stellt die in zwei verschiedenen Modi benutzbare *Navigation-Sphere* dar (siehe Abbildung 6.11). Diese besteht jeweils aus der eigentlichen Navigation-Sphere in der Mitte und darum angeordneten Segmenten. Mit der Kugel in der Mitte wird der Blickwinkel der Kamera verändert und um einen festgelegten Punkt rotiert. Angestoßen wird die Rotation durch Klicken auf die Kugel und bei gedrückt gehaltener Maustaste wird die Rotation fortgesetzt. Im *achsenbasierten Modus* ist die Rotation allerdings auf die drei Dimensionsachsen reduziert, sodass die Kamera nur um diese rotiert werden kann. Im *freien Modus* sind beliebige Rotationen möglich. Die Rotation wird bei Betätigung der Navigation-Sphere sofort durch Änderung des Blickwinkels auf die Szene sichtbar. Der Punkt um den rotiert wird, ist zu Beginn der Nullpunkt der Szene, es kann aber durch Auswahl im Kontextmenü jedes beliebige Sprachkonstrukt als Mittelpunkt der Rotation bestimmt werden.




**(a)** Achsenbasierter Modus. **(b)** Freier Modus.

**Abbildung 6.11:** Die zwei Modi der Navigation-Sphere.

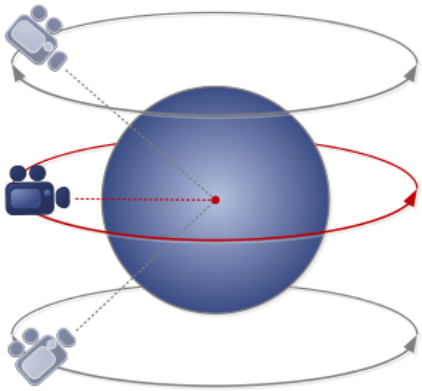


**Abbildung 6.12:** Bewegung der Kamera auf der Umlaufbahn um ein ausgewähltes Objekt mit der Orbiting-Technik; Quelle: [RR13, S. 109].

Die Segmente rings um die Navigation-Sphere sind mit Himmelsrichtungen beschriftet die nach dem Anklicken die Szene aus dieser Position zeigen. Das mit T (für „top“) beschriftete Segment zeigt die 3D-Szene von oben. Weiterhin kann der Anwender beliebige durch Bedienung der Navigation-Sphere eingenommene Positionen abspeichern und über das mit H beschriftete Segment anschließend wieder laden. An der linken Seite sind zwei Segmente zum Zoomen platziert.

### 6.6.2 Orbiting

Ähnlich wie die Navigation-Sphere gehört auch die Orbiting-Technik zu den Travelling-Techniken, die die Orientierung des Blickwinkels verändern. Die Orbiting-Technik erlaubt die Betrachtung eines ausgewählten *Point-of-Interest* (POI) aus verschiedenen Blickwinkeln (Abbildung 6.12 zeigt dies schematisch). Zwischen dieser Technik und der „normalen“ Navigation kann im Editorfenster gewählt werden. Ist das Orbiting aktiviert, kann ein Sprachkonstrukt als POI ausgewählt werden, woraufhin die Kamera zu diesem Sprachkonstrukt teleportiert wird. Bei gedrückt gehaltener rechter Maustaste umkreist die Kamera dann das POI auf einer vom Benutzer zu beeinflussenden Umlaufbahn. Diese kann vom Benutzer durch Bewegen der Maus in beliebige Richtungen verändert werden.

## 6.7 Wayfinding

Wayfinding-Techniken sollen den Anwender bei der Orientierung im 3D-Raum helfen und ihn unterstützen ein mentales Modell von der 3D-Szene aufzubauen, welches ihm ermöglicht sich schneller in der Szene zurechtzufinden. Bowman et al. unterscheiden *benutzerorientierte* und *umgebungsorientierte* Wayfindingtechniken. Die benutzerorientierten Techniken umfassen Bewegungsreize, die z. B. dabei

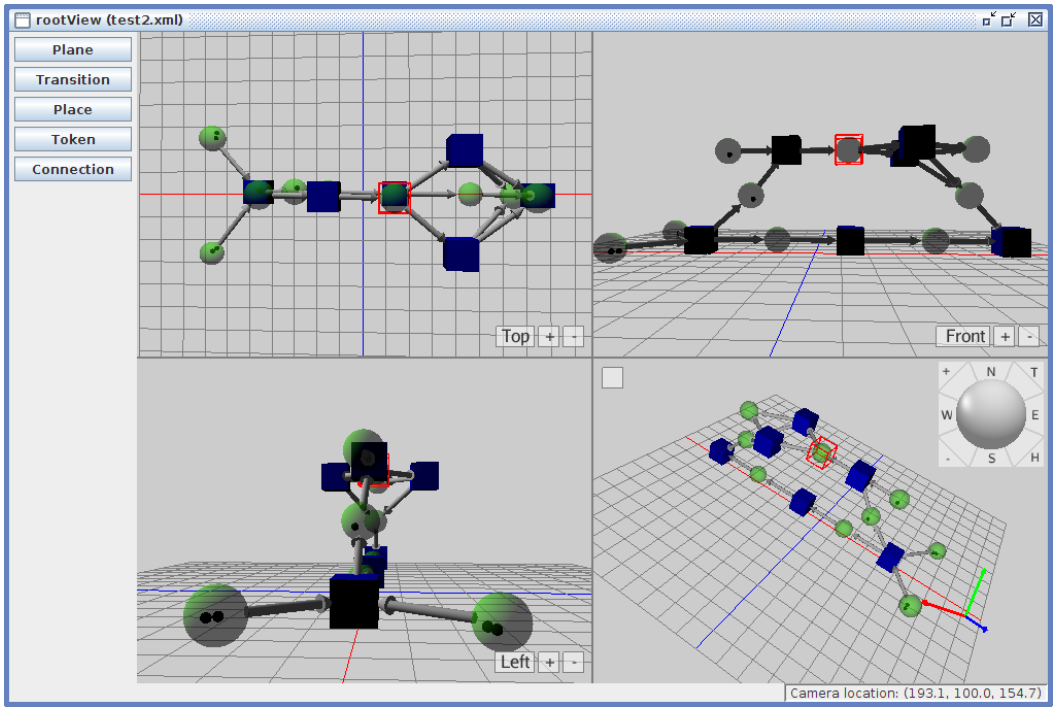



**Abbildung 6.13:** Lateral-Views stellen die 3D-Szene zusätzlich aus drei weiteren Perspektiven dar.

helfen, die Distanz zu Objekten besser einzuschätzen, und Suchstrategien, mit denen nach festgelegten Schemata Suchen durchgeführt werden. Die umgebungsorientierten Techniken umfassen bestimmte Hilfsmittel, wie z. B. Übersichtskarten der 3D-Szene. Die mit DEViL3D generierten Editoren stellen mit den *Lateral-Views* und dem *Überblicks-Widget* zwei Wayfindingtechniken bereit, die als umgebungsorientiert einzuordnen sind.

### 6.7.1 Lateral-Views

In manchen Situationen ist es wünschenswert die 3D-Szene gleichzeitig aus mehreren Blickwinkeln zu betrachten, z. B. weil die Positionierung eines Sprachkonstrukts besonders exakt erfolgen muss und somit ein zusätzlicher Blickwinkel von Nutzen ist. Die sogenannten Lateral-Views unterstützen den Benutzer bei solchen Aufgaben. Wie Abbildung 6.13 zeigt, umfassen die Lateral-Views vier verschiedene Sichten: die 3D-Sicht, in der wie gewohnt navigiert werden kann, und drei weitere Sichten, die die Szene orthogonal zu den drei Koordinatenachsen darstellen. Die zusätzlichen drei Sichten dienen dabei nicht nur der Darstellung, sondern sie können auch zur Interaktion mit dem 3D-Diagramm genutzt werden. Die ortho-

gonalen Sichten können in jedem Editor ohne zusätzlichen Spezifikationsaufwand hinzugeschaltet werden.

Vergleichbare Ansichten sind aus vielen etablierten 3D-Editoren – insbesondere CAD-Anwendungen – wohlbekannt und ein wertvolles Hilfsmittel bei der Erstellung von Konstruktionszeichnungen. Ein einheitlicher Name für diese Sichten hat sich in allerdings nicht etabliert. In *Maya*[2] wird z. B. von einem *four-panel layout* gesprochen, wohingegen bei 3ds Max Darstellungseigenschaften der *Viewports* vom Anwender umdefiniert werden können.

### 6.7.2 Überblicks-Widget

Das gleiche gilt für das sogenannte *Überblicks-Widget*, welches in der linken unteren Ecke jeder 3D-Sicht aktiviert werden kann. Wie in Abbildung 6.14 dargestellt, gibt es dem Anwender einen größeren Überblick über die Szene und ist hilfreich, wenn er auf ein Detail fokussiert, das große Ganze aber nicht aus dem Blick verlieren möchte. In diesem Überblicks-Widget wird der Ausschnitt der Szene, der in der Hauptsicht sichtbar ist, durch einen hellblauen Rahmen markiert. In der Ecke des Widgets sind zwei Buttons angeordnet mit denen der im Übersichts-Widget dargestellte Ausschnitt angepasst werden kann.

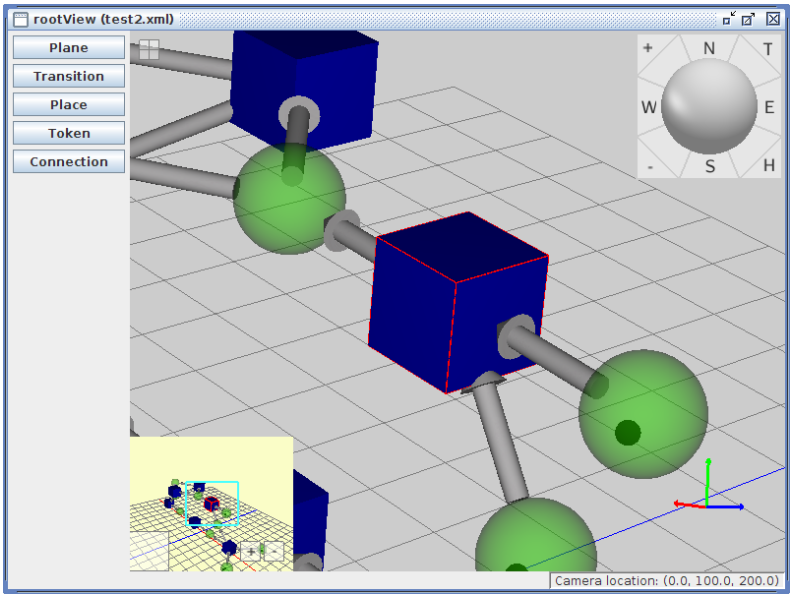



**Abbildung 6.14:** Überblicks-Widget in der linken unteren Ecke.

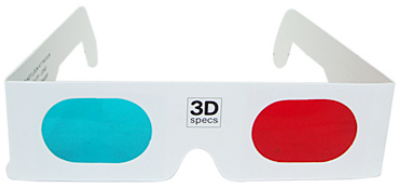
[2] `http://www.autodesk.de/products/maya/overview`

## 6.8 Darstellung der 3D-Szene mit stereoskopischen Verfahren

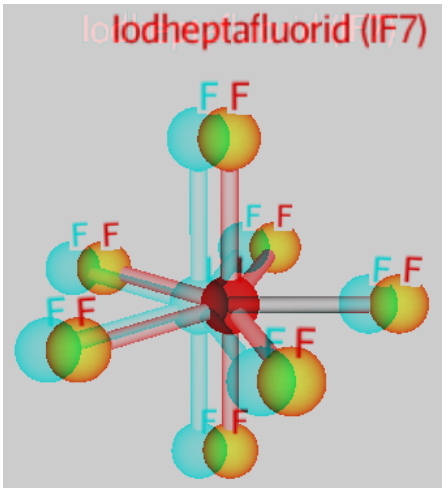
Werden zur Anzeige von mit DEViL3D generierten Strukturediteren herkömmliche Computermonitore genutzt, wird die 3D-Sicht auf die 2D-Darstellungsfläche des Monitors projiziert. Man spricht hierbei auch von einer *monoskopischen* Darstellung[3], die keinen Tiefeneindruck vermittelt. Der von einer 3D-Szene implizierte Tiefeneindruck kann durch *stereoskopische* Darstellungsverfahren[4] besser vermittelt werden. Damit auch die DEViL3D-Editoren diesen Tiefeneindruck vermitteln, wurden mit dem *Anaglyph-3D* und *Shutter-3D* zwei stereoskopische Verfahren implementiert. Die folgenden beiden Abschnitte stellen die Arbeitsweise dieser Verfahren kurz vor; danach zeige ich auf, wie sie in die generierten Editoren eingebunden wurden.

### 6.8.1 Anaglyph-3D

Anaglyph-3D (vgl. [Röd07]) ist das älteste Verfahren zur Tiefenbetrachtung stereoskopischer Bilder und wurde 1853 von Wilhelm Rollmann entwickelt. Bei dem Verfahren wird ein Farbanaglyphenbild durch eine spezielle Anaglyphenbrille (siehe Abbildung 6.15a) betrachtet. Die Gläser oder Farbfolien der Brille sind mit Komplementärfarben eingefärbt; das eine Glas ist rot eingefärbt, das andere meist




**(a)** Anaglyphenbrille.




**(b)** Anaglyph-3D-Darstellung im Moleküleditor.

**Abbildung 6.15:** Anaglyph-3D-Verfahren.

[3]monoskopisch: von griechisch *mónos*, „einzig“ / *skopein*, „sehen“.
[4]stereoskopisch: von griechisch *stereo*, „räumlich“ / *skopein*, „sehen“.

blau (oder auch grün). Das Anaglyph-3D-Verfahren stellt die kostengünstigste Möglichkeit zur Darstellung stereoskopischer Bilder dar.

Durch die verschiedenfarbigen Brillengläser entstehen zwei sich überlagernde Teilbilder. Beim Ansehen eines Anaglyphenbilds löscht der Rotfilter das rote Filmbild aus, sodass das grüne Bild schwarz dargestellt wird. Umgekehrt löscht der Grünfilter das grüne Farbbild, sodass dieses rot-schwarz erscheint. Jedes Auge sieht eines der beiden Bilder, die dann im Gehirn des Betrachters zu einem Raumbild verschmelzen.

### 6.8.2 Shutter-3D

Das Shutter-3D-Verfahren (vgl. [Röd07]) basiert – seinem Namen entsprechend englisch für Rollladen – darauf, dass ein 3D-Bild durch eine spezielle *Shutter-Brille*[5] betrachtet wird, deren Gläser abwechselnd auf durchlässig und undurchlässig geschaltet werden können. Das grundlegende Prinzip wurde bereits 1922 von Laurens Hammond prototypisch entwickelt; moderne Shutter-3D-Systeme gehen auf ein von Lenny Lipton gehaltenes Patent aus dem Jahr 1985 zurück. Um ein Shutter-3D-System zu betreiben, ist spezielle Hardware erforderlich: neben einer 3D-Brille auch ein 3D-Monitor.

Mit hoher Frequenz wird auf dem Monitor abwechselnd das Bild für das linke und für das rechte Auge angezeigt. Die beiden Bilder unterscheiden sich nur

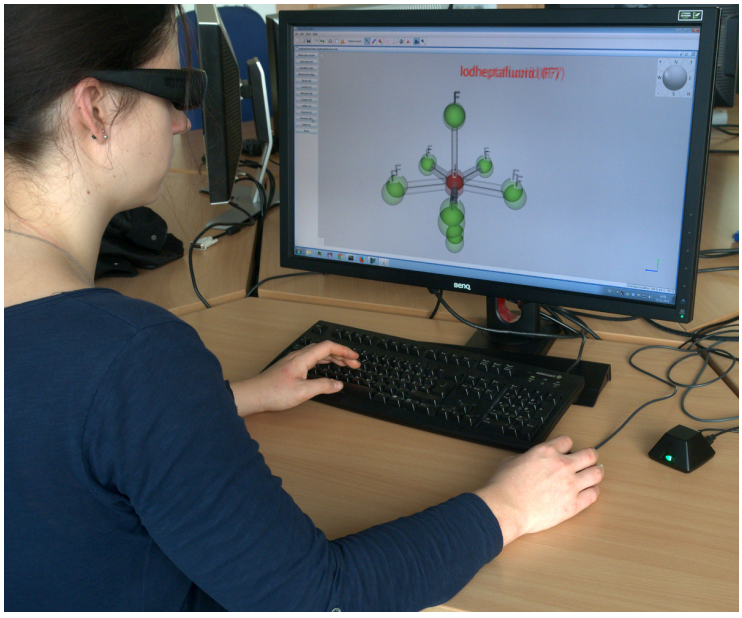


**Abbildung 6.16:** Anwenderin mit Shutter-Brille vor dem Moleküleditor.

[5] genauer engl.: *liquid crystal shutter glasses*

dahingehend, dass sie leicht versetzt dargestellt werden. Synchron zu den sich abwechselnden Bildern muss die 3D-Brille so geschaltet werden, dass beim Anzeigen des linken Bilds auf dem Monitor das linke Brillenglas durchlässig und das rechte undurchlässig ist (und umgekehrt). Wie beim Anaglyph-3D-Verfahren entsteht im Gehirn des Betrachters daraus ein 3D-Bild, welches sich aus den zwei Bildern ergibt, die für jedes Auge leicht unterschiedlich sind. Das resultierende Bild, welches der Anwender durch die Brillen wahrnimmt, suggeriert einen Tiefeneffekt sowohl vor als auch hinter dem 3D-Monitor. Abbildung 6.16 zeigt eine Anwenderin vor dem Moleküleditor. Da jedem Auge nur jedes zweite Bild angezeigt wird, halbiert sich die Bildwiederholfrequenz pro Auge. Um Flimmern zu verhindern, muss der verwendete Monitor eine möglichst hohe Bildwiederholfrequenz aufweisen. Üblich ist die Verwendung von Monitoren mit 120 Hertz, bei denen sich dann das Bild pro Sekunde und Auge 60-mal ändert.

### 6.8.3 Einbindung in die von DEViL3D generierten Editoren

Alle mit DEViL3D generierten Strukturediteren unterstützen neben der herkömmlichen monoskopischen Darstellung auch die zwei zuvor beschriebenen stereoskopischen Verfahren. Shutter-3D allerdings nur, wenn die notwendige Hardware am Computer vorhanden ist. Der Editornutzer kann das zu verwendende Darstellungsverfahren aus dem Menü jedes mit DEViL3D generierten Editors auswählen (siehe Abbildung 6.17). Der Entwickler einer Sprache hat mit stereoskopischen Verfahren keinen zusätzlichen Spezifikationsaufwand, er braucht lediglich in einer Konfigurationsdatei festzulegen, mit welchem Darstellungsverfahren der von ihm spezifizierte Spracheditor initial starten soll.

Zur Realisierung der beiden Verfahren wurden zwei Renderer zur Darstellung des Anaglyph-3D- und Shutter-3D-Verfahrens implementiert. Beide Renderer basieren darauf, dass die 3D-Szene von zwei leicht versetzten Kameras betrachtet wird, deren Bilder anschließend zusammengeführt werden. In beiden Bildern werden 3D-Objekte mit gleicher Breite und Höhe dargestellt; der Tiefeneindruck der

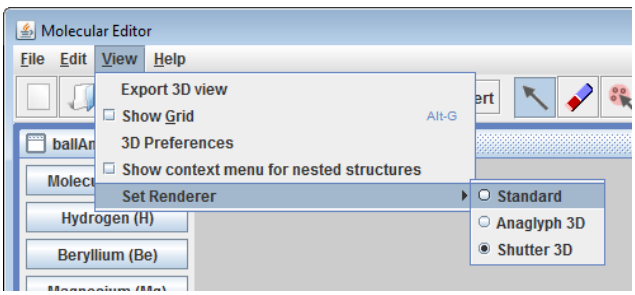



**Abbildung 6.17:** Der Renderer der 3D-Sicht kann dynamisch getauscht werden.

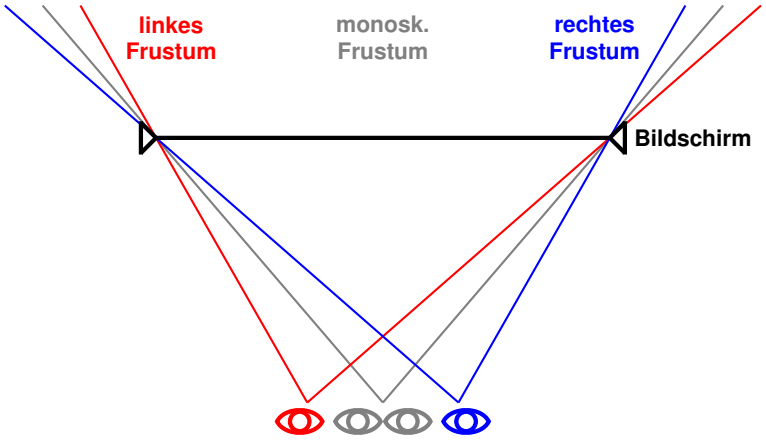



**Abbildung 6.18:** Jedes Auge sieht einen horizontal verschobenen Ausschnitt der Szene.

Objekte ergibt sich aus dem Versatz der Bilder fürs linke und rechte Auge. In der 3D-Computergrafik wird der Ausschnitt eines Raums, der auf einem Bildschirm sichtbar ist, als *View Frustum* bezeichnet. Bei der monoskopischen Darstellung sehen beide Augen das gleiche Frustum (siehe Abbildung 6.18). Im Fall einer stereoskopischen Darstellung ist das Frustum der beiden Augen leicht versetzt. Dieser Versatz ist dafür verantwortlich, dass das menschliche Gehirn eine Tiefenwahrnehmung der 3D-Szene erhält.

Der Programmcode für die Renderer gehört zu dem statischen Teil des Editorrahmens, der unabhängig von Spezifikationen des Sprachentwerfers in jedem generierten Editor vorhanden ist. Die Realisierung war also nur mit einmaligem Aufwand verbunden und kann in allen generierten Editoren genutzt werden. Die Umsetzung des Shutter-3D-Verfahrens war nicht direkt mit Mitteln der jMonkeyEngine möglich, sodass dies direkt mit OpenGL umgesetzt wurde.

Um 3D-Editoren in stereoskopischer Shutter-3D-Darstellung zu betrachten, wurde ein Computer mit spezieller Hardwareausstattung angeschafft. Neben einem 3D-Monitor beinhaltet dies besondere Komponenten wie eine *NVIDIA Quadro*[6] Grafikkarte und *NVIDIA 3D Vision Pro*[7]. Letzteres umfasst 3D-Brillen und eine Funksendeeinheit zum Anschluss an die Grafikkarte. Diese spezielle Hardware ist nur unter Windows lauffähig, da für andere Betriebssysteme keine entsprechenden Treiber vorhanden sind.

[6] `http://www.nvidia.de/object/product-quadro-4000-de.html`
[7] `http://www.nvidia.de/object/3d-vision-professional-users-de.html`

## 6.9 Diskussion und Vergleich mit Techniken aus etablierten 3D-Editoren

In diesem Abschnitt werde ich Gemeinsamkeiten und Unterschiede zwischen Interaktions- und Navigationstechniken in etablierten und von Hand implementierten 3D-Editoren (vlg. Abschnitt 2.4) und solchen, wie sie in den von DEViL3D generierten Editoren verwendet werden, diskutieren.

Eine Vielzahl 3D-Editoren wird zur Erstellung von 3D-Modellen oder dreidimensionalen CAD-Zeichnungen verwendet. Dabei hingegen ist vollständig freies Editieren gefragt. Das umfasst auch die Erstellung neuer 3D-Konstrukte durch Vereinigung oder Differenz vorgegebener 3D-Formen. Wichtig dafür ist auch die pixelgenaue Positionierung der 3D-Objekte. In 3D-Spracheditoren dagegen werden 3D-Diagramme durch strukturelles Editieren konstruiert, wodurch sichergestellt wird, dass ein konstruiertes Diagramm jederzeit syntaktisch korrekt ist. Charakteristisch dafür sind die auf Anforderungen der jeweiligen Darstellung angepassten Einfügekontexte, die eine nachträgliche pixelgenaue Positionierung der Sprachkonstrukte überflüssig machen. Von den in Abschnitt 2.4 vorgestellten 3D-Editoren hat einzig der LEGO Digital Designer gewisse Ähnlichkeiten mit Spracheditoren, da nur vorgegebene Sprachkonstrukte verwendet werden können und diese auch nur auf schon vorhandenen Konstrukten platziert werden dürfen.

Abgesehen von den skizzierten Unterschieden zwischen freiem und strukturiertem Editieren, umfassen die manuell implementierten 3D-Editoren eine Reihe von etablierten Interaktions- und Navigationstechniken. Diese waren Vorlage für die generierten Spracheditoren mit dem Ziel, dass die gebotene Funktionalität der Spracheditoren mit guten manuell implementierten Editoren vergleichbar ist. Beispielsweise gehören Widgets zur Translation von 3D-Objekten zum Standard fast jeden 3D-Editors. Diese wurden auch für die von DEViL3D generierten Strukturediteren generisch verfügbar gemacht, indem sie in visuellen Mustern gekapselt wurden. So sind die Widgets in individueller Ausprägung für z. B. 3D-Mengen-, Ebenen- oder Listenelemente verfügbar. Auch die in den Spracheditoren verwendeten Navigationstechniken sind jenen aus etablierten 3D-Editoren nachempfunden.

## 6.10 Verwandte Arbeiten

Betrachtet man die 20 Jahre alte Implementierung der ersten 3D-Sprache Cube (siehe Abschnitt 2.2.1), die nach Aussage ihres Entwicklers Najork [Naj96, S. 238] nur von ihm selbst zu bedienen ist, so sind die von DEViL3D generierten Spracheditoren deutlich leistungsfähiger. Zum Einfügen eines neuen Cubes in Najorks 3D-

Implementierung wird deren 3D-Position durch folgendes, grundsätzlich auf Raycasting basierendes Vorgehen, bestimmt: Durch einen Mausklick wird ein Strahl in die 3D-Szene „geschossen“, der durch Rotieren der Szene sichtbar wird. Durch Anklicken eines Punkts auf diesem Strahl wird dann die fehlende $z$-Position bestimmt. Der in diesem Kapitel vorgestellte Einfügekontext für 3D-Mengenelemente erleichtert diese Aufgabe erheblich, da die beiden notwendigen Operationen in einem Schritt zusammengefasst wurden.

Viele der hier vorgestellten Interaktions- und Navigationstechniken finden sich auch in den in Abschnitt 2.4 vorgestellten 3D-Editoren wieder, wie z. B. Widgets zur Manipulation von Sprachkonstrukten. All diese Widgets gehen konzeptionell auf das Papier [CSH+92] zurück.

An die Darstellungseigenschaften der 3D-Sprachkonstrukte angepasste Einfügekontexte sind mir aus anderen 3D-Editoren nicht bekannt. Das gleiche Konzept wurde allerdings schon bei 2D-Editoren verwendet, die vom DEViL-System generiert wurden. Ein auf Meta-Modellen basierendes Interaktionskonzept für 2D-Editoren wird von Bottoni et al. in [BGL06] vorgestellt.

Die vorgestellte Zylinder-Technik zur Mehrfachselektion von Objekten basiert auf einer Idee von Tavanti et al. [TDL04]. Die *Spotlight*-Technik [LG93] zählt ebenfalls zu den Distanzselektionstechniken. Bei dieser Technik erstreckt sich ein Kegel in den 3D-Raum und wählt dasjenige Objekt aus, welches am nächsten an der Kegelachse positioniert ist. Eine Erweiterung dieser Technik ist *Aperature* [FHZ96], bei deren Benutzung der Anwender die Spannweite des Kegels und deren Verschiebung in die Tiefe beeinflussen kann.

Die Steuerung der Kamera durch Maus und Tastatur ist eine besonders in 3D-Spielen weit verbreitete Navigationstechnik. Die in von DEViL3D generierten Editoren zusätzlich angebotene Navigation-Sphere ist mit der Arcball-Metapher verwandt und basiert auf Ideen von Shoemake [Sho92]. Das Orbiting ist von Tan et al. [TRC01] inspiriert, wobei die Berechnung, die die Bewegung der Maus einer Bewegung auf einer Kugeloberfläche zuordnet, in [CMS88] beschrieben wird. Die Lateral-Views zur gleichzeitigen Darstellung orthogonaler Sichten auf die 3D-Szene wird z. B. in 3ds Max (siehe Abschnitt 2.4.1) und vielen weiteren CAD-Werkzeugen verwendet. Einen sehr umfassenden Überblick zahlreicher weiterer Interaktions- und Navigationstechniken ist in [JH13] zu finden.

Bei dem Shutter-3D-Verfahren wird die 3D-Szene von zwei virtuellen und leicht versetzt angeordneten Kameras gerendert, was bei Benutzung der Shutter-Brillen zur deutlichen Wahrnehmung eines 3D-Eindrucks führt. Allerdings wird der Mauscursor, der zur Interaktion mit dem 3D-Diagramm benötigt wird, nicht entsprechend gerendert. Dahingehende Erweiterungen werden in [SE14] vorgestellt.

KAPITEL 7

# Evaluation

In diesem Kapitel sollen das Konzept zur Spezifikation von dreidimensionalen visuellen Sprachen mit DEViL3D und die damit generierten 3D-Editoren evaluiert werden. Von Interesse ist, mit wie viel Aufwand es verbunden ist einen 3D-Editor für eine Sprache mit DEViL3D zu spezifizieren und insbesondere wie gut sich mit diesem 3D-Diagramme konstruieren lassen.

Eine ausführliche Evaluation des DEViL-Systems, auf dem DEViL3D konzeptionell aufbaut, wurde von Schmidt [Sch06, S. 201ff.] durchgeführt. Die Untersuchung teilt sich in die Untersuchung der Usability des Generators auf der einen Seite und der Usability der generierten Editoren auf der anderen Seite. Zur Untersuchung beider Teile werden Fallstudien, Feld-Beobachtungen, Fragebögen und kontrollierte Experimente mit nachfolgendem Interview eingesetzt. Die Evaluation des Generators ergab, dass sowohl kleine Sprachen als auch deutlich komplexere im Team entwickelte Sprachen spezifiziert werden können, der Ansatz also skaliert. Einzig Anfänger haben mit dem System einige Schwierigkeiten, was sich aus dem Lernaufwand für die verschiedenen mächtigen Spezifikationssprachen ergibt. Die generierten Editoren erwiesen sich in der Usability-Untersuchung als benutzerfreundlich, was vor allem auf grundlegende Interaktionstechniken wie *direct manipulation* und *Drag-and-Drop* zurückzuführen ist. Weiterhin hat sich das Verfahren, die Interaktionstechniken durch visuelle Muster bereitzustellen, als sehr nützlich erwiesen, da diese somit immer an das Layout der Sprache angepasst sind und ohne Zutun des Sprachentwicklers im generierten Editor präsent sind. Eine Evaluation der Simulations- und Animationskomponenten des DEViL-Systems hat Cramer [Cra10, S. 149ff.] durchgeführt.

Zur Evaluation des DEViL3D-Systems bietet sich eine ähnliche Zweiteilung an, wie sie zur Evaluation von DEViL herangezogen wurde. Es lassen sich wieder

folgende zwei Ebenen ausmachen: die Spezifikation einer Sprache mit DEViL3D durch den Sprachentwerfer und die Benutzung der generierten Editoren durch die Endanwender. Vergleicht man diese beiden Ebenen mit jenen bei DEViL, fällt auf, dass sich der generierte Editor als Frontend einer Sprachimplementierung vollständig geändert hat. Er verwendet eine 3D- anstatt einer 2D-Darstellung, wodurch sich die Interaktionstechniken grundlegend verändert haben und der Faktor der Navigation hinzugekommen ist. Auf Seiten des Generators konnten sehr viele Konzepte des DEViL-Systems übernommen werden. Aus Perspektive eines Sprachentwicklers ist sogar die Syntax zur Spezifikation der abstrakten Struktur und der visuellen Darstellung nahezu unverändert geblieben. Natürlich wurden die visuellen Muster für 3D-Sprachen von Grund auf neu konzipiert. Die Technik der Anwendung der visuellen Muster ist aus Sicht der Sprachspezifizierer allerdings gleich geblieben. Natürlich müssen 3D-spezifische Gestaltungsaspekte bei der Anwendung der Muster berücksichtigt werden. Aus diesen Betrachtungen ergibt sich, dass der Schwerpunkt dieser Evaluation auf den generierten Editoren liegen muss. Viele Erkenntnisse, die zur Benutzung des Generators DEViL ermittelt wurden, sind aufgrund der hohen Ähnlichkeit der Spezifikationen auch auf DEViL3D übertragbar.

Zur Evaluation von konkreten visuellen Sprachen hat sich die Verwendung des *Cognitive Dimensions Framework* (CDF) von Green und Petre [GP96] etabliert. Es umfasst 13 kognitive Dimensionen, mit denen sich die Qualität von visuellen zweidimensionalen Sprachen beurteilen lässt. Das CDF sollte sich auch auf dreidimensionale Sprachen anwenden lassen, dies wird hier allerdings nicht Betrachtungsgegenstand sein, da der Fokus der Evaluation auf dem Generatorsystem DEViL3D sowie den generierten 3D-Editoren und nicht auf Spracheigenschaften einzelner 3D-Sprachen liegt.

Nachfolgend werde ich zunächst auf Grundlagen zur Usability und zu kontrollierten Experimenten eingehen. Danach folgt die eigentliche Evaluation, die aus Untersuchungen der Benutzbarkeit des Generators und der Benutzbarkeit der generierten Editoren besteht, was eine Untersuchung zur Wahrnehmung des 3D-Eindrucks bei Verwendung von stereoskopischer Hardware einschließt.

## 7.1 Grundlagen der Usability

Mit dem Begriff Usability wird allgemein die Benutzerfreundlichkeit eines Gegenstands oder Produkts bezeichnet. Eine formale Definition liefert die Norm ISO DIS 9241-11[1]:

[1] Die Definition von Usability wurde von mir aus einem Zitat der ISO-Norm in [Jor98, S. 5] ins Deutsche übersetzt.

> Die Usability ist der Grad an Effektivität, Effizienz und Zufriedenheit, die ein bestimmter Nutzer erreichen kann, wenn er festgelegte Ziele mit einem speziellen Produkt verfolgt.

Die Effektivität gibt an, in welchem Umfang ein Ziel oder eine Aufgabe erreicht wird. Die Effizienz misst den Aufwand, der notwendig ist ein Ziel zu erreichen. Die Zufriedenheit gibt den Komfort-Grad an, den Nutzer beim Verwenden des Produkts wahrnehmen und weiterhin, wie akzeptabel das Produkt für die Nutzer beim Erreichen des Ziels ist.

Effektivität, Effizienz und Zufriedenheit lassen sich auf verschiedene Weise feststellen und messen. Allerdings sind sie immer vom Benutzer und dem von ihm verfolgten Ziel abhängig. So ist die Benutzbarkeit stark von der Erfahrung eines Anwenders abhängig. Es sind Produkte vorstellbar, die für Nutzer ohne jegliche Erfahrung nahezu unbenutzbar sind. Andere Benutzer hingegen, die schon Erfahrung mit dem Produkt haben, sind in deren Benutzung geübt. Auf der anderen Seite kann es Ziele geben, die mit Produkt a) gelöst werden können, mit Produkt b) hingegen nicht.

Jordan [Jor98, S. 5ff.] gibt Kriterien zur Klassifikation von Benutzern an, die einen bedeutenden Einfluss auf die Usability haben können: (1) Erfahrung mit dem zu untersuchenden Produkt, (2) Erfahrung mit der Anwendungsdomäne des Produkts, (3) kultureller Hintergrund, (4) Alter und Geschlecht sowie (5) eventuelle Behinderungen. Die Erfahrung mit dem zu untersuchenden Produkt ist ein Faktor, der sich im Vergleich mit den anderen Kriterien schnell ändern kann. Hat ein Benutzer mehr Erfahrung gesammelt, sinken in der Regel die Fehlerrate und der Zeitbedarf für eine Aufgabe. Laut [Jor98, S. 13ff.] nähern sich die beiden Werte asymptotisch einem sogenannten *experienced user performance* (EUP) Level. Dieses Level ist für jeden Nutzer individuell und liegt meist etwas über dem theoretisch möglichen Performanz-Level des Produkts.

Zur Evaluation der Usability eines Produkts gibt es zahlreiche empirische und nicht-empirische Methoden. Dazu zählen u. a. Fragebögen, Interviews, Expertenabschätzungen, Feldstudien und kontrollierte Experimente. In Fragebögen beantworten Benutzer Fragen zu dem untersuchten Produkt; die Antworten werden häufig auf einer *Likert-Skala* angekreuzt. Interviews dienen dazu, Benutzer in einem Gespräch anhand eines Fragenkatalogs nach ihren Erfahrungen mit dem Produkt zu befragen. Bei Expertenabschätzungen bewertet ein Experte anhand von Checklisten ein Produkt. Feldstudien beobachten Probanden bei der täglichen Arbeit mit dem Produkt und finden daher unter nicht kontrollierten Bedingungen statt. Anders verhält es sich bei kontrollierten Experimenten, die unter genau kontrollierten Bedingungen stattfinden, um zu reproduzierbaren Aussagen zu kommen. Zur Evaluation der von DEViL3D generierten 3D-Editoren habe ich kontrollierte

Experimente eingesetzt, auf deren Grundlagen ich im Folgenden genauer eingehen werde. Dabei stütze ich mich auf Lutz Prechelts Buch „Kontrollierte Experimente in der Softwaretechnik“ [Pre01].

### 7.1.1 Aufbau und statistische Auswertung kontrollierter Experimente

Prechelt definiert [Pre01, S. 45] ein kontrolliertes Experiment wie folgt:

> Ein kontrolliertes Experiment ist eine Studie, bei der alle voraussichtlich für das Experiment relevanten Umstände (*Variablen*) konstant gehalten werden (*Kontrolle*), mit Ausnahme von einem oder wenigen, die den Gegenstand der Untersuchung bilden (*Experimentvariablen*). Die Beobachtungen (*abhängige Variablen*) für verschiedene gezielt ausgesuchte Werte der Experimentvariablen (*unabhängige Variablen*) werden miteinander verglichen, um so zu reproduzierbaren Aussagen zu kommen, die eine vor dem Experiment definierte Experimentfrage beantwortet. Die Experimentfrage ist ein genügend enger Aspekt einer relevanten Forschungsfrage.

Bei fortlaufender Wiederholung eines Experiments, bei dem alle Umstände $\mathcal{U}$ bis auf einen gleich sind, müssen eventuelle Unterschiede in den Ergebnissen $\mathcal{E}$ vom gewählten Umstand $\mathcal{U}$ abhängen. Die Ergebnisse $\mathcal{E}$ sind Werte der Größen, die im Verlauf des Experiments gemessen werden und werden, wie auch in obiger Definition, als *abhängige Variablen* bezeichnet. Sie sind abhängig vom Umstand $\mathcal{U}$, der auch als *unabhängige Variable* bezeichnet wird. Sowohl von abhängigen als auch unabhängigen Variablen kann es innerhalb eines Experiments mehrere geben. Letzteres ist der Fall, wenn z. B. die kombinierte Wirkung verschiedener Umstände untersucht werden soll. Abhängige Variablen beschreiben häufig den Aufwand eines Experiments (gemessen wird dabei die benötigte Zeit) und dessen Qualität (Messung der Fehler).

Neben den abhängigen und unabhängigen Variablen gibt es noch *Störvariablen*, die unerkannterweise das beobachtete Ergebnis verändern. Dies sind unterschiedliche individuelle Umstände der Versuchspersonen, die vom Leiter des Experiments gar nicht vollständig ermittelt werden können. Dies ist aber auch nicht notwendig, da sich Störvariablen durch *Replikation* und *Randomisierung* gut kontrollieren lassen. Mit Replikation ist gemeint, dass nicht das Verhalten einer einzelnen Versuchsperson für jeden Wert der unabhängigen Variable verglichen wird, sondern gleich eine ganze Gruppe. Die Mitglieder einer Gruppe werden, zur Erfüllung der Randomisierung, zufällig aus der Menge aller Versuchspersonen ausgewählt.

Der Entwurf eines Experiments besteht aus der Festlegung, (1) welche Gruppen (2) in welcher Reihenfolge (3) welche Aufgaben erledigen und (4) welche abhängigen Variablen dabei beobachtet werden. Ein Experiment kann gemäß verschiedener Pläne entworfen werden, die ich nachfolgend kurz anhand eines Experiment-Beispiels beschreiben werde, bei dem die Produktivität von zwei Programmiersprachen, z. B. Java und Scala, verglichen wird.

Der *Ein-Faktor-Plan* vergleicht zwei Gruppen von Versuchspersonen $G_{Java}$ und $G_{Scala}$ bei der Lösung der gleichen Programmieraufgabe P und misst dabei als abhängige Variablen z. B. die benötigte Zeit und die Korrektheit. Notiert wird der Entwurf wie folgt:

$$G_{Java} : \text{P/Java}$$
$$G_{Scala} : \text{P/Scala}$$

Bei *Mehr-Faktor-Plänen* liefert jede Versuchsperson zwei Datenpunkte, indem eine zweite Aufgabe Q hinzukommt und jeder Proband zwei Aufgaben mit unterschiedlichen Sprachen nacheinander lösen muss:

$$G_1 : \text{P/Java} \quad \text{Q/Scala}$$
$$G_2 : \text{P/Scala} \quad \text{Q/Java}$$

Bei einem solchen Entwurf ist zu beachten, dass durch die Reihenfolge der gewählten Sprache eine dritte unabhängige Variable ins Spiel kommt, die das Ergebnis verfälschen kann. Es wird immer zugleich die Sprache und die Programmieraufgabe geändert. Um dies zu verhindern kann jede der Gruppen halbiert und das Experiment nach folgendem Plan durchgeführt werden:

$$G_{1a} : \text{P/Java} \quad \text{Q/Scala}$$
$$G_{1b} : \text{Q/Scala} \quad \text{P/Java}$$
$$G_{2a} : \text{P/Scala} \quad \text{Q/Java}$$
$$G_{2b} : \text{Q/Java} \quad \text{P/Scala}$$

Zusätzlich gibt es noch Pläne, die eine feste Behandlung vorsehen, d. h. alle Gruppen werden gleich behandelt. In solchen Konstellationen wird oft ein Vortest durchgeführt, um die Kompetenz der Versuchspersonen einzuschätzen oder es wird zwischen den Aufgaben ein Training eingeführt, um anschließend dessen Nutzen zu überprüfen.

Eine unabhängige Variable in einem kontrollierten Experiment heißt *Faktor* und jeder ihrer Werte wird als *Niveau* bezeichnet. In obigem Beispiel ist die Program-

miersprache der Faktor und es gibt die Niveaus Java und Scala. Die Veränderung der abhängigen Variablen zwischen zwei Niveaus heißt *Effekt*.

Zur Auswertung kontrollierter Experimente werden meist statistische Hypothesentests, wie der *t-Test*, eingesetzt. Dabei wird eine konkrete Vermutung (Hypothese) der Form „A ist besser als B bezüglich Eigenschaft X“ aufgestellt und durch statistische Verfahren geprüft. Ein Test der Hypothese, z. B. durch einen t-Test, bekommt als Eingabe eine Stichprobe von Daten und untersucht diese im Hinblick auf ein bestimmtes Merkmal. Dazu wird eine gewisse Annahme (die *Nullhypothese*) als gegeben angenommen und die Wahrscheinlichkeit (bezeichnet als *p-Wert*) berechnet, dass die beobachteten Daten aufgetreten sein können, wenn die Annahme stimmt. Ist die Wahrscheinlichkeit hierfür klein, kann die Nullhypothese verworfen und ihr Gegenteil (die *Alternativhypothese*) als wahr angenommen werden. Die Schwelle, ab wann eine Wahrscheinlichkeit ausreichend klein ist, wird als *Signifikanzniveau* $\alpha$ bezeichnet. In den meisten Untersuchungen, wie auch in dieser Arbeit, wird von $\alpha = 0,05$ ausgegangen.

Meist wird der *Zweistichproben-t-Test nach Welch* verwendet, der die arithmetischen Mittelwerte zweier Stichproben vergleicht. Er setzt voraus, dass die zwei Stichproben einer Normalverteilung entstammen und deren Varianz gleich ist. Zum Testen, ob eine Normalverteilung vorliegt, kann der *Shapiro-Wilk-Test* herangezogen werden. Ermittelt dieser keine Normalverteilung, kann alternativ der *Wilcoxon-Vorzeichen-Rang-Test* angewendet werden, der die Mediane zweier Stichproben vergleicht. Er vergleicht die Datenwerte nicht direkt, sondern die Größenreihenfolgen auf einer Ordinalskala. Bei der Auswertung der Experimente in Abschnitt 7.3 kommen genau diese beiden Tests zur Anwendung.

## 7.2 Usability des Generators

Der Aufbau des Generatorsystems DEViL3D ist konzeptionell vergleichbar mit dem von DEViL, welches von Schmidt [Sch06, S. 210ff.] ausführlich evaluiert wurde. Die meisten dort untersuchten Fragestellungen, die z. B. der Wirksamkeit visueller Muster und der nachträglichen Änderbarkeit visueller Repräsentationen nachgehen, sind auch für den 3D-Sprachen-Generator relevant. Aufgrund der großen Ähnlichkeit der entsprechenden Spezifikationen bei DEViL und DEViL3D, ist mit sehr ähnlichen Ergebnissen zu rechnen. Aus diesem Grund wird der Generator DEViL3D nicht in ähnlicher Breite evaluiert wie sein Vorgängersystem. Hinzu kommt, dass zu einer empirischen Evaluation des Generators Probanden benötigt werden, die eine gewisse Erfahrung im Umgang mit ihm haben. Schmidt konnte dafür auf eine homogene Gruppe von Projektgruppen-Teilnehmern zurückgreifen, die ein Jahr lang eine komplexe visuelle Sprache mit DEViL spezifiziert haben. Auf eine solche Gruppe kann ich nicht zurückgreifen und ein zwangsläufig kurzes

Anlernen ausgewählter Studenten erscheint wenig erfolgversprechend zur Erlangung von aussagekräftigen Ergebnissen. Daher habe ich mich entschieden, den Generatoraspekt des DEViL3D-Systems durch die Vorstellung von implementierten Beispielsprachen und die Untersuchung deren Komplexität zu evaluieren. Dies zeigt zum einen, dass der Generator praktisch relevante Spracheditoren generieren kann und zum anderen gibt die Untersuchung der Komplexität Aufschluss darüber, wie aufwendig die Spezifikation für einen Sprachentwickler ist.

### 7.2.1 Spezifikation von Beispielsprachen

Dieser Abschnitt stellt die mit DEViL3D-System spezifizierten Beispielsprachen vor und demonstriert ein breites Einsatzspektrum, welches von inhärent dreidimensionalen Sprachen – wie Molekülmodelle – über Sprachen wie beispielsweise jene zur Komposition von Musik – die die drei Dimensionen für semantische Zwecke nutzen – bis hin zu Sprachen die wohlbekannte zweidimensionale Modellierungssprachen aufgreifen, um bestimmte Aspekte in 3D deutlicher darzustellen. Diese Sprachen wurden entweder von mir oder im Rahmen von Bachelorarbeiten spezifiziert.

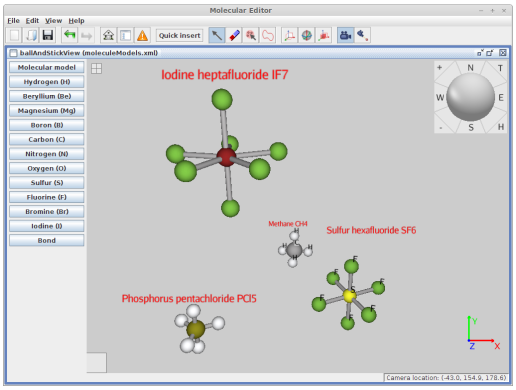
#### Molekülmodelle

Mit dem Editor für Molekülmodelle (siehe Abbildung 7.1a) können Moleküle gemäß dem Kugel-Stäbchen-Modell konstruiert werden. Der Nutzer kann eine Auswahl der Atome aus dem Periodensystem der Elemente in der 3D-Sicht positionieren und Bindungen zwischen ihnen einfügen. Die Berechnung des Layouts eines Moleküls gemäß der chemischen Regeln ist Teil der DEViL3D-Spezifikation dieses Editors (siehe Abschnitt 5.5.1). Weiterhin bietet der Moleküleditor eine zweite Sicht an, die ein bereits konstruiertes Molekül als Kalottenmodell darstellt (siehe Abschnitt 5.5.2).

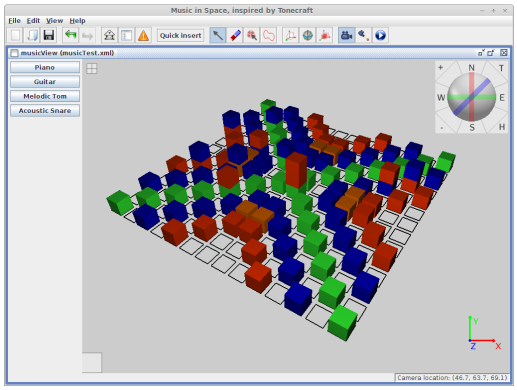
#### Music in Space

Der Editor *Music in Space* (siehe Abbildung 7.1b) ist von der 3D-Web-Anwendung ToneCraft (vgl. Abschnitt 2.2.10) inspiriert und ermöglicht das Komponieren von Musik durch Einfügen von Instrumenten in eine matrixartige 3D-Szene, die durch verschiedenfarbige Würfel visualisiert werden. Durch Klick auf einen Abspiel-Button in der Statusleiste des Editors wird die visuell komponierte Musik abgespielt. Ermöglicht wird dies durch Angabe einer Simulationsspezifikation[2], die

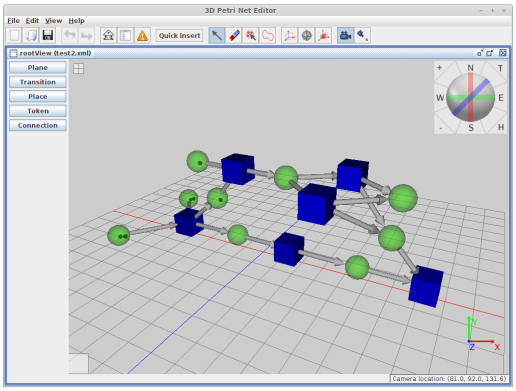
[2]Bei dieser Simulation handelt es sich nicht um eine Transformation des semantischen Modells der visuellen Sprache, wie sie von Cramer [Cra10] für das Vorgängersystem DEViL entwickelt wurde. Das Modell der Sprache bleibt in diesem Fall unverändert.



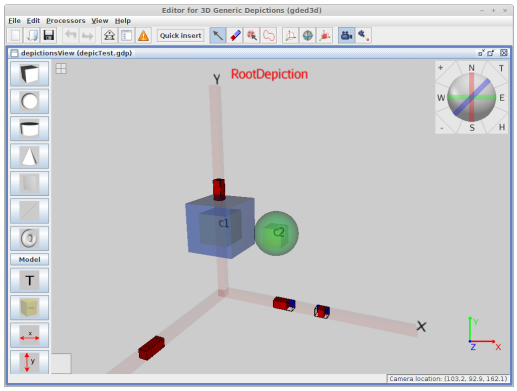

**(a)** Molekülmodelle.



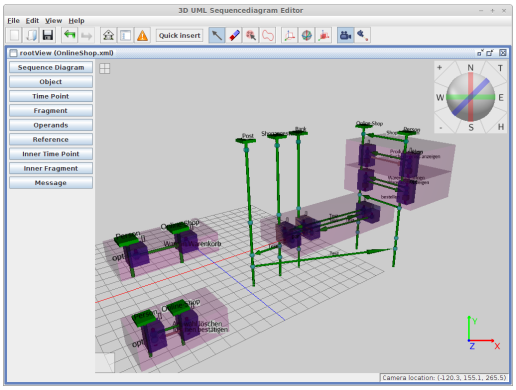

**(b)** Music in Space.



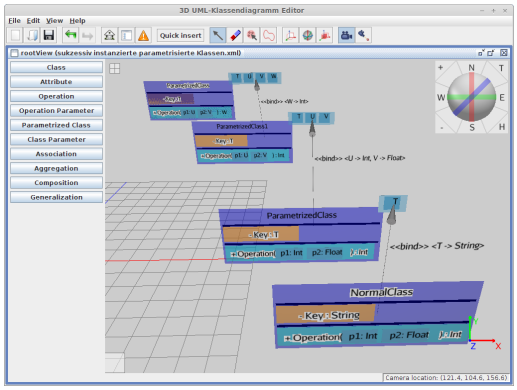

**(c)** 3D-Petri-Netze.




**(d)** Generische 3D-Darstellungen.




**(e)** 3D-Sequenzdiagramme.




**(f)** 3D-Klassendiagramme.

**Abbildung 7.1:** Mit DEViL3D spezifizierte Beispielsprachen.

aus dem 3D-Diagramm einen Musik-String gemäß der *JFugue* Musik-Programmierungs-Bibliothek [Koe14] erstellt und diesen abspielt.

#### 3D-Petri-Netze

Der Editor für 3D-Petri-Netze (siehe Abbildung 7.1c) erlaubt die Konstruktion von Petri-Netzen, deren Stellen und Transaktionen sich auf verschiedenen Ebenen befinden. Damit wird die Idee von Rölke [Röl07] (vgl. auch Abschnitt 2.2.7) aufgegriffen, Teilaspekte eines Petri-Netzes, wie z. B. Daten- und Kontrollfluss, auf diesen Ebenen voneinander zu trennen. Die Spezifikation dieses Editors war besonders gradlinig und ist ein gutes Beispiel für die effiziente Kombination verschiedener visueller Muster: Das Stapeln der Ebenen wird durch das Listen-Muster realisiert, die Stellen und Transitionen unterliegen dem Ebenen-Muster und die sich in den Stellen befindlichen Token werden gemäß des 3D-Mengen-Musters dargestellt.

#### Generische 3D-Darstellungen

Generische 3D-Darstellungen beschreiben das visuelle Erscheinungsbild von Sprachkonstrukten und sind bei DEViL3D Teil der Spezifikation für eine 3D-Sprache. In Abschnitt 4.7 wurde bereits das Konzept der 3D-Darstellungen genauer beschrieben. Die Spezifikation des 3D-Editors (siehe Abbildung 7.1d), mit denen sich die generischen 3D-Darstellungen konstruieren lassen, wurde in Dransfelds Bachelorarbeit [Dra12] begonnen und später weiterentwickelt [Wol13a]. Die Sprache besteht aus verhältnismäßig vielen Sprachkonstrukten, was allerdings daher rührt, dass eine 3D-Darstellung aus verschiedenen grafischen Primitiven bestehen kann. Deren Positionierung in der 3D-Szene obliegt vollkommen dem Benutzer und basiert auf dem 3D-Mengen-Muster.

#### 3D-Sequenzdiagramme

Den Editor für 3D-Sequenzdiagramme (siehe Abbildung 7.1e) hat Stürmann in seiner Bachelorarbeit [Stü14] spezifiziert. Dabei wird die Idee von Gogolla [GRR99; RG00] (siehe auch Abschnitt 2.2.6) aufgegriffen, Sequenzdiagramme in 3D darzustellen, was ermöglicht, Nachrichtenlinien durchdringungsfrei darzustellen. Ein besonderer Fokus bei der Spezifikation des Editors wurde auf kombinierte Fragmente gelegt, die es erlauben Nebenläufigkeit, Bedingungen oder Schleifen darzustellen. Diese werden innerhalb von transparenten Quadern dargestellt und können rekursiv geschachtelt werden. Durch Experimentieren mit verschiedenen Sequenzdiagrammen wurde herausgefunden, dass tiefes Schachteln von kombinierten Fragmenten das Sequenzdiagramm schnell unübersichtlich macht.

#### 3D-Klassendiagramme

Die Sprache für 3D-Klassendiagramme wurde von Stolte im Rahmen seiner Bachelorarbeit [Sto14] spezifiziert. Der Fokus lag auf einer verbesserten Darstellung von parametrisierten Klassen im 3D-Raum. Der Editor ermöglicht die Konstruktion von Klassendiagrammen, wie sie aus der 2D-UML bekannt sind; unterstützt werden Klassen, Interfaces, Attribute, Operationen und verschiedene Verbindungen (Assoziation, Aggregation, Komposition und Generalisierung) zwischen Klassen. Diese Diagramme werden auf einer Ebene konstruiert. Die dritte Dimension kommt ins Spiel, wenn das Klassendiagramm parametrisierte Klassen enthält. Eine parametrisierte Klasse besitzt zusätzlich eine nicht leere Menge von Parametern, die an der oberen rechten Ecke der Klassenrepräsentation dargestellt wird. Andere Klassen können parametrisierte Klassen instanziieren, wobei ein Parameter an einen konkreten Typ gebunden wird. Dieses Konzept ist aus Sprachen bekannt, die Polymorphie unterstützen (z. B. Java mit generischen Klassen oder C++ mit Templates). Parametrisierte Klassen befinden sich im 3D-Editor auf entlang der $z$-Achse versetzten Ebenen. Je mehr Parameter die Klasse hat, desto weiter hinten befindet sie sich. Auf jeder dieser Ebenen befinden sich parametrisierte Klassen, die die gleiche Anzahl Parameter haben. Dies und die sukzessive Instanziierung der Klassen ist in Abbildung 7.1f gut zu sehen.

### 7.2.2 Komplexität der Sprachspezifikationen

Dieser Abschnitt untersucht den Aufwand, der notwendig ist, um die oben vorgestellten 3D-Sprachen mit DEViL3D zu spezifizieren. Dabei wird zwischen verschiedenen Spezifikationen, z. B. für die abstrakte Struktur oder visuelle Darstellung, unterschieden. Die quantitativen Daten über Spezifikationen für die Sprachen Molekülmodelle, Music in Space, Petri-Netze, generische 3D-Darstellungen, 3D-Sequenzdiagramme und 3D-Klassendiagramme sind in Tabelle 7.1 zu finden. Der Spezifikationsaufwand wird dort durch die Anzahl der Quelltextzeilen (*lines of code*, LOC) ohne Kommentare sowie durch die Anzahl von Klassen in der abstrakten Struktur und die Anzahl der angewendeten visuellen Mustern quantifiziert. Die Tabelle ist in fünf Bereiche geteilt: Charakteristika der abstrakten Struktur, der visuellen Darstellung, der strukturellen Konsistenzprüfungen und besonderen Spezifikationen z. B. für sprachspezifisches Layout. Der fünfte Bereich summiert die LOC für jede der Sprachen.

Die Anzahl der Klassen die in der abstrakten Struktur spezifiziert sind, sind ein guter Indikator für die Komplexität einer Sprache. Dabei kann zwischen abstrakten und nicht-abstrakten Klassen unterschieden werden. Die Anzahl der abstrakten Klassen ist ein Maß dafür, wie viel Abstraktion eine Sprache verwendet und damit für deren Komplexität. Sie ist in der ersten Zeile der Tabelle in Klammern notiert.

| | Molekül-modelle | Music in Space | 3D-Petri-Netze | Generische 3D-Darstellungen | 3D-Sequenz-diagramme | 3D-Klassen-diagramme |
|---|---|---|---|---|---|---|
| # Klassen (abstrakt) | 22(9) | 9(1) | 7(1) | 32(6) | 29(8) | 19(3) |
| LOC abstrakte Struktur | 81 | 27 | 30 | 146 | 184 | 108 |
| # Sichten | 2 | 1 | 1 | 2 | 1 | 1 |
| # Muster-Anwend. (# verschied. Muster) | 31(4) | 8(4) | 9(6) | 27(5) | 32(6) | 30(7) |
| # Muster-Anwend./# verschied. Muster | 7,75 | 2 | 1,33 | 5,4 | 5,33 | 4,29 |
| # generische Darstellungen | 12 | 5 | 4 | 40 | 17 | 10 |
| LOC visuelle Darstellung | 315 | 69 | 78 | 697 | 455 | 302 |
| # Prüf-Funktionen | 8 | – | – | 15 | 1 | 5 |
| # Synchronisations-Funktionen | 12 | 1 | – | 8 | 7 | 9 |
| # Initialisierungs-Funktionen | – | 1 | 1 | 2 | 8 | 3 |
| LOC Prüf-/Sync.-/Init.-Fkt. | 138 | 30 | 9 | 523 | 1163 | 409 |
| LOC Codegenerierung | – | – | – | 422 | – | 160 |
| LOC benutzerdefiniertes Layout | 320 | – | – | – | – | – |
| LOC Simulation | – | 137 | – | – | – | – |
| Summe LOC | 854 | 263 | 117 | 1788 | 1802 | 819 |

**Tabelle 7.1:** Komplexität von 3D-Sprachspezifikationen.

Die sechs Sprachen decken ein Spektrum von sieben Klassen bei Petri-Netzen bis zu 32 Klassen bei der Sprache für generische 3D-Darstellungen ab.

Die meisten der Sprachen bestehen nur aus einer Sicht, mit Ausnahme von z. B. der Molekül-Sprache, die neben der Sicht für das Kugel-Stäbchen-Modell noch eine Sicht anbietet, die Moleküle als Kolottenmodell darstellt. Neben der Anzahl der verschiedenen Sichten enthält die Tabelle Angaben dazu, wie oft visuelle Muster angewendet wurden. In Klammern ist notiert, wie viele verschiedene Muster die Sprache verwendet. Der Quotient aus diesen beiden Angaben ist ein Maß für die visuelle Komplexität der Sprache: je kleiner der Wert ist, desto breiter gefächert ist die visuelle Darstellung. Die Anzahl der generischen 3D-Darstellungen und die LOC für die Spezifikation der visuellen Darstellung korrelieren mit der Anzahl in der abstrakten Struktur spezifizierter Klassen.

Je komplexer die Struktur und damit einhergehend auch die visuelle Darstellung einer Sprache sind, desto größer ist die Wahrscheinlichkeit, dass die Sprache semantische Prüfungen oder Synchronisation benötigt. Der dritte Abschnitt der Tabelle umfasst diesen Aspekt. Im vierten Teil wird der Spezifikationsaufwand für die Verwendung spezieller Funktionalitäten dargestellt. Für den Moleküleditor wurde eine Spezifikation zur Realisierung von sprachspezifischem Layout angegeben. Für die Music in Space Sprache wurde Simulationscode spezifiziert, der das 3D-Diagramm in einen Musik-String überführt, der anschließend abgespielt werden kann. Für die generischen 3D-Darstellungen wurde die Schnittstelle zur Codegenerierung genutzt, um das 3D-Diagramm in Java-Code zu übersetzen.

Die Summe der benötigten LOC zeigt, dass selbst relativ umfangreiche Sprachen mit moderatem Aufwand spezifiziert werden können. Umfangreiche Sprachen wie generische 3D-Darstellungen oder 3D-Sequenzdiagramme lassen sich mit einem Aufwand von ca. 1800 LOC realisieren, ganz zu schweigen von mit 117 LOC sehr übersichtlichen Spezifikationen für 3D-Petri-Netze. Die Anzahl der insgesamt notwendigen LOC hängt in großem Maße auch von der Anzahl der definierten Klassen ab.

### 7.2.3 Zusammenfassende Betrachtungen

Mit DEViL3D wurden verschiedenartige 3D-Sprachen unterschiedlicher Anforderungen und Sprachstile spezifiziert. Dies demonstriert die Leistungsfähigkeit des Generators. Die Sprachen nutzen die dritte Dimension in verschiedener Weise: von inhärent dreidimensionalen Molekülmodellen über Editoren für Petri-Netze und zur Musik-Komposition, die den Dimensionen eine Semantik zuordnen, sowie dem Editor für generische Darstellungen, der wiederum Teil des DEViL3D-Systems ist, hin zu UML-Sequenz- und -Klassendiagrammen, mit denen sich bestimmte Zusammenhänge in 3D darstellen lassen.

Diese Sprachen weisen unterschiedliche Komplexität auf. Zur Untersuchung der Sprachspezifikationen wurden die Maße Anzahl der Quelltextzeilen (LOC) und Anzahl der Sprachkonstruktklassen herangezogen. Bei den meisten Sprachspezifikationen wird deren LOC durch die Anzahl der Sprachkonstruktklassen determiniert. Allerdings benötigen einige Sprachen, wie z. B. die Sequenzdiagramme, besonders viel Synchronisation, was einen zusätzlichen aber insgesamt immer noch überschaubaren Spezifikationsaufwand bedeutet. Weiterhin kann es notwendig sein, zusätzliche Spezifikationen für Codegenerierung, benutzerdefiniertes Layout oder Simulation bereitzustellen.

## 7.3 Usability der generierten Editoren

Die von DEViL3D generierten 3D-Spracheditoren sind komplexe Editorimplementierungen. Sie umfassen sprachspezifische Eigenschaften, wie z. B. auf die Sprache zugeschnittene Layoutaspekte. Darüber hinaus besitzt jeder 3D-Strukturedítor universelle Funktionen, die zum effizienten Arbeiten mit dem Editor unabdingbar sind. Dazu zählen *Cut-and-Paste*, *Undo*- und *Redo*-Funktionalität, das Abspeichern von 3D-Diagrammen in und das Laden aus einer XML-Datei, der Export von „Schnappschüssen“ eines Diagramms als Bild-Datei, eine baumbasierte Sicht, mit der das 3D-Diagramm auf Struktur-Ebene betrachtet und editiert werden kann, sowie das textbasierte Suchen nach Sprachkonstrukten. Diese Funktionen sind allerdings nicht spezifisch für mit DEViL3D generierte 3D-Editoren, sondern sind so auch in vielen anderen Editoren (u. a. auch in von DEViL generierten 2D-Editoren) zu finden. Daher spielen diese Funktionen bei der Evaluation keine Rolle. Stattdessen soll hier die Usability von ausgewählten 3D-spezifischen Funktionen zur Interaktion und Navigation, wie sie in Kapitel 6 beschrieben wurden, evaluiert werden (vgl. auch [WK14] und [WK15]).

In diesem Abschnitt werden zunächst die Experimentfragen vorgestellt und genauer dargelegt, welche Interaktions- und Navigationsaspekte evaluiert werden. Danach werden die Vorkenntnisse der den Experimenten zur Verfügung stehenden Probanden beschrieben. Eine genaue Aufgabenbeschreibung erklärt, wie die Experimente strukturiert und unter welchen Bedingungen sie durchgeführt wurden. An dieser Stelle wird auch begründet, welche abhängigen und unabhängigen Variablen gewählt wurden. Danach folgt für jeden evaluierten Interaktions- und Navigationsaspekt ein Abschnitt mit genauer Beschreibung des Experiments. Abschließend erfolgt eine Diskussion der erzielten Ergebnisse.

### 7.3.1 Experimentfragen

Die von DEViL3D generierten 3D-Strukturediteren bieten eine Vielzahl an Navigations- und Interaktionstechniken an. Für viele der damit zu absolvierenden Aufgaben gibt es unterschiedliche Techniken, die vom Anwender nach individueller Präferenz oder in Abhängigkeit der zu erledigenden Aufgabe gewählt werden können. So bietet ein 3D-Editor z. B. unterschiedliche Techniken zur Navigation in der 3D-Szene und zur Mehrfachauswahl von Sprachkonstrukten an. Weiterhin werden in einem 3D-Editor Navigationshilfen wie die Lateral-Views zur Verfügung gestellt, die den Anwendern helfen sollen sich in komplexen 3D-Szenen mit vielen sich unter Umständen verdeckenden Sprachkonstrukten zurechtzufinden. Ein 3D-Editor bietet außerdem verschiedene Techniken an, um mit geschachtelten Strukturen zu interagieren. Außerdem kann die 3D-Szene eines Editors neben der normalen monoskopischen Darstellung auch stereoskopisch dargestellt werden. Daher werden diese konkurrierenden Techniken empirisch miteinander verglichen.

Die in den Experimenten zu untersuchenden Fragen sind im Einzelnen folgende:

1. Wie gut sind die Navigationstechniken benutzbar und welche wird von den Benutzern bevorzugt?

2. Gibt es einen positiven Einfluss bei Benutzung der Lateral-Views, wenn sich in der 3D-Szene Objekte verdecken?

3. Wie gut können Benutzer mit hierarchisch geschachtelten Objekten interagieren?

4. Welche der Techniken zur Mehrfachselektion (Lasso- oder Zylinder-Metapher) wird von den Benutzern bevorzugt?

5. Ist die 3D-Wahrnehmung einer 3D-Szene mit stereoskopischer Hardware besser als bei monoskopischer Darstellung?

### 7.3.2 Probanden

Die Experimente wurden mit zeitlichem Abstand in zwei voneinander unabhängigen Durchgängen absolviert. Der erste Durchgang adressierte die ersten vier Experimentfragen. Im zweiten Durchgang wurde der Einfluss von stereoskopischer Hardware auf die 3D-Wahrnehmung untersucht. Weiterhin wurde ein zweites Experiment zur Interaktion mit hierarchisch geschachtelten Objekten durchgeführt (Details dazu sind Abschnitt 7.3.6 zu entnehmen).

| | mit Vorkenntnissen | davon für eine Dauer von... | | | | |
|---|---|---|---|---|---|---|
| | | < 1 Jahr | 1–2 J. | 3–5 J. | 6–10 J. | > 10 J. |
| 3D-Spiele | 94,4 % | 0,0 % | 5,9 % | 23,5 % | 11,8 % | 58,8 % |
| 3D-Editoren | 83,3 % | 40,0 % | 33,3 % | 20,0 % | 6,7 % | 0,0 % |
| 3D-Programmierung | 72,2 % | 38,5 % | 38,5 % | 23,0 % | 0,0 % | 0,0 % |
| DEViL3D | 16,7 % | 66,7 % | 33,3 % | 0,0 % | 0,0 % | 0,0 % |

**Tabelle 7.2:** Auflistung der Vorkenntnisse der 18 Probanden.

### Erster Experiment-Durchgang

Die Experimente im ersten Durchgang wurden mit 18 männlichen Probanden durchgeführt. Diese waren entweder Studenten (89 %) oder technische Mitarbeiter (11 %) an der Universität Paderborn. Die meisten der teilnehmenden Studenten wurden im Rahmen der Bachelor-Veranstaltung *Programming Languages and Compilers* um die Teilnahme am Experiment gebeten. Alle anderen wurden direkt angesprochen. Die Studenten studieren entweder Informatik oder sehr verwandte Studiengänge. 50 % von ihnen sind als Bachelor-Studenten eingeschrieben, 31 % streben einen Master-Abschluss an und 19 % sind Promotionsstudenten. Das Alter der Probanden liegt im Mittel bei 23,9 Jahren (Standardabweichung: 2,9 J.; Median: 23 J.). Bei der Frage nach Vorkenntnissen gaben 94 % an, bereits 3D-Spiele gespielt zu haben, 59 % davon bereits seit mehr als 10 Jahren. 83 % haben mit 3D-Editoren gearbeitet und 72 % haben Erfahrung mit 3D-Programmierung. Lediglich knapp 17 % haben bereits mit DEViL3D und damit generierten Editoren gearbeitet.

Tabelle 7.2 gibt einen genaueren Überblick über die Vorkenntnisse der Probanden und zeigt eindeutig, dass die Mehrheit der Probanden sehr viel Erfahrung mit 3D-Anwendungen hat, insbesondere mit 3D-Spielen und 3D-Editoren. Dies mag für manchen aufgrund der starken Vorkenntnisse eine fragwürdige Probanden-Auswahl sein.[3] Dieser Meinung bin ich nicht, sondern sehe ganz im Gegenteil darin sogar einen Vorteil diesbezüglich, dass diese Probanden durch ihre erworbene Erfahrung wissen, was von einer 3D-Anwendung zu erwarten ist. Dies bedeutet zum einen weniger Einarbeitungszeit in die Benutzung der 3D-Editoren und zum anderen halte ich es für wahrscheinlich, dass weniger Störvariablen das Experiment beeinflussen, die darauf zurückzuführen wären, dass Probanden durch Unvertrautheit mit 3D-Umgebungen von der eigentlichen Aufgabe abgelenkt werden.

[3] Dieser Meinung war ein Reviewer der VINCI 2014 Konferenz, der in der Beurteilung des Papiers [WK14] folgendes anmerkte: „The composition of the user study participants is questionable, the participants appear to have too much experience with 3D editors/programming/etc.“

#### Zweiter Experiment-Durchgang

Im zweiten Durchgang standen zwölf Probanden zur Verfügung, die allerdings durch Verwendung eines Mehr-Faktor-Plans jeweils zwei Datenpunkte liefern konnten (siehe nächsten Abschnitt). Die Teilnehmer waren im Durchschnitt 25,8 Jahre alt (Standardabweichung: 3,3 J.; Median: 26 J.). Jeweils ein Drittel der Teilnehmer war im Bachelor-, Master- bzw. Promotionsstudium eingeschrieben. Die Vorkenntnisse waren ähnlich verteilt wie bei den Teilnehmern des ersten Durchgangs. Mit 3D-Spielen hatten 92 % der Probanden Erfahrungen und davon alle bis auf einen sogar seit mehr als zehn Jahren. 75 % hatten Erfahrungen mit stereoskopischen 3D-Darstellungen und ein Drittel davon auch mit 3D-Shutter-Brillen.

### 7.3.3 Aufgabenbeschreibung

Zur Beantwortung der in Abschnitt 7.3.1 aufgeworfenen Experimentfragen wurden fünf kontrollierte Experimente konzipiert, die jeweils eine Experimentfrage adressieren. Für jeden Teil mussten die Probanden mit einem bereitgestellten 3D-Editor eine bestimmte Navigations- oder Interaktionsaufgabe lösen. Die auf das jeweilige Experiment zugeschnittenen Editoren wurden vorher von mir mit DEViL3D spezifiziert. Dadurch sind in den 3D-Szenen keine Elemente enthalten, die die Probanden vom eigentlichen Ziel ablenken können. Damit sollten auch die Antworten der Experiment-Teilnehmer auf die adressierten Fragen möglichst wenig durch Entwurfsentscheidungen der speziellen Sprache beeinflusst werden. Die 3D-Editoren bestehen aus einer zugehörigen 3D-Szene, die bestimmte Objekte enthält. Die gestellte Aufgabe musste mit einer vorgegebenen Technik erledigt werden. Im Anschluss mussten die Teilnehmer Fragen dazu beantworten. Vor dem Start der Experimente habe ich den Probanden jeden einzelnen 3D-Editor und die damit durchzuführende Aufgabe beschrieben. Da die meisten Probanden keine konkrete Erfahrung mit den zu untersuchenden 3D-Editoren mitbrachten, konnten sie sich in einer Eingewöhnungsphase – einer Art informellem Vortest – ausführlich mit jedem Editor praktisch vertraut machen. Dafür haben die Probanden an einer einfachen Beispielaufgabe die später zu lösende Aufgabe erprobt. Daran hat sich auch gezeigt, dass keiner der Probanden farbenblind ist, was bei einigen Aufgaben zwangsläufig zu Fehlern geführt hätte.

Beim ersten Experiment-Durchgang wurden die Probanden zur Durchführung der Experimente zufällig in zwei Gruppen aufgeteilt. Für die Experimente wurde ein Ein-Faktor-Plan herangezogen, um evtl. auftretende Lerneffekte, wie sie bei Mehr-Faktor-Plänen vorkommen können, zu vermeiden. Die Teilnehmer jeder Gruppe mussten die gleiche Aufgabe unter Verwendung verschiedener Techniken

durchführen. Beim Vergleich des achsenbasierten und freien Modus der Navigation z. B. musste die Aufgabe A nach folgendem Plan gelöst werden:

$$G_{achsenbasiert} : \text{A/achsenbasierter Modus}$$
$$G_{frei} : \text{A/freier Modus}$$

Die unabhängige Variable in dem Experiment ist jeweils die Wahl der vorgegebenen Navigations- oder Interaktionstechnik der Gruppe. In obigem Beispiel ist dies die Wahl, ob der achsenbasierte oder freie Modus verwendet wird. Die abhängigen Variablen sind die Zeit zur Lösung der Aufgabe, sowie die Information, ob die Aufgabe korrekt gelöst wurde. Am Ende jedes Teilexperiments wurde den Probanden eine Frage zur Beantwortung auf einer Likert-Skala mit sechs Items gestellt, um ihre persönliche Einschätzung zu der angewendeten Technik zu erfragen. Das vollständige Experiment dauerte für jeden der Probanden – inkl. Einführung, Eingewöhnungsphase und vier Tests – zwischen 20 und 30 Minuten.

Dem zweiten Experiment-Durchgang wurde ein Mehr-Faktor-Plan zugrunde gelegt und es mussten von jedem Probanden zwei Aufgaben mit unterschiedlichen Techniken gelöst werden. Um beim Experiment-Entwurf nicht gleichzeitig die zu lösende Aufgabe und die zur Verfügung stehende Technik zu variieren, wurden die Gruppen, gemäß dem Schema aus Abschnitt 7.1.1, jeweils unterteilt. Damit ergibt sich für das Experiment zum Vergleich von stereoskopischer und monoskopischer Darstellung beispielhaft folgender Plan:

$$G_{1a} : \text{A/monoskopisch} \quad \text{B/stereoskopisch}$$
$$G_{1b} : \text{B/stereoskopisch} \quad \text{A/monoskopisch}$$
$$G_{2a} : \text{A/stereoskopisch} \quad \text{B/monoskopisch}$$
$$G_{2b} : \text{B/monoskopisch} \quad \text{A/stereoskopisch}$$

Im Folgenden werde ich die Tests genauer beschreiben, den dafür verwendeten Editor vorstellen und konkret die jeweils zu lösende Aufgabe beschreiben. Danach erfolgt jeweils die Präsentation der ermittelten Ergebnisse.

### 7.3.4 Experiment: Navigation

Innerhalb der 3D-Sicht in von DEViL3D generierten Strukturediteren kann der Anwender auf drei verschiedene Arten navigieren. Zum einen kann mithilfe von sechs Tastatur-Tasten in Kombination mit der Maus und zum anderen mit der Navigation-Sphere in der oberen rechten Ecke der Sicht navigiert werden. Diese wiederum kann in zwei Modi – dem freien und achsenbasierten – betrieben wer-

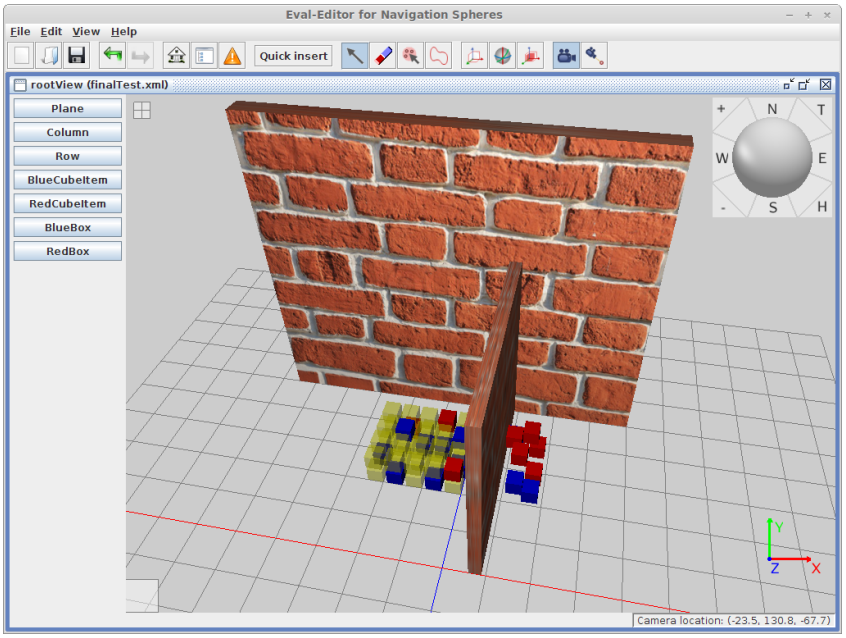



**Abbildung 7.2:** Editor zum Testen der Navigation.

den. Im Experiment wurden die beiden Modi der Navigation-Sphere miteinander verglichen.

### Der verwendete Editor

Der Editor der beim Experiment zur Navigation-Sphere zum Einsatz kam, ist in Abbildung 7.2 zu sehen. Die 3D-Szene besteht aus zwei Wänden, hinter denen sich in zwei Gruppen rote und blaue Würfel verbergen. Diese sind so platziert, dass die Szene aus verschiedenen Blickwinkeln betrachtet werden muss, um alle Würfel zu sehen. Die Aufgabe der Probanden war es, möglichst schnell die Anzahl der blauen Würfel zu ermitteln. In der Startposition sind die Gruppen der Würfel noch nicht zu sehen, sondern die Sicht zeigt nur die Vorderseite der großen Wand. Die eine Gruppe von Probanden nutzte zur Exploration der Szene den achsenbasierten Modus der Navigation-Sphere, die andere Gruppe den freien Modus.

### Die Ergebnisse

Unter Variation des gewählten Navigationsmodus als unabhängige Variable ergaben sich für die zwei Gruppen folgende Zeiten für die Durchführung der Aufgabe: die Gruppe mit dem freien Modus genötigte im Mittel 20 Sekunden (Standardabweichung = 8.75 Sek., Median = 21 Sek.) wohingegen die Gruppe mit dem achsenbasierten Modus 42,4 Sekunden im Durchschnitt (Standard-

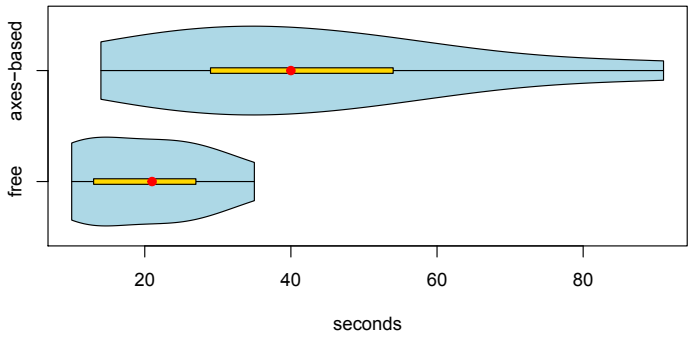



**Abbildung 7.3:** Benötigte Zeit zum Lösen der Navigationsaufgabe.

abweichung = 23,37 Sek., Median = 40 Sek.) benötigte. Damit ist die Nutzung des freien Modus nach dem Zweistichproben-t-Test nach Welch signifikant schneller ($p$-Wert = 0,022). Zu sehen ist dies auch deutlich an dem *Violin-Plot* in Abbildung 7.3. Einige der Probanden bezeichneten den freien Modus als „intuitiver".

Die starken Unterschiede bei den gemessenen Zeiten hatten hingegen keinen großen Einfluss auf die Anzahl der korrekt gelösten Aufgaben: 56 % der Teilnehmer mit dem freien Modus lösten die Aufgabe korrekt, wohingegen nur 44 % mit dem achsenbasierten Modus fehlerfrei blieben. Interessanterweise gibt es eine starke positive Korrelation ($r$ = 0,76) zwischen den Jahren an Erfahrung mit 3D-Spielen und der hier korrekt gelösten Aufgabe. Beim direkten Vergleich der Datensätze ergab sich sogar, dass all jene, die die Aufgabe korrekt gelöst haben, mehr als zehn Jahre Erfahrung mit 3D-Spielen haben. Dies legt die Vermutung nahe, dass Anwender, die viel Erfahrung mit 3D-Anwendungen haben, sich auch schnell in anderen 3D-Umgebungen zurechtfinden.

Im Anschluss an den Test mussten die Probanden die Frage beantworten, für wie gut benutzbar sie die verwendete Navigationstechnik empfanden. Die Antworten wurden auf einer Likert-Skala mit sechs Items von „sehr gut benutzbar" bis „überhaupt nicht gut benutzbar" gegeben. Der freie Modus schnitt dabei leicht besser ab (siehe Abbildung 7.4).

Nach Abschluss des Haupttests wurden die Probanden jeder Gruppe aufgefordert, die beiden Navigationstechniken zu testen, die sie bisher noch nicht kennengelernt hatten (Tastatur und freier Modus oder Tastatur und achsenbasierter Modus). Anschließend mussten sie eine Rangfolge der drei Techniken aufstellen, die ihre Präferenz widerspiegelt. Die meisten der Teilnehmer präferierten die Navigation mittels Tastatur, gefolgt vom freien Modus; abgeschlagen auf dem dritten Platz landete der achsenbasierte Modus. Letzterer war für die meisten Teilnehmer zu unflexibel, da eine Rotation immer nur auf eine Achse beschränkt ist. Dass die Tastatur-Navigation von den meisten bevorzugt wird, ist insofern nicht überra-

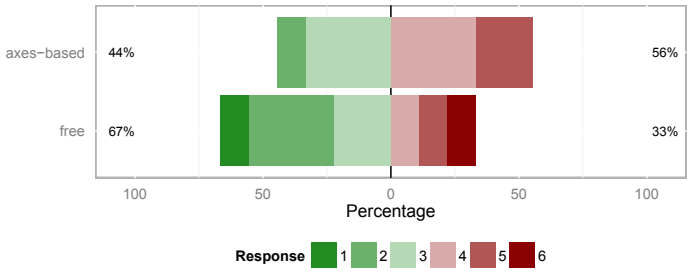



**Abbildung 7.4:** Anhand einer Likert-Skala ermittelte Einschätzung zur Benutzbarkeit der Navigationstechniken. Der Wert 1 steht auf der Skala für „sehr gut benutzbar“ und der Wert 6 für „überhaupt nicht gut benutzbar“.

schend, als dass die Teilnehmer stark von 3D-Spielen geprägt sind, die mehrheitlich auf diese Tastatur-Navigation setzen.

### 7.3.5 Experiment: Lateral-Views

Die Lateral-Views, die in jedem 3D-Editor hinzugeschaltet werden können, zeigen die 3D-Szene zusätzlich zu der Hauptsicht aus drei weiteren Perspektiven orthogonal zu den drei Achsen. Die Vermutung ist, dass die zusätzlichen Sichten hilfreich sein können, wenn sich in der Standardsicht besonders viele Sprachkonstrukte verdecken.

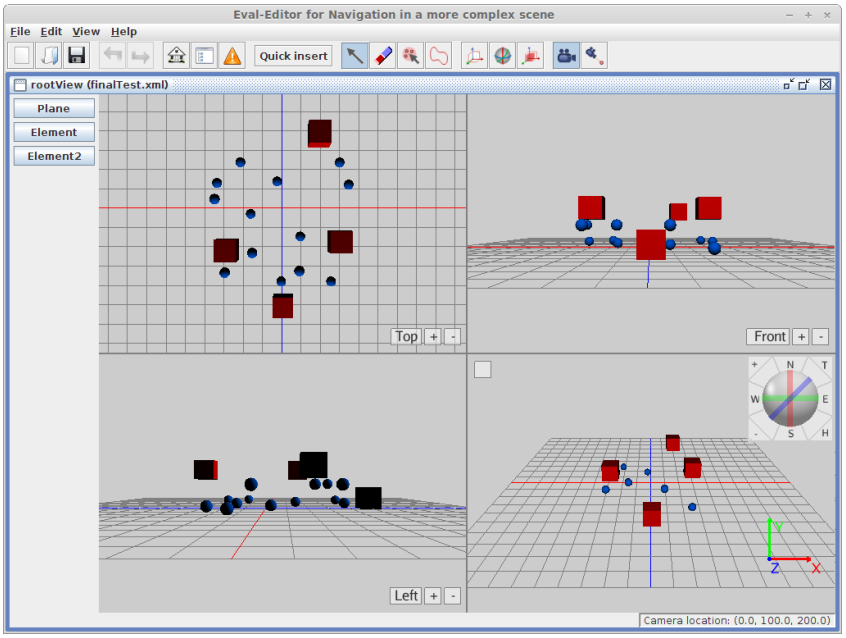



**Abbildung 7.5:** Editor zum Testen der Lateral-Views.

### Der verwendete Editor

Der in Abbildung 7.5 zu sehende Editor besteht aus einer 3D-Sicht in der sich rote Würfel und blaue Kugeln befinden, von denen sich in der Startposition (rechte untere Sicht in Abbildung 7.5) viele gegenseitig verdecken. Im Experiment war es die Aufgabe der Probanden, die Anzahl der roten Würfel und blauen Kugeln zu ermitteln. Die eine Gruppe durfte die Lateral-Views nutzen (wie in Abbildung 7.5), der anderen Gruppe stand nur die einfache Sicht zur Verfügung. Beide Gruppen durften die Navigation-Sphere nutzen, um die Szene aus verschiedenen Blickwinkeln zu betrachten.

### Die Ergebnisse

Entgegen meiner Erwartungen konnte im Experiment kein Vorteil für die Lateral-Views gemessen werden; das Gegenteil war sogar der Fall. Die Probanden, die die Lateral-Views benutzten, benötigten im Schnitt 28,2 Sekunden (Standardabweichung = 20,68 Sek., Median = 19 Sek.) für das Lösen der Aufgabe; die Gruppe, der die einfache Navigation ohne weitere Hilfen zur Verfügung stand, benötigte im Mittel 12,4 Sekunden (Standardabweichung = 5,29 Sek., Median = 11 Sek.); siehe auch Abbildung 7.6. Gemäß dem Wilcoxon-Vorzeichen-Rang-Test[4] benötigte das Lösen der Aufgabe mit Lateral-Views signifikant ($p$-Wert = 0,0039) mehr Zeit als ohne sie. Allerdings war die Anzahl korrekter Lösungen bei Verwendung der Lateral-Views mit 78 % geringfügig höher, verglichen mit 67 % korrekter Lösungen bei normaler Navigation.

Bei der Beobachtung der Probanden fiel auf, dass nur zwei die Lateral-Views zielgerichtet einsetzten. Alle anderen schienen sie zu ignorieren und ausschließlich die normale 3D-Sicht in der rechten unteren Ecke (siehe Abbildung 7.5) zu

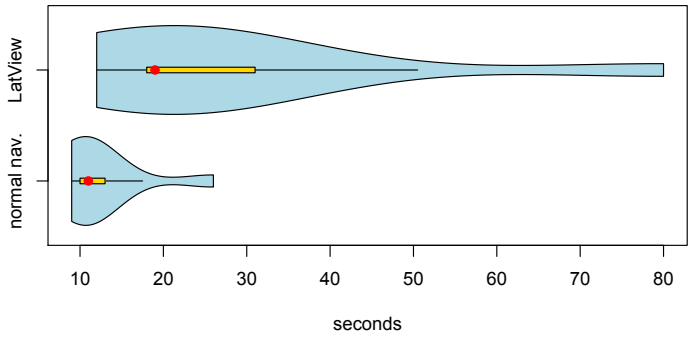



**Abbildung 7.6:** Benötigte Zeit zum Zählen von Sprachkonstrukten mit und ohne Lateral-Views.

[4]Der Wilcoxon-Vorzeichen-Rang-Test wurde angewendet, da der Shapiro-Wilk-Test keine Normalverteilung der gemessenen Zeiten ermittelt hat. Nur bei normalverteilten Zeiten kann der t-Test nach Welch angewendet werden.

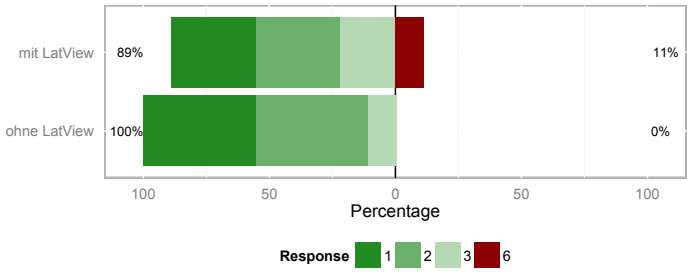



**Abbildung 7.7:** Anhand einer Likert-Skala ermittelte Einschätzung wie hilfreich die Lateral-Views sind. Der Wert 1 steht auf der Skala für „sehr hilfreich“ und der Wert 6 für „überhaupt nicht hilfreich“.

verwenden. Die Teilnehmer der anderen Gruppe hatten hingegen nur die klassische 3D-Sicht auf die sie sich vollständig konzentrieren konnten. Die Vermutung liegt nahe, dass für die meisten Probanden die Lateral-Views zusätzliche kognitive Ressourcen beansprucht haben, was sie so von der Aufgabe etwas ablenkte. Allerdings ist es denkbar, dass sich Probanden mit mehr Training an die Lateral-Views gewöhnen und sie zielgerichteter einsetzen. Bei der abschließenden Frage, ob die für die Aufgabe zur Verfügung stehende Technik zum Inspizieren der 3D-Szene ausreichend ist (die Antworten wurden in einem Spektrum von „sehr hilfreich“ bis „überhaupt nicht hilfreich“ gegeben), erzielten beide Möglichkeiten ähnlich gute Ergebnisse auf der Likert-Skala (siehe Abbildung 7.7).

### 7.3.6 Experiment: Interaktion mit geschachtelten Strukturen

Die Herausforderung bei der Interaktion mit geschachtelten Strukturen liegt darin, Benutzern die Interaktion mit Objekten zu ermöglichen, die von anderen Objekten umschlossen sind. Dies ist auf drei verschiedene Arten (vgl. Abschnitt 6.3.2) möglich: das am weitesten eingeschachtelte Objekt aus der Liste der vom Raycasting getroffenen Objekte wird direkt ausgewählt, es wird ein Kontextmenü angeboten, aus welchem eines der vom Strahl getroffenen Objekte ausgewählt werden kann oder es werden die äußeren Objekte schrittweise entfernt, bis das in Betracht kommende Objekt direkt zugreifbar ist.

#### Der verwendete Editor

Die 3D-Szene des für das Experiment genutzten Editors (siehe Abbildung 7.8) besteht aus einem umgebenden roten Quader in dem sich vier Kugeln befinden, die einen roten und weitere blaue Würfel enthalten. Die Aufgabe der Probanden war es, den im Inneren der grünen Kugeln enthaltenen roten Würfel auszuwählen und Attribut-Werte aus dem Eigenschaften-Dialog des Würfels auszulesen.

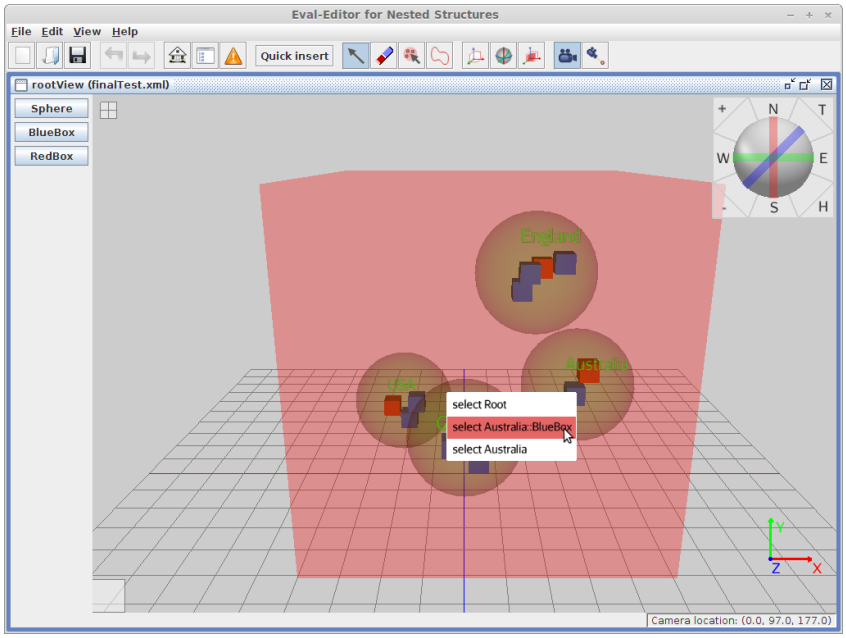



**Abbildung 7.8:** Editor zum Testen der Interaktion mit geschachtelten Strukturen.

Zum Zeitpunkt der Durchführung der ersten Experimente war die direkte Auswahl geschachtelter Objekte noch nicht implementiert. Daher wurde zunächst das Auswählen mithilfe des Kontextmenüs mit dem schrittweisen Entfernen der äußeren Objekte verglichen. Zu einem späteren Zeitpunkt wurde das Experiment erneut durchgeführt und dabei die direkte Auswahl mit dem Auswählen mittels Kontextmenü verglichen.

### Die Ergebnisse

Beim ersten Experiment lösten beide Gruppen die Aufgabe ausnahmslos korrekt. Bei den ermittelten Zeiten ergab sich allerdings kein signifikanter Unterschied. Die Gruppe, die durch das Kontextmenü das Objekt auswählte, brauchte im Mittel 16,2 Sekunden (Standardabweichung = 5,83 Sek., Median = 17 Sek.), wohingegen die Gruppe, die die umgebenden Objekte entfernen musste, 16,9 Sekunden (Standardabweichung = 2,37 Sek., Median=17 Sek.) benötigte. Am Ende dieses Tests wurden die Probanden gefragt, wie aufwendig sie die Aufgabe empfanden. Die Antworten gaben sie auf einer Likert-Skala von „überhaupt nicht aufwendig“ bis „sehr aufwendig“. Dabei ergaben sich nur geringe Unterschiede, die leicht darauf hindeuten, dass die Auswahl per Kontextmenü präferiert wird.

Das zweite Experiment wurde gemäß eines Mehr-Faktor-Plans (siehe Abschnitt 7.3.3) durchgeführt. Jeder Gruppe wurden zwei verschiedene, aber vergleichbare Aufgaben gegeben, bei denen die eingeschachtelten Objekte entweder direkt oder

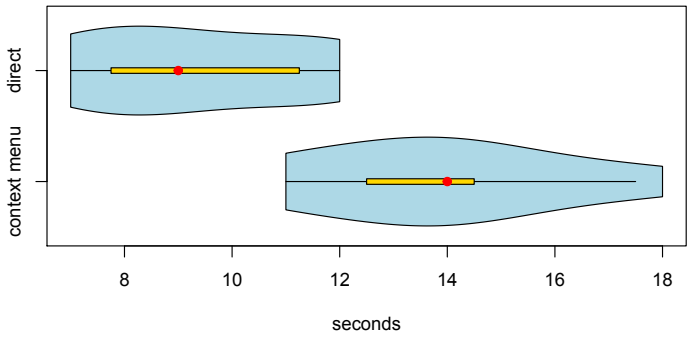



**Abbildung 7.9:** Benötigte Zeiten zur Interaktion mit geschachtelten Strukturen mittels Kontextmenü und direkt.

über das Kontextmenü ausgewählt werden mussten. Bei beiden Aufgaben schnitt die direkte Interaktion besser ab. Für Aufgabe A benötigten die Probanden mit der direkten Interaktion im Mittel 9,25 Sekunden (Standardabweichung = 2,63 Sek., Median = 9 Sek.) gegenüber 14,75 Sekunden (Standardabweichung = 2,22 Sek., Median = 14 Sek.) mithilfe des Kontextmenüs. Bei Aufgabe B ergaben sich Mittelwerte von 9,5 Sekunden(Standardabweichung = 1,73 Sek., Median = 9 Sek.) für die direkte Interaktion und 13 Sekunden (Standardabweichung = 2,45 Sek., Median = 12,5 Sek.) für Interaktion mit dem Kontextmenü. Abbildung 7.9 zeigt die Summe der Zeiten aus den Aufgaben A und B. Nach dem t-Test nach Welch ergibt sich für diese summierten Zeiten ein $p$-Wert von 0,0012. Betrachtet man Aufgabe A separat, ergibt sich ein $p$-Wert von 0,0194; bei Aufgabe B ein $p$-Wert von 0,063. Bei Annahme des üblichen Signifikanzniveaus $\alpha = 5\,\%$, ergibt sich bei Aufgabe B alleine kein signifikanter Vorteil der direkten Interaktion; bei Aufgabe A und der Summe der beiden Aufgaben hingegen schon.

Den Aufwand schätzten die Probanden bei der direkten Interaktion geringer ein als bei Interaktion mit dem Kontextmenü. Außerdem präferierten 75 % die direkte Interaktion. Die 25 %, die gerne das Kontextmenü verwendeten, präferierten es mit dem Argument, dass damit eine Auswahl leichter falle, wenn die eingeschachtelten Objekte fast so groß sind wie die sie umschließenden Objekte.

### 7.3.7 Experiment: Simultane Mehrfachselektion

Zur simultanen Auswahl mehrerer Objekte bieten mit DEViL3D generierte Editoren mit der Zylinder- und Lasso-Auswahl zwei verschiedene Möglichkeiten an.

#### Der verwendete Editor

Der im Experiment verwendete Editor ist in Abbildung 7.10 zu sehen. Die 3D-Sicht besteht aus verschiedenfarbigen Würfeln. Die Aufgabe der Probanden war

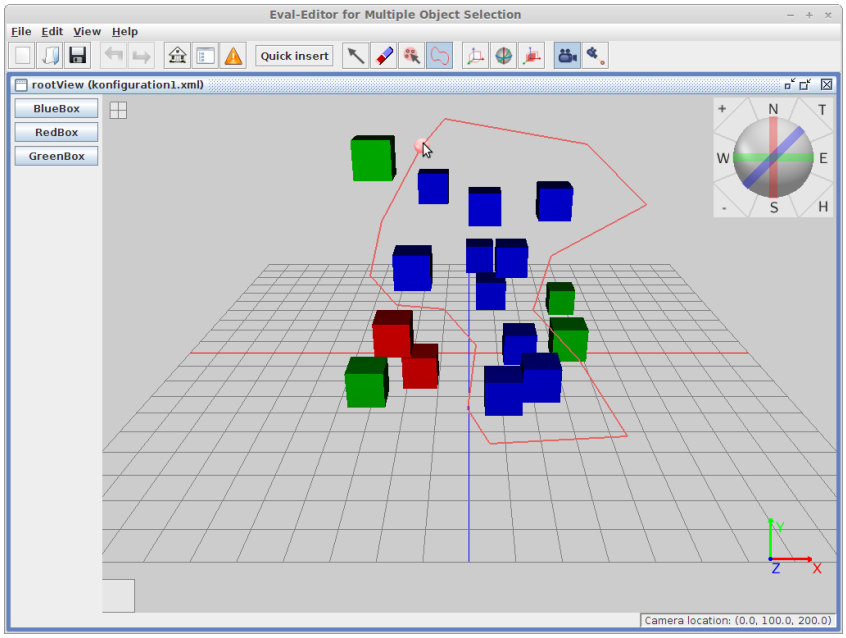



**Abbildung 7.10:** Editor zum Testen der Mehrfachselektion.

es, jeweils die blauen Würfel zu markieren und anschließend zu löschen. Die eine Gruppe verwendete ausschließlich die Zylinder-, die andere ausschließlich die Lasso-Selektion. Das Löschen musste nicht mit einer einzigen Auswahl geschehen, sondern durfte in mehreren Auswahloperationen erfolgen.

### Die Ergebnisse

Die Gruppe, die die Mehrfachselektion mittels Zylinder-Metapher durchführten, benötigten im Mittel 11,1 Sekunden (Standardabweichung = 2,47 Sek., Median = 10 Sek.) zum Lösen der Aufgabe. Mithilfe der Lasso-Metapher brauchten die Probanden durchschnittlich 17,44 Sekunden (Standardabweichung = 4,22 Sek., Median = 18 Sek.). Nach dem Welch t-Test ist die Zylinder-Metapher signifikant ($p$-Wert = 0,00189) schneller, was auch Abbildung 7.11 verdeutlicht. Die Aufgabe wurde von beiden Gruppen korrekt gelöst.

Die Anzahl der durchgeführten Auswahloperationen unterschied sich zwischen beiden Gruppen: Die Gruppe, die den Zylinder verwendet hat, vollzog im Schnitt 5,4 Auswahloperationen (Standardabweichung = 1,81 Sek.; Median = 6 Sek.); verglichen mit durchschnittlich 2,4 Auswahloperationen (Standardabweichung = 0,53 Sek.; Median = 2 Sek.) bei der Lasso-Metapher. Trotz mehr Auswahloperation war die Benutzung des Zylinders deutlich schneller. Der Grund dafür wird sein, dass die Benutzung des Lassos mehr Genauigkeit erfordert, mit dem Zylinder kann hingegen schneller ausgewählt werden, da evtl. nur die Größe seiner Grundflä-

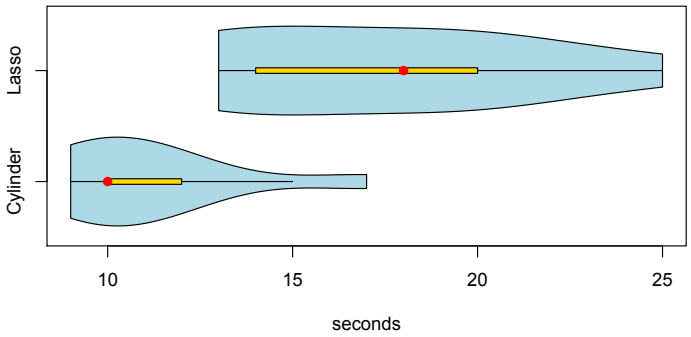



**Abbildung 7.11:** Benötigte Zeit zum Mehrfachauswählen von Objekten mit der Zylinder- und Lasso-Metapher.

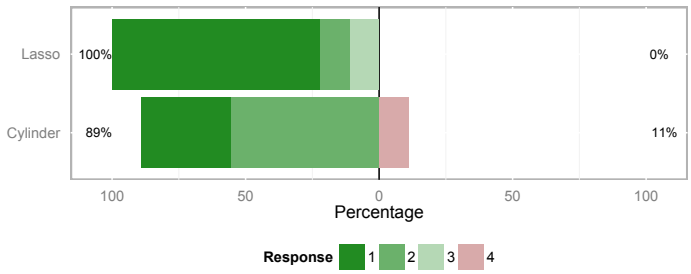



**Abbildung 7.12:** Anhand einer Likert-Skala ermittelte Einschätzung zur Benutzbarkeit der Mehrfachselektion. Der Wert 1 steht auf der Skala für „sehr benutzbar“ und der Wert 6 für „überhaupt nicht gut benutzbar“.

che angepasst werden muss, nicht allerdings wie beim Lasso von Grund auf eine geometrische Form gezeichnet werden muss. Auf einer Likert-Skala mussten die Probanden angeben, für wie gut benutzbar sie die verwendete Selektionstechnik einschätzen. Dabei schnitt die Lasso-Metapher geringfügig besser ab (siehe Abbildung 7.12).

### 7.3.8 Experiment: 3D-Eindruck mit stereoskopischer Hardware

Wie in Abschnitt 6.8 beschrieben, lassen sich die 3D-Sichten der mit DEViL3D generierten Editoren – entsprechende Hardwareausstattung vorausgesetzt – neben der monoskopischen Darstellung auch stereoskopisch anzeigen. Subjektiv betrachtet vermittelt die stereoskopische Darstellung mit dem Shutter-3D-Verfahren einen besseren räumlichen Eindruck von der 3D-Szene als die monoskopische Darstellung.

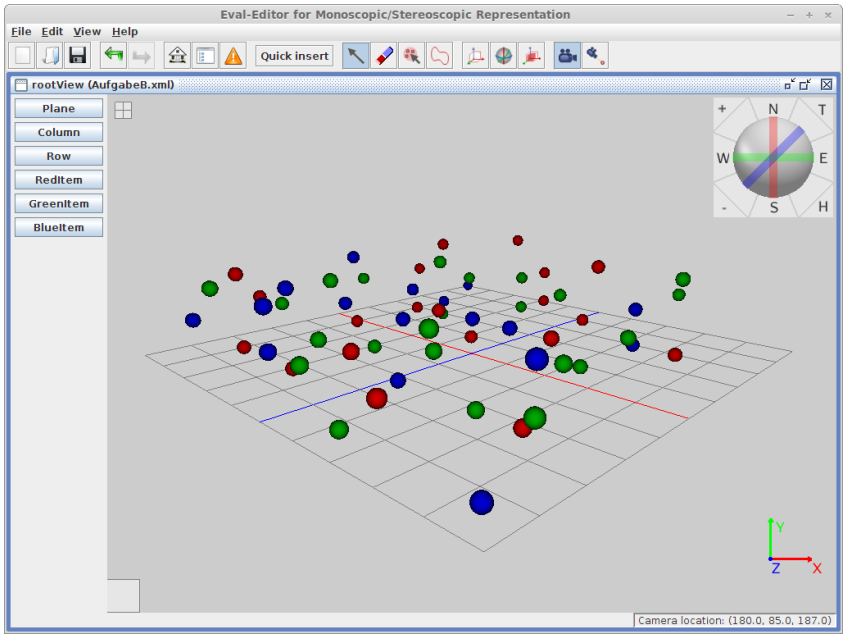


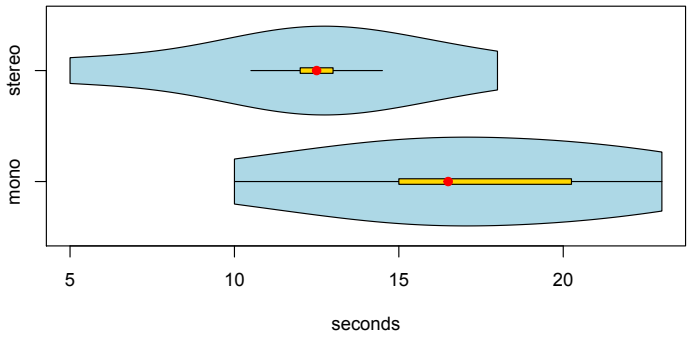

**Abbildung 7.13:** Editor zum Testen der monoskopischen bzw. stereoskopischen Darstellung.

### Der verwendete Editor

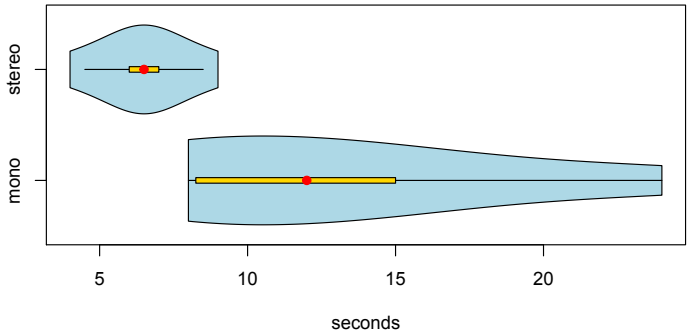
Zum Testen des 3D-Eindrucks wurde ein 3D-Editor generiert (siehe Abbildung 7.13), der das Quader-Muster instanziiert und in dessen Zellen sich zufällig platzierte Kugeln in drei verschiedenen Farben befinden. Der Quader besteht aus einer $10 \times 10$ Zellen großen Grundfläche und drei Ebenen. Die Aufgabe der Probanden war in Aufgabe A die Anzahl der grünen Kugeln in der zweiten Ebenen zu ermitteln. Bei Aufgabe B (in Abbildung 7.13 dargestellt) mussten die roten Kugeln in der untersten Ebene gezählt werden. Beide Aufgaben mussten nach dem Schema aus Abschnitt 7.3.3 entweder in monoskopischer oder stereoskopischer Darstellung mithilfe von Shutter-Brillen gelöst werden. Aufgabe B ist etwas einfacher einzuschätzen, da das auf der untersten Ebene befindliche Gitter eine gute Einschätzung der räumlichen Position bietet. Dieser Vorteil ist beim Zählen von Kugeln auf der mittleren Ebene nicht vorhanden.

### Die Ergebnisse

Für das Zählen der grünen Kugeln der mittleren Ebene in Aufgabe A benötigten die Probanden mit der stereoskopischen Darstellung im Mittel 12,2 Sekunden (Standardabweichung = 4,17 Sek.; Median = 12,5 Sek.) und mit der monoskopischen Darstellung 17 Sekunden (Standardabweichung = 4,7 Sek.; Median = 16,5 Sek.). Abbildung 7.14a visualisiert dies in einem Violin-Plot. Allerdings ist der gemes-




**(a)** Benötigte Zeit zum Lösen von Aufgabe A.




**(b)** Benötigte Zeit zum Lösen von Aufgabe B.

**Abbildung 7.14:** Benötigte Zeit zum Zählen von Kugeln unter Verwendung der stereoskopischen bzw. monoskopischen Darstellung.

sene Unterschied der beiden Darstellungen nach Welchs t-Test mit einem $p$-Wert von 0,0889 nicht signifikant. Allerdings lösten 83 % der Probanden, die die stereoskopische Darstellung nutzten, Aufgabe A korrekt. Von den Probanden, die die monoskopische Darstellung nutzten, löste nur die Hälfte die Aufgabe korrekt.

Bei Aufgabe B benötigten die Probanden zum Zählen der roten Kugeln auf der untersten Ebene im Mittel 6,5 Sekunden (Standardabweichung = 1,64 Sek.; Median = 6,5 Sek.), verglichen mit 13,2 Sekunden (Standardabweichung = 6,24 Sek.; Median = 12 Sek.) bei Benutzung der monoskopischen Darstellung. Dies visualisiert Abbildung 7.14b. Nach Welchs t-Test ist die Benutzung der stereoskopischen Darstellung signifikant schneller ($p$-Wert = 0,046). Zwei Drittel der Probanden, die die stereoskopische Darstellung nutzten, lösten die Aufgabe korrekt, wohingegen nur die Hälfte keinen Fehler machte, wenn die monoskopische Darstellung genutzt wurde.

Bzgl. der Korrektheit der gelösten Aufgaben ist auffällig, dass bei Verwendung der monoskopischen Darstellung bei beiden Aufgaben nur 50 % der Probanden keine Fehler machten. Bei Verwendung der stereoskopischen Darstellung wurde die Aufgabe mehrheitlich richtig gelöst. Den beiden Plots aus Abbildung 7.14

ist außerdem gut zu entnehmen, dass die mittleren 50 % der Daten – dargestellt durch den gelben Balken – bei Benutzung der stereoskopischen Darstellung nur gering um den Median streuen. Bei der monoskopischen Darstellung zeigt sich eine deutlich breitere Verteilung. Daraus lässt sich schlussfolgern, dass die Verwendung stereoskopischer Techniken die Zuverlässigkeit der Ergebnisse erhöht.

Die Auswertung der auf einer Likert-Skala gegebenen Antworten ergab, dass die Probanden das Lösen der beiden Aufgaben mit der stereoskopischen Darstellung als weniger aufwendig empfanden. Eine abschließende Frage, welche der Darstellungen den besseren 3D-Eindruck vermittelt, ging eindeutig zugunsten der stereoskopischen Darstellung aus, für die sich alle Probanden entschieden.

### 7.3.9 Diskussion der Ergebnisse

Die Durchführung der Experimente verlief insgesamt problemlos und die statistische Auswertung ergab überwiegend eindeutige Ergebnisse, die z. B. aussagen, dass eine gegebene Aufgabe mit einer Technik signifikant schneller erledigt werden kann als mit einer anderen. Zu berücksichtigen ist allerdings, dass die Anzahl der Versuchsteilnehmer mit 18 im ersten und zwölf im zweiten Durchgang relativ klein war. Durch den Ein-Faktor-Plan im ersten Durchgang ergaben sich für jede Technik neun Messwerte. Der Mehr-Faktor-Plan im zweiten Durchgang lieferte für jede Versuchsperson zwei Datenpunkte, sodass jede Technik mit zwölf Messwerten statistisch evaluiert werden konnte. Um absolut verlässliche Aussagen zu erhalten, wären eine Vielzahl mehr Probanden notwendig gewesen. Bei der Interpretation der hier präsentierten Ergebnisse muss dies berücksichtigt werden.

Nichtsdestotrotz konnten die fünf adressierten Experimentfragen überwiegend eindeutig beantwortet werden. Bei den Navigationstechniken konnten die Probanden die Aufgabe mit der Navigation-Sphere im freien Modus schneller erledigen als im achsenbasierten Modus. Die Verwendung der Lateral-Views ergab bei den Probanden einen zusätzlichen Zeitaufwand verglichen mit der Gruppe, die die normale Navigation nutzte. Bei der Interaktion mit eingeschachtelten Objekten war die direkte Auswahl derjenigen mit Kontextmenü überlegen. Bei der Mehrfachselektion konnten sowohl die Zylinder- als auch die Lasso-Metapher von den Probanden einfach bedient werden, wobei die Auswahl mit der Zylinder-Metapher prinzipbedingt schneller vollzogen werden konnte. Das Experiment, welches stereoskopische und monoskopische Repräsentation des 3D-Editors verglichen hat, deutet auf Vorteile bei der stereoskopischen Darstellung hin, auch wenn diese nur bei einer der zwei durchgeführten Aufgaben statistisch signifikant war. Die Rückmeldung der Probanden war hingegen durchgehend positiv, da diese einen subjektiv besseren 3D-Eindruck mit der Shutter-Brille verglichen mit der monoskopischen Darstellung wahrgenommen haben.

Im Nachhinein hätten einige Details der Experimente verbessert werden können. So hätte bei dem Experiment zur Navigation (siehe Abschnitt 7.3.4) – zusätzlich zu den beiden Modi der Navigation-Sphere – die Navigation mittels Tastatur mit getestet werden können, anstatt nur am Ende des Experiments den Probanden die Tastatur-Navigation kurz vorzuführen und ihre informelle Einschätzung dazu zu erhalten. Ein Mehr-Faktor-Plan mit drei Niveaus hätte auch explizite Daten erbracht, wie lange die Navigation mittels Tastatur dauert und hätte somit direkt mit den beiden anderen Modi verglichen werden können. Ein Nutzen der Lateral-Views konnte in dem Experiment (siehe Abschnitt 7.3.5) nicht direkt nachgewiesen werden. Allerdings bin ich durchaus überzeugt, dass die Lateral-Views in einigen Situationen hilfreich sind. Soll z. B. ein 3D-Mengenelement eingefügt werden, ist die exakte Position der verschiebbaren Einfügeebene durch die Draufsicht der Lateral-Views besser ersichtlich. Wenn Benutzer den Nutzen der Lateral-Views in solchen Situationen kennen, können sie gezielt und gewinnbringend eingesetzt werden. Dies ließe sich theoretisch in einer Langzeitstudie ermitteln, bei der Probanden regelmäßig mit von DEViL3D generierten Editoren arbeiten und die Benutzung der Lateral-Views erlernen. In einem an die Lernphase angeschlossenen Experiment könnte evtl. ein Nutzen der Lateral-Views messbar werden.

Alle hier vorgestellten Tests umfassen nur kleine abgeschlossene Aufgaben, die von den zwei Experimentgruppen mit zwei konkurrierenden Techniken gelöst wurden. Ziel war dabei unter kontrollierten Bedingungen herauszufinden, mit welcher der Techniken sich die Aufgabe besser und schneller erledigen lässt. Darüber hinaus können weniger feingranulare Experimente wünschenswert sein, die praxisnähere Aufgaben vorsehen, die eine (möglicherweise iterative) Konstruktion und Manipulation größerer 3D-Diagramme vorsehen. Bei solchen Aufgaben bekommt natürlich die zugrundeliegende Sprache einen größeren Einfluss. Dies erhöht allerdings auch die Anzahl der Störvariablen und außerdem ist die Wahl einer unabhängigen Variable bei komplexen Aufgaben nicht einfach. Daher würden sich eher Feld-Beobachtungen zur Untersuchung komplexerer Aufgaben anbieten. Damit lassen sich sicherlich weitere nützliche Hinweise zur Verbesserung der Usability finden.

## 7.4 Verwandte Arbeiten

Das Vorgängersystem DEViL wurde mit der auch hier vorgenommenen Zweiteilung in Usability des Generators und Usability der generierten Editoren evaluiert (siehe [Sch06, S. 201ff.] und [SCK07]). Viele der Erkenntnisse bzgl. der Usability des Generators sind auch für DEViL3D relevant, da viele Spezifikationen syntaktisch ähnlich sind. Darüber hinaus wurde in [Cra10, S. 149ff.] die DEViL-Spezifikation für die Simulation evaluiert.

Die Nützlichkeit von Übersichtskarten oder Kompassen bei der Navigation in einer 3D-Szene konnten Haik et al. [HBST02] nachweisen.

Die Darstellung von 3D-Szenen mit stereoskopischen Methoden ist in zahlreichen Studien evaluiert worden. McIntire und Liggett [ML14] geben einen guten Überblick über 3D-Stereoskopie-Ansätze im Bereich von Informationsvisualisierungen. Sie heben besonders hervor, dass stereoskopische Darstellungen nützlich sind, um 3D-Konstrukte besser zu identifizieren, zu manipulieren und mit anderen Objekten zu vergleichen. Volbracht et al. [VSDF96] vergleichen in einer Applikation für 3D-Molekülmodelle die monoskopische Darstellung mit stereoskopischen Repräsentationen (Shutter-3D und Anaglyph-3D) und kommen zu dem Ergebnis, dass die gestellten domänenspezifischen Aufgaben mit stereoskopischer Darstellung schneller und korrekter gelöst werden konnten. Die Studie von Hubona et al. [HSF97] weist empirisch eine gesteigerte Tiefenwahrnehmung bei Verwendung der Stereoskopie nach. Dabei verringerten sich die Antwortzeit und die Fehlerrate gegenüber einer monoskopischen Darstellung signifikant.

KAPITEL 8

# Fazit

Im Folgenden werde ich die wichtigsten Erkenntnisse resümieren, die ich bei der Entwicklung von Methoden und Werkzeugen für dreidimensionale visuelle Sprachen gewonnen habe. Es ist ein generierenden Systems entwickelt worden, welches 3D-Editoren für verschiedene dreidimensionale visuelle Sprachen automatisiert herstellt. Dabei konnte ich auf Erkenntnisse aus der Generierung von Editoren für 2D-Sprachen zurückgreifen, dort bewährte Konzepte nutzen und für den dreidimensionalen Anwendungsfall erweitern. Im folgenden Abschnitt stelle ich zunächst die gewonnenen Erkenntnisse und Ergebnisse vor. Anschließend gebe ich einen Ausblick auf lohnenswerte zukünftige Forschungsthemen.

## 8.1 Ergebnisse und Erkenntnisse

In dieser Arbeit habe ich zunächst systematisch dreidimensionale visuelle Sprachen analysiert und deren Beschreibungsmittel herausgearbeitet. Die Sprachkonstrukte einiger Sprachen sind an Objekte aus der realen Welt angelehnt, wobei die Mehrzahl der Sprachen Sprachkonstrukte abstrakter Form nutzen. Das Prinzip der Schachtelung, bei dem Sprachkonstrukte andere vollständig umschließen, wird von vielen Sprachen verwendet und drückt meist eine Enthaltensein-Relation aus. Weiterhin hat die Anordnung der Sprachkonstrukte im Diagramm eine semantische Bedeutung, die z. B. fest den Dimensionsachsen zugeordnet sind. Mit dem vorgestellten Klassifikationsschema kann weiterhin ermittelt werden, ob eine 3D-Sprache gewinnbringend Gebrauch von der dritten Dimension macht.

Ziel dieser Arbeit war die Entwicklung von Methoden und Werkzeugen, mit denen sich die Implementierung von 3D-Sprachen automatisieren lässt. Dabei konnte auf vorhandene und bewährte Konzepte zurückgegriffen werden, wie sie

für die Entwicklung zweidimensionaler visueller Sprachen im DEViL-System verwendet werden. Insbesondere für das zentrale Konzept der visuellen Muster konnte gezeigt werden, dass die ihm zugrundeliegenden Prinzipien auch verwendet werden können visuelle Darstellungen dreidimensionaler visueller Sprachen zu beschreiben. Ich habe die abstrakten Musterbeschreibungen, aus denen die konkreten visuellen Muster abgeleitet werden, generalisiert und verallgemeinert. Aus den Musterbeschreibungen lassen sich dadurch konkrete visuelle Muster für 3D- als auch 2D-Sprachen ableiten. Weiterhin habe ich einen Satz von konkreten visuellen Mustern für 3D-Sprachen entwickelt, der zur Beschreibung der in der Arbeit vorgestellten 3D-Sprachen mächtig genug ist.

Zentral bei der Konstruktion dreidimensionaler Diagramme im Strukturediteor sind gut benutzbare Interaktions- und Navigationstechniken. In dieser Arbeit mussten solche Techniken nicht einfach für eine bestimmte Sprache entwickelt werden, sondern Anforderung war, die Techniken in einer ganzen Reihe von Strukturediteoren für verschiedene 3D-Sprachen zu verwenden. Dafür habe ich erfolgreich Interaktions- und Navigationstechniken, die sich in bestehenden 3D-Editoren bewährt haben, generisch verfügbar gemacht und in den visuellen Mustern gekapselt. Das stellt sicher, dass die Techniken immer an die Erfordernisse der visuellen Darstellung angepasst sind.

Bedeutend für den visuellen Eindruck eines Diagramms ist dessen Layout. Es wurden vier verschiedene Kategorien von Layouts identifiziert: musterspezifisches, benutzerspezifisches, sprachspezifisches Layout sowie das Layout allgemeiner Natur. Zur letzten Kategorie zählen das Nichtdurchdringen oder Andocken von Sprachkonstrukten, wofür verschiedene generische Methoden entwickelt wurden. Grundsätzlich sind die Layouteigenschaften in den visuellen Mustern gekapselt. Bei manchen Sprachen sind die Layoutanforderungen so spezifisch, dass generische Layoutmethoden nicht ausreichen. Dafür habe ich Methoden zur Sicherstellung von sprachspezifischem Layout entwickelt, die z. B. ermöglichen, das Layout von Molekülmodellen gemäß chemischer Regeln zu beschreiben. Es hat sich gezeigt, dass das Layout einer Sprache bei fortlaufender Schachtelung von Sprachkonstrukten unübersichtlich werden kann. Die Interaktion mit solchen geschachtelten Strukturen ist zwar durch darauf spezialisierte Techniken sichergestellt, allerdings bleibt die Unübersichtlichkeit bei einer Schachtelungstiefe von typischerweise mehr als drei Sprachkonstrukten.

Zur automatisierten Entwicklung einer großen Bandbreite visueller 3D-Sprachen wurde das Generatorsystem DEViL3D entwickelt, welches die oben beschriebenen Methoden umfasst. DEViL3D generiert aus der Gesamtheit von Sprachspezifikationen, die Spezifizierer einer Sprache auf hohem Niveau formulieren, einen 3D-Strukturediteor.

Ein schönes Beispiel für die Mächtigkeit des verwendeten Generatorprinzips manifestiert sich am Beispiel der Verwendung stereoskopischer Verfahren zur Darstellung der 3D-Szene. Die Unterstützung dafür wurde innerhalb des letzten Drittels der Bearbeitungsperiode implementiert (siehe Abschnitt 6.8). Allerdings profitieren danach alle, auch die zuvor spezifizierten, 3D-Sprachen von dieser Funktionalität, da nach einer Neugenerierung der Spezifikationen die stereoskopische Darstellung automatisch verfügbar ist.

Bei der Evaluation dieser Arbeit habe ich den Fokus auf die vom DEViL3D-System generierten Strukturediteren gelegt. Anspruch ist, dass die Funktionalität, wie man sie von guten manuell entwickelten Editoren kennt, auch von den generierten Editoren geboten wird. Die im Vorhinein formulierten Experimentfragen konnten meist eindeutig beantwortet werden, wodurch sich Interaktions- und Navigationstechniken herauskristallisiert haben, die von den meisten Probanden eindeutig bevorzugt werden. So wurde auch die stereoskopische Darstellung der 3D-Szene von allen Teilnehmern präferiert, wenn es darum geht einen realistischen 3D-Eindruck zu gewinnen.

## 8.2 Ausblick

Das entwickelte Generatorsystem DEViL3D ist in der Lage für eine Reihe verschiedenartiger 3D-Sprachen zuverlässig Editoren zu generieren, mit denen sich Diagramme im 3D-Raum konstruieren lassen. Nichtsdestotrotz halte ich folgende weitere Forschungsarbeiten für lohnenswert.

Das Spektrum der in dieser Arbeit vorgestellten und spezifizierten 3D-Sprachen ist sicherlich nicht vollständig. Aber mit DEViL3D steht nun ein Generatorsystem zur Verfügung, welches ermöglicht, relativ schnell neue Sprachen zu spezifizieren. Vorstellbar sind Sprachen aus einer Architektur-Domäne, mit denen sich z. B. Schranksysteme strukturiert konstruieren lassen. In der Literatur gibt es außerdem verschiedene Ansätze *Geschäftsprozesse* in 3D darzustellen [Zha12; EKL+09; SBE00]. Teilweise beruhen diese Darstellungen auf Petri-Netzen, sodass die bereits spezifizierte Sprache für 3D-Petri-Netze wiederverwendet werden könnte.

Weiterhin können die dreidimensionalen Sprachstile, die 2D-Darstellungen in den 3D-Raum einbetten, grundsätzlicher untersucht und durch Implementierungen weiterer Sprachen praktisch erprobt werden. In dieser Arbeit wurde eine 3D-Sprache für UML-Klassendiagramme vorgestellt, die parametrisierte Klassen gemäß ihrer Anzahl an Parametern auf verschiedenen Ebenen gruppiert. Dieser Ansatz lässt sich bei Klassendiagrammen auch anwenden, um Klassen mit bestimmter Paketzugehörigkeit oder bestimmtem Stereotyp zu gruppieren. Außerdem kann die von Radfelder und Gogolla [RG00] präsentierte Idee, Klassen-

und Sequenzdiagramme in einer 3D-Sicht zu kombinieren, umgesetzt werden, um statische und dynamische Aspekte des Softwareentwurfs gemeinsam darzustellen.

Die visuelle Darstellung der in dieser Arbeit vorgestellten 3D-Sprachen können alle mit den vorgestellten visuellen Mustern beschrieben werden. Allerdings können sich bei der Entwicklung neuer 3D-Sprachen neue Anforderungen an die visuelle Repräsentation ergeben, die von den vorhandenen visuellen Mustern nicht unterstützt wird. Aus diesem Grund ist es recht wahrscheinlich, dass die Menge der visuellen Muster noch erweitert werden muss.

Eine für visuelle 2D-Sprachen etablierte und auch für 3D-Sprachen gewinnbringende Erweiterung ist die Simulation und Animation ihrer visuellen Diagramme (vgl. [CK09; SM10]). Dies ist besonders bei Diagrammen sinnvoll, aus deren statischer Repräsentation nicht unmittelbar deren dynamische Ausführung nachvollziehbar ist. Bei inhärenten 3D-Sprachen aus realweltlichen Domänen kann so deren natürliches Ablaufkonzept nachempfunden werden. Ein gutes Beispiel dafür ist die Simulation von Straßenverkehr, die auch vom AgentCubes-System [IRW09] aufgegriffen wurde (siehe Seite 20). Eine weitere Anwendungsidee ist die von Gogolla et al. [GRR99; RG00] vorgeschlagene Animation des Nachrichtenflusses in 3D-Sequenzdiagrammen. Um DEViL3D um Unterstützung für Simulation und Animation zu erweitern, bietet sich das von Cramer [Cra10] entwickelte Konzept an, welches im Kontext von DEViL entwickelt wurde und ermöglicht, 2D-Sprachen mit Simulationsunterstützung auszurüsten. Da DEViL3D auf dem gleichen konzeptionellen Fundament beruht wie DEViL, sollte die von Cramer entwickelte Methodik übertragen werden können. Der entwickelte Ansatz stellt eine domänenspezifische Sprache zur Beschreibung der Simulation bereit, die eine Transformation des semantischen Modells der Sprache beschreibt. Aus dieser wird automatisch eine Animation generiert, die die diskreten Transformationen kontinuierlich visualisiert. Der Sprachspezifizierer hat durch Anwendung von animierten visuellen Mustern die Möglichkeit auf die Ausprägung der Animation Einfluss zu nehmen.

Das DEViL3D-System ist auf allen drei großen Betriebssystemen Linux, Mac OS X und Windows lauffähig, sodass auch die generierten Strukturediteren auf diesen Systemen lokal zu betreiben sind. Um die 3D-Spracheditoren noch universeller verfügbar zu machen, ist die zusätzliche Generierung eines webbasierten Frontends wünschenswert. Zur technischen Realisierung des 3D-Webeditors bietet sich z. B. die auf *WebGL*[1] basierende und in *JavaScript* implementierte 3D-Umgebung *Three.js*[2] an. Ein wie bisher generierter Strukturediteor, der auch die Attributauswerter umfasst, die für die Berechnung der visuellen Darstellung zuständig sind, würde dann als Server-Anwendung fungieren. Die Operationen zum Darstellen

[1] `https://www.khronos.org/webgl/`
[2] `http://threejs.org/`

der Sprachkonstrukte müssen dann mittels eines zu entwickelnden Protokolls an den Webeditor auf dem Client weitergeleitet werden. Selbstverständlich muss der Webeditor Funktionen zur Interaktion und Navigation umfassen, die es ermöglichen 3D-Diagramme zu editieren. Änderungen am Diagramm müssen dann an die Serveranwendung geschickt werden, woraufhin diese die Darstellung neu berechnet.

Um die Portabilität der 3D-Strukturediteren darüber hinaus zu erhöhen, ist es interessant 3D-Strukturediteren auch auf *Tablets* benutzbar zu machen. Dabei stellen sich ganz neue Herausforderungen, insbesondere bei der Realisierung von Techniken zur Interaktion und Navigation, die dann ausschließlich durch Bedienung des *Touchscreens* realisiert werden müssten.

Aktuell können mit DEViL3D Editoren für visuelle 3D-Sprachen generiert werden, deren Sichten alle eine 3D-Zeichenfläche bereitstellen. Die von DEViL generierten 2D-Editoren, deren Frontend mit *Tcl* implementiert wurde, leiden – besonders bei Verwendung der Simulation – unter Geschwindigkeitsproblemen [Cra10, S. 177ff.]. Da 3D-Sprachen als echte Obermenge von 2D-Sprachen betrachtet werden können, ist es erstrebenswert mit DEViL3D auch zusätzlich klassische 2D-Sprachen generieren zu können. Diese würden dann von dem moderneren Java-Frontend profitieren und außerdem können so die vielen nützlichen mit DEViL spezifizierten Sprachen erhalten werden. Wenn man den oben angesprochenen Aspekt der Simulation und Animation mit berücksichtigt, kann so das DEViL3D-System DEViL ersetzen. Dies würde auch ermöglichen, Sprachen zu spezifizieren, die 2D-Übersichtssichten enthalten, die eine Grobstruktur eines Diagramms repräsentieren und alle inhärent dreidimensionalen Aspekte mit den hier vorgestellten 3D-Repräsentationen darstellen.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Quelltextverzeichnis

# Literaturverzeichnis


[AF02] Klaus Alfert und Alexander Fronk. „Manipulation of 3-dimensional Visualizations of Java Class Relations“. In: *Proceedings of the World Conference on Integrated Design and Process Technology (IDPT)*. Juni 2002 (referenziert auf Seite 18).

[AFE01] Klaus Alfert, Alexander Fronk und Frank Engelen. „Experiences in 3-Dimensional Visualization of Java Class Relations“. In: *Journal of Integrated Design and Process Science*, 5(3) (2001), S. 91–106 (referenziert auf Seite 18).

[Alb04] Jonathan Albe. *Spezifikation und Generierung graphischer Editoren für dreidimensionale Szenerien*. Diplomarbeit, Universität der Bunderwehr München. Jan. 2004 (referenziert auf Seite 45).

[AWP97] Keith Andrews, Josef Wolte und Michael Pichler. „Information Pyramids: A New Approach to Visualizing Large Hierarchies“. In: *Proceedings of the 8th IEEE Conference on Visualization*. Okt. 1997, S. 49–52 (referenziert auf Seite 35).

[BB94] Margaret M. Burnett und Marla J. Baker. „A Classification System for Visual Programming Languages“. In: *Journal of Visual Languages and Computing*, 5(3) (1994), S. 287–300 (referenziert auf Seite 11).

[BBB+95] Margaret M. Burnett, Marla J. Baker, Carisa Bohus, Paul Carlson, Sherry Yang und Pieter Van Zee. „Scaling Up Visual Programming Languages“. In: *IEEE Computer*, 28(3) (1995), S. 45–54 (referenziert auf Seite 51).

[Ber83] Jacques Bertin. *Semiology of graphics: diagrams, networks, maps*. Madison: University of Wisconsin Press, 1983 (referenziert auf Seite 34).

[BG04] Paolo Bottoni und Antonio Grau. „A Suite of Metamodels as a Basis for a Classification of Visual Languages“. In: *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Sep. 2004, S. 83–90 (referenziert auf Seite 80).

[BGL06] Paolo Bottoni, Esther Guerra und Juan de Lara. „Metamodel-based definition of interaction with visual environments“. In: *Proceedings of the Workshop on Model Driven Development of Advanced User Interfaces*

*(MDDAUI)*. Hrsg. von Andreas Pleuss, Jan Van den Bergh, Heinrich Hussmann, Stefan Sauer und Alexander Boedcher. CEUR Workshop Proceedings. Okt. 2006 (referenziert auf Seite 127).

[BGRT99] Prosenjit Bose, Francisco Gómez, Pedro Ramos und Godfried Toussaint. „Drawing Nice Projections of Objects in Space“. In: *Journal of Visual Communication and Image Representation*, 10(2) (1999), S. 155–172 (referenziert auf Seite 88).

[Bie06] Peter Biermann. „Ein visuelles VR-Programmiersystem mit lokalen Constraints für die interaktive virtuelle Konstruktion“. Dissertation. Universität Bielefeld, Nov. 2006 (referenziert auf Seite 50).

[BKH97] Doug A. Bowman, David Koller und Larry F. Hodges. „Travel in Immersive Virtual Environments: An Evaluation of Viewpoint Motion Control Techniques“. In: *Proceedings of the Virtual Reality Annual International Symposium*. März 1997, S. 45–52 (referenziert auf Seite 117).

[BKLP04] Doug A. Bowman, Ernst Kruijff, Joseph J. LaViola und Ivan Poupyrev. *3D User Interfaces – Theory and Practice*. Boston: Addison-Wesley, 2004 (referenziert auf Seiten 33, 104, 106, 117).

[Bri09] Florian Brieler. „A Generic Approach to the Recognition and Analysis of Sketched Diagrams Using Context Information“. Dissertation. Universität der Bundeswehr München, Juni 2009 (referenziert auf Seiten 12, 44).

[CDP04] Gennaro Costagliola, Vincenzo Deufemia und Giuseppe Polese. „A Framework for Modeling and Implementing Visual Notations With Applications to Software Engineering“. In: *ACM Transactions on Software Engineering and Methodology*, 13(4) (Okt. 2004), S. 431–487 (referenziert auf Seite 38).

[CHM99] Vincent Chung, John Hosking und Rick Mugridge. „Visual Specification of 3D Notations using 3DComposer“. In: *Proceedings of the IEEE Symposium on Visual Languages*. Sep. 1999, S. 198–199 (referenziert auf Seiten 81, 101).

[Chu99] Vincent Wing Hong Chung. *3DComposer – A visual builder for 3D notations*. Master's thesis, University of Auckland, New Zealand. Feb. 1999 (referenziert auf Seite 101).

[CK09] Bastian Cramer und Uwe Kastens. „Animation automatically generated from simulation specifications“. In: *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Sep. 2009, S. 157–164 (referenziert auf Seiten 41, 43, 164).

[CKK08] Bastian Cramer, Dennis Klassen und Uwe Kastens. „Entwicklung und Evaluierung einer Domänenspezifischen Sprache für SPS-Schrittket-

ten“. In: *Proceedings of the Workshop on Domain-Specific Modeling Languages*. März 2008, S. 59–73 (referenziert auf Seite 43).

[CMS88] Michael Chen, S. Joy Mountford und Abigail Sellen. „A Study in Interactive 3-D Rotation Using 2-D Control Devices“. In: *SIGGRAPH Compututer Graphics*, 22(4) (Juni 1988), S. 121–129 (referenziert auf Seite 127).

[CMS99] Stuart K. Card, Jock D. Mackinlay und Ben Shneiderman. *Readings in Information Visualization: Using Vision to Think*. San Francisco: Morgan Kaufmann, 1999 (referenziert auf Seite 34).

[Cra10] Bastian Cramer. „Generierung von Animation und Simulation für graphische Strukturediteren“. Dissertation. Universität Paderborn, Sep. 2010 (referenziert auf Seiten 38–41, 129, 135, 158, 164, 165).

[CSH+92] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik und Andries van Dam. „Three-Dimensional Widgets“. In: *Proceedings of the Symposium on Interactive 3D Graphics*. März 1992, S. 183–188 (referenziert auf Seite 127).

[Dac04] Raimund Dachselt. *Eine deklarative Komponentenarchitektur und Interaktionsbausteine für dreidimensionale multimediale Anwendungen*. Dissertation. Tönning: Der Andere Verlag, 2004 (referenziert auf Seiten 104, 106).

[DMW09] Tim Dwyer, Kim Marriott und Michael Wybrow. „Dunnart: A Constraint-Based Network Diagram Authoring Tool“. In: *Graph Drawing*. Hrsg. von Ioannis G. Tollis und Maurizio Patrignani. Bd. 5417. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, 2009, S. 420–431 (referenziert auf Seite 101).

[Dra12] Marius Dransfeld. *Spezifikation eines graphischen Strukturediteors für dreidimensionale generische Zeichnungen*. Bachelorarbeit, Universität Paderborn. Juni 2012 (referenziert auf Seite 137).

[DV99] Alberto Del Bimbo und Enrico Vicario. „A Visual Formalism for Computational Tree Logic“. In: *Journal of Visual Languages and Computing*, 10(2) (1999), S. 165–187 (referenziert auf Seite 23).

[DW13] Vida Dujmović und Sue Whitesides. „Three-Dimensional Drawings“. In: *Handbook of Graph Drawing and Visualization*. Hrsg. von Roberto Tamassia. CRC Press, 2013. Kap. 14, S. 455–488 (referenziert auf Seiten 87, 88).

[DWS13] Manfred Dangelmaier, Philipp Westner und Frank Sulzmann. „Mixed Reality Environments für die montagegerechte Konstruktion und Montageplanung von komplexen Produkten“. In: *Digitale Produktion*. Hrsg. von Engelbert Westkämper, Dieter Spath, Carmen Constanti-

nescu und Joachim Lentes. Berlin Heidelberg: Springer, 2013, S. 223–240 (referenziert auf Seite 1).

[Dwy01] Tim Dwyer. „Three Dimensional UML Using Force Directed Layout". In: *Proceedings of the Asia-Pacific Symposium on Information Visualisation*. Dez. 2001, S. 77–85 (referenziert auf Seite 18).

[Ead84] Peter Eades. „A Heuristic for Graph Drawing". In: *Congressus Nemerantium*, 42(11) (1984), S. 149–160 (referenziert auf Seite 71).

[EHW97] Peter Eades, Michael E. Houle und Richard Webber. „Finding the Best Viewpoints for Three-Dimensional Graph Drawings". In: *Proceedings of the International Workshop on Graph Drawing*. Hrsg. von Giuseppe DiBattista. Bd. 1353. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, 1997, S. 87–98 (referenziert auf Seiten 87, 88).

[EKL+09] Daniel Eichhorn, Agnes Koschmider, Yu Li, Andreas Oberweis, Peter Stürzel und Ralf Trunko. „3D Support for Business Process Simulation". In: *33rd Annual IEEE International Computer Software and Applications Conference*. Juli 2009, S. 73–80 (referenziert auf Seite 163).

[Fár48] István Fáry. „On straight-line representation of planar graphs". In: *Acta Sci. Math. (Szeged)*, 11 (1948), S. 229–233 (referenziert auf Seite 87).

[FGJ95] Elisabeth Freeman, David Gelernter und Suresh Jagannathan. „In Search of a Simple Visual Vocabulary". In: *Proceedings of the International IEEE Symposium on Visual Languages*. Sep. 1995, S. 302–309 (referenziert auf Seite 23).

[FGJ96] Elisabeth Freeman, David Gelernter und Suresh Jagannathan. „Uniformity of Environment and Computation in MAP". In: *Proceedings of the IEEE Symposium on Visual Languages*. Sep. 1996, S. 130–137 (referenziert auf Seite 23).

[FHZ96] Andrew Forsberg, Kenneth Herndon und Robert Zeleznik. „Aperture Based Selection for Immersive Virtual Environments". In: *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*. Nov. 1996, S. 95–96 (referenziert auf Seite 127).

[Fra89] Paul Franchi-Zannettacci. „Attribute Specifications for Graphical Interface Generation". In: *Proceedings of the 11th World Computer IFIP Congress*. Aug. 1989, S. 149–155 (referenziert auf Seite 100).

[GK98] Joseph Yossi Gill und Stuart Kent. „Three Dimensional Software Modelling". In: *Proceedings of the International Conference on Software Engineering*. Apr. 1998, S. 105–114 (referenziert auf Seite 18).

[Gla93] P. Glaister. „Calculating the Tetrahedral Bond Angle Using Spherical Polars and the Dot Product". In: *Journal of Chemical Education*, 70(7) (1993), S. 546–547 (referenziert auf Seite 97).

[Gli87] Ephraim P. Glinert. „Out of Flatland: Towards 3-D Visual Programming“. In: *Proceedings of the Fall Joint Computer Conference on Exploring technology: today and tomorrow*. 1987, S. 292–299 (referenziert auf Seiten 1, 12).

[GLM+96] P. Griebel, G. Lehrenfeld, W. Mueller, C. Tahedl und H. Uhr. „Integrating a Constraint Solver into a Real-Time Animation Environment“. In: *Proceedings of the IEEE Symposium on Visual Languages*. Sep. 1996, S. 12–19 (referenziert auf Seite 101).

[GLW06] Orla Greevy, Michele Lanza und Christoph Wysseier. „Visualizing Live Software Systems in 3D“. In: *Proceedings of the ACM Symposium on Software Visualization (SoftVis)*. Sep. 2006, S. 47–56 (referenziert auf Seite 36).

[GMR98] Christian Geiger, Wolfgang Müller und Waldemar Rosenbach. „SAM – An Animated 3D Programming Language“. In: *Proceedings of the IEEE Symposium on Visual Languages*. Sep. 1998, S. 228–235 (referenziert auf Seiten 1, 14).

[GP96] T. R. G. Green und Marian Petre. „Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework“. In: *Journal of Visual Languages and Computing*, 7(2) (1996), S. 131–174 (referenziert auf Seiten 74, 85, 130).

[GR05] Ronald J. Gillespie und Edward A. Robinson. „Models of molecular geometry“. In: *Chemical Society Reviews*, 34(5) (2005), S. 396–407 (referenziert auf Seiten 21, 95).

[Gra98] Calum A.M. Grant. „Visual Language Editing Using a Grammar-Based Visual Structure Editor“. In: *Journal of Visual Languages and Computing*, 9(4) (1998), S. 351–374 (referenziert auf Seiten 81, 100).

[GRR99] Martin Gogolla, Oliver Radfelder und Mark Richters. „Towards Three-Dimensional Representation and Animation of UML Diagrams“. In: *Proceedings of the International Conference on The Unified Modeling Language: Beyond the Standard*. Sep. 1999, S. 489–502 (referenziert auf Seiten 18, 137, 164).

[HBST02] Eyal Haik, Trevor Barker, John Sapsford und Simon Trainis. „Investigation into Effective Navigation in Desktop Virtual Interfaces“. In: *Proceedings of the Seventh International Conference on 3D Web Technology*. Feb. 2002, S. 59–66 (referenziert auf Seite 159).

[HSF97] Geoffrey S. Hubona, Gregory W. Shirah und David G. Fout. „The effects of motion and stereopsis on three-dimensional visualization“. In: *International Journal of Human-Computer Studies*, 47(5) (1997), S. 609–627 (referenziert auf Seite 159).

[IRW09] Andri Ioannidou, Alexander Repenning und David C. Webb. „AgentCubes: Incremental 3D end-user development“. In: *Journal of Visual Languages and Computing*, 20(4) (2009), S. 236–251 (referenziert auf Seiten 19, 20, 81, 164).

[JH13] Jacek Jankowski und Martin Hachet. „A Survey of Interaction Techniques for Interactive 3D Environments“. In: *Proceedings of the Eurographics 2013 – STARs (State of the Art Reports)*. Mai 2013, S. 65–93 (referenziert auf Seite 127).

[Joc14] Jesko Jockenhövel. *Der digitale 3D-Film: Narration, Stereoskopie, Filmstil*. Wiesbaden: Springer, 2014 (referenziert auf Seite 1).

[Jor98] Patrick W. Jordan. *An Introduction to Usability*. London: Taylor & Francis Ltd, 1998 (referenziert auf Seiten 130, 131).

[Jun00] Matthias Jung. „Ein Generator zur Entwicklung visueller Sprachen“. Dissertation. Universität Paderborn, Nov. 2000 (referenziert auf Seiten 11, 38, 43, 61, 101).

[Koe14] David Koelle. *JFugue – Java API for Music Programming*. `http://www.jfugue.org/`. [Online; Stand 3. März 2015]. 2014 (referenziert auf Seite 137).

[KP07] Caitlin Kelleher und Randy Pausch. „Using storytelling to motivate programming“. In: *Communications of the ACM*, 50(7) (Juli 2007), S. 58–64 (referenziert auf Seite 22).

[KPJ98] Uwe Kastens, Peter Pfahler und Matthias Jung. „The Eli System“. In: *Compiler Construction*. Bd. 1383. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, 1998, S. 294–297 (referenziert auf Seiten 39, 58).

[Kus13] Ruth Kusterer. *jMonkeyEngine 3.0 – Develop professional 3D games for desktop, web, and mobile, all in the familiar Java programming language*. Birmingham: Packt Publishing, 2013 (referenziert auf Seiten 58, 111).

[LG93] Jiandong Liang und Mark Green. „Geometric Modeling Using Six Degrees of Freedom Input Devices“. In: *Proceedings of the 3rd International Conference on CAD and Computer Graphics*. Aug. 1993, S. 217–222 (referenziert auf Seite 127).

[Mai12] Sonja Maier. „A Pattern-based Approach for the Combination of Different Layout Algorithms in Diagram Editors“. Dissertation. Universität der Bundeswehr München, Juni 2012 (referenziert auf Seiten 45, 100).

[McI98] David W. McIntyre. *Comp.Lang.Visual - Frequently-Asked Questions List*. `http://www.faqs.org/faqs/visual-lang/faq/`. [Online; Stand 3. März 2015]. März 1998 (referenziert auf Seite 51).

[Met15] MetaCase. *MetaEdit+ Domain-Specific Modeling (DSM) environment*. `http://www.metacase.com/products.html`. [Online; Stand 3. März 2015]. MetaCase Consulting, 2015 (referenziert auf Seite 38).

[MHM10] Daniel L. Moody, Patrick Heymans und Raimundas Matulevicius. „Visual syntax does matter: improving the cognitive effectiveness of the i* visual notation". In: *Requirements Engineering*, 15(2) (2010), S. 141–175 (referenziert auf Seite 34).

[Min01] Mark Minas. *Spezifikation und Generierung graphischer Diagrammeditoren*. Habilitation. Aachen: Shaker Verlag, 2001 (referenziert auf Seiten 44, 91).

[Min02] Mark Minas. „Concepts and realization of a diagram editor generator based on hypergraph transformation". In: *Science of Computer Programming*, 44(2) (2002), S. 157–180 (referenziert auf Seiten 38, 44).

[Min06] Mark Minas. „Generating Meta-Model-Based Freehand Editors". In: *Proceedings of the 3rd International Workshop on Graph Based Tools (GraBaTs'06), Natal (Brazil), Satellite event of the 3rd International Conference on Graph Transformation*. Hrsg. von Albert Zündorf und Dániel Varró. Bd. 1. Electronic Communications of the EASST. Sep. 2006 (referenziert auf Seiten 38, 44, 45).

[MK08] Leonel Morgado und Ken Kahn. „Towards a specification of the ToonTalk language". In: *Journal of Visual Languages and Computing*, 19(5) (2008), S. 574–597 (referenziert auf Seite 22).

[ML14] John McIntire und Kristen Liggett. „The (Possible) Utility of Stereoscopic 3D Displays for Information Visualization: The Good, the Bad, and the Ugly". In: *Proceedings of the IEEE VIS International Workshop on 3DVis: Does 3D really make sense for Data Visualization?* Nov. 2014 (referenziert auf Seite 159).

[MM07] Sonja Maier und Mark Minas. „A Pattern-Based Layout Algorithm for Diagram Editors". In: *Proceedings of the Workshop on the Layout of (Software) Engineering Diagrams in conjunction with the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Hrsg. von Harald Störrle Andrew Fish Alexander Knapp. Bd. 7. Electronic Communications of the EASST. Sep. 2007 (referenziert auf Seite 100).

[MM08] Sonja Maier und Mark Minas. „A Generic Layout Algorithm for Metamodel Based Editors". In: *Applications of Graph Transformations with Industrial Relevance*. Hrsg. von Andy Schürr, Manfred Nagl und Albert Zündorf. Bd. 5088. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, 2008, S. 66–81 (referenziert auf Seite 100).

[MM09] Sonja Maier und Mark Minas. „Pattern-Based Layout Specifications for Visual Language Editors". In: *Proceedings of the Workshop on Visu-*

*al Formalisms for Patterns in conjunction with the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Hrsg. von Paolo Bottoni, Esther Guerra und Juan de Lara. Bd. 25. Electronic Communications of the EASST. Sep. 2009 (referenziert auf Seite 100).

[MM10] Sonja Maier und Mark Minas. „Combination of Different Layout Approaches“. In: *Proceedings of the 2nd Workshop on Visual Formalisms for Patterns in conjunction with the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Hrsg. von Paolo Bottoni, Esther Guerra und Juan de Lara. Bd. 31. Electronic Communications of the EASST. Sep. 2010 (referenziert auf Seite 100).

[MM12a] Sonja Maier und Mark Minas. „Integration of a Pattern-Based Layout Engine into Diagram Editors“. In: *Applications of Graph Transformations with Industrial Relevance*. Hrsg. von Andy Schürr, Dániel Varró und Gergely Varró. Bd. 7233. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, 2012, S. 89–96 (referenziert auf Seiten 81, 100).

[MM12b] Sonja Maier und Mark Minas. „Layout Improvement in Diagram Editors by Automatic Ad-hoc Layout“. In: *Proceedings of the 11th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT)*. Hrsg. von Andrew Fish und Leen Lambers. Bd. 47. Electronic Communications of the EASST. März 2012 (referenziert auf Seite 100).

[Mye90] Brad A. Myers. „Taxonomies of Visual Programming and Program Visualization“. In: *Journal of Visual Languages and Computing*, 1(1) (1990), S. 97–123 (referenziert auf Seite 10).

[Naj94] Mark-Alexander Najork. „Programming in Three Dimensions“. Dissertation. University of Illinois at Urbana-Champaign, 1994 (referenziert auf Seite 13).

[Naj96] Marc A. Najork. „Programming in Three Dimensions“. In: *Journal of Visual Languages and Computing*, 7(2) (1996), S. 219–242 (referenziert auf Seiten 1, 13, 27, 49, 50, 126).

[OASS01] Noritaka Osawa, Kikuo Asai, Yuji Y. Sugimoto und Fumihiko Saito. „A Dancing Programmer in an Immersive Virtual Environment“. In: *Proceedings IEEE Symposia on Human-Centric Computing Languages and Environments (HCC)*. Sep. 2001, S. 348–349 (referenziert auf Seite 50).

[OT99] Takashi Oshiba und Jiro Tanaka. „“3D-PP”: Visual Programming System with Three-Dimensional Representation“. In: *Proceedings of the International Symposium on Future Software Technology*. Okt. 1999, S. 61–66 (referenziert auf Seiten 1, 15, 16, 24).

[PCJ97] Helen C. Purchase, R.F. Cohen und M.I. James. „An Experimental Study of the Basis for Graph Drawing Algorithms“. In: *ACM Journal of Experimental Algorithmics (JEA)*, 2 (Jan. 1997) (referenziert auf Seite 87).

[PD08] Jens von Pilgrim und Kristian Duske. „GEF3D: a Framework for Two-, Two-and-a-Half-, and Three-Dimensional Graphical Editors“. In: *Proceedings of the ACM Symposium on Software Visualization*. Sep. 2008, S. 95–104 (referenziert auf Seite 18).

[PD10] Bernhard Preim und Raimund Dachselt. *Interaktive Systeme – Band 1: Grundlagen, Graphical User Interfaces, Informationsvisualisierung*. Berlin Heidelberg: Springer, 2010 (referenziert auf Seite 106).

[Pre01] Lutz Prechelt. *Kontrollierte Experimente in der Softwaretechnik: Potential und Methodik*. Berlin Heidelberg: Springer, 2001 (referenziert auf Seite 132).

[PTT97] János Pach, Torsten Thiele und Géza Tóth. „Three-dimensional grid drawings of graphs“. In: *Proceedings of the International Workshop on Graph Drawing*. Hrsg. von Giuseppe DiBattista. Bd. 1353. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, 1997, S. 47–51 (referenziert auf Seite 87).

[RF93] Frank Van Reeth und Eddy Flerackers. „Three-dimensional Graphical Programming in CAEL“. In: *Proceedings of the IEEE Workshop on Visual Languages*. Aug. 1993, S. 389–391 (referenziert auf Seite 51).

[RG00] Oliver Radfelder und Martin Gogolla. „On Better Understanding UML Diagrams through Interactive Three-Dimensional Visualization and Animation“. In: *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI)*. Mai 2000, S. 292–295 (referenziert auf Seiten 18, 24, 137, 163, 164).

[RG93] Jun Rekimoto und Mark Green. „The Information Cube: Using Transparency in 3D Information Visualization“. In: *Proceedings of the Annual Workshop on Information Technologies and Systems*. Dez. 1993, S. 125–132 (referenziert auf Seiten 35, 36).

[RMC91] George G. Robertson, Jock D. Mackinlay und Stuart K. Card. „Cone Trees: Animated 3D Visualizations of Hierarchical Information“. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Reaching Through Technology*. Apr. 1991, S. 189–194 (referenziert auf Seiten 35, 36, 71).

[Röd07] Oliver Röder. *Grundlagen der Stereoskopie – Analyse der Aufnahme und Projektion von 3D-Bildern*. Saarbrücken: VDM Verlag Dr. Müller, 2007 (referenziert auf Seiten 122, 123).

[Röl07] Heiko Rölke. „3-D Petri nets – Making use of 3 Dimensions in Executable Petri Net Modelling“. In: *Petri Net Newsletter*, 72 (2007), S. 3–9 (referenziert auf Seiten 19, 137).

[RR13] Elena Rybka und Johann Rybka. *Navigation und Interaktion in Editoren für dreidimensionale Sprachen*. Masterarbeit, Universität Paderborn. Jan. 2013 (referenziert auf Seiten 104, 119).

[SAH14] Klaus Schoeffmann, David Ahlström und Marco A. Hudelist. „3-D Interfaces to Improve the Performance of Visual Known-Item Search“. In: *IEEE Transactions on Multimedia*, 16(7) (Nov. 2014), S. 1942–1951 (referenziert auf Seiten 51, 77).

[SBE00] Bastiaan Schönhage, Alex van Ballegooij und Anton Elliëns. „3D Gadgets for Business Process Visualization — a case study“. In: *Proceedings of the Fifth Symposium on Virtual Reality Modeling Language (VRML)*. Feb. 2000, S. 131–138 (referenziert auf Seiten 36, 163).

[Sch03a] Eike Schwindt. *Ein graphischer Editor für Generische Zeichnungen*. Studienarbeit, Universität Paderborn. Aug. 2003 (referenziert auf Seiten 73, 81).

[Sch03b] Eike Schwindt. „Ein graphischer Editor für Generische Zeichnungen“. In: *Fachwissenschaftlicher Informatikkongress - Informatiktage 2002*. Nov. 2003 (referenziert auf Seiten 73, 81).

[Sch06] Carsten Schmidt. „Generierung von Strukturediteren für anspruchsvolle visuelle Sprachen“. Dissertation. Universität Paderborn, Jan. 2006 (referenziert auf Seiten 3, 10–12, 38, 39, 42, 43, 61, 62, 64, 65, 73, 81, 85, 93, 100, 129, 134, 158).

[Sch98] Stefan Schiffer. *Visuelle Programmierung – Grundlagen und Einsatzmöglichkeiten*. Bonn: Addison-Wesley-Longman, 1998 (referenziert auf Seiten 9, 10).

[SCK07] Carsten Schmidt, Bastian Cramer und Uwe Kastens. „Usability Evaluation of a System for Implementation of Visual Languages“. In: *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Sep. 2007, S. 231–238 (referenziert auf Seite 158).

[SE14] Leïla Schemali und Elmar Eisemann. „Design and Evaluation of Mouse Cursors in a Stereoscopic Desktop Environment“. In: *IEEE Symposium on 3D User Interfaces (3DUI)*. März 2014, S. 67–70 (referenziert auf Seite 127).

[Shn83] Ben Shneiderman. „Direct Manipulation: A Step Beyond Programming Languages“. In: *IEEE Computer*, 16(8) (Aug. 1983), S. 57–69 (referenziert auf Seite 43).

[Sho92] Ken Shoemake. „ARCBALL: A User Interface for Specifying Three-dimensional Orientation Using a Mouse“. In: *Proceedings of the Conference on Graphics Interface '92*. Mai 1992, S. 151–156 (referenziert auf Seite 127).

[SK03] Carsten Schmidt und Uwe Kastens. „Implementation of visual languages using pattern-based specifications“. In: *Software – Practice and Experience*, 33(15) (2003), S. 1471–1505 (referenziert auf Seiten 38, 41, 61, 80).

[SKC06] Carsten Schmidt, Uwe Kastens und Bastian Cramer. „Using DEViL for Implementation of Domain-Specific Visual Languages“. In: *Proceedings of the Workshop on Domain-Specific Program Development*. Juli 2006 (referenziert auf Seite 38).

[SM10] Torsten Strobl und Mark Minas. „Specifying and Generating Editing Environments for Interactive Animated Visual Models“. In: *Proceedings of the Ninth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT)*. Hrsg. von Jochen Küster und Emilio Tuosto. Bd. 29. Electronic Communications of the EASST. März 2010 (referenziert auf Seiten 45, 164).

[SP92] Randy Stiles und Michael Pontecorvo. „Lingua Graphica: A Visual Language for Virtual Environments“. In: *Proceedings of the IEEE Workshop on Visual Languages*. Sep. 1992, S. 225–227 (referenziert auf Seite 17).

[SS00] Carsten Schmidt und Christian Schindler. *Muster-basierte Generierung von Struktur-Editoren für visuelle Sprachen*. Diplomarbeit, Universität Paderborn. Jan. 2000 (referenziert auf Seiten 61, 62, 64, 80).

[Sto14] Florian Stolte. *Entwurf einer 3D-Sprache für UML-Klassendiagramme zur Verbesserung der Darstellung von parametrisierten Klassen*. Bachelorarbeit, Universität Paderborn. Nov. 2014 (referenziert auf Seiten 77, 138).

[Stü14] Hendrik Stürmann. *Spezifikation eines Editors für dreidimensionale UML-Sequenzdiagramme*. Bachelorarbeit, Universität Paderborn. Mai 2014 (referenziert auf Seiten 89, 137).

[Sut64] Ivan E. Sutherland. „Sketchpad: A man-machine graphical communication system“. In: *Proceedings of the SHARE design automation workshop (DAC)*. 1964, S. 329–346 (referenziert auf Seite 101).

[SW93] John T. Stasko und Joseph F. Wehrli. „Three-Dimensional Computation Visualization“. In: *Proceedings of the IEEE Workshop on Visual Languages*. Aug. 1993, S. 100–107 (referenziert auf Seiten 26, 35).

[TC09] Alfredo R. Teyseyre und Marcelo R. Campo. „An Overview of 3D Software Visualization“. In: *IEEE Transactions on Visualization and Computer Graphics*, 15(1) (2009), S. 87–105 (referenziert auf Seiten 1, 35, 37).

[TDL04] Monica Tavanti, Nguyen-Thong Dang und Hong-Ha Le. „Usability Inspection of a 3D Interaction Metaphor“. In: *Proceedings of the Conference Internationale Associant Chercheurs Vietnamiens et Francophones en Informatique*. Feb. 2004, S. 41–46 (referenziert auf Seite 127).

[TRC01] Desney S. Tan, George G. Robertson und Mary Czerwinski. „Exploring 3D Navigation: Combining Speed-coupled Flying with Orbiting“. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. März 2001, S. 418–425 (referenziert auf Seite 127).

[Tve08] Merete Skjelten Tveit. „Towards Diagrammatic Patterns“. In: *Diagrammatic Representation and Inference*. Hrsg. von Gem Stapleton, John Howse und John Lee. Bd. 5223. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, 2008, S. 427–429 (referenziert auf Seite 80).

[TWHG98] Michael J. Tarr, Pepper Williams, William G. Hayward und Isabel Gauthier. „Three-Dimensional Object Recognition is Viewpoint-Dependent“. In: *Nature Neuroscience*, 1(4) (1998), S. 275–277 (referenziert auf Seite 51).

[Voß09] Volker Voß. *Dreidimensionale Darstellung zweidimensionaler visueller Sprachen*. Diplomarbeit, Universität der Bunderwehr München. Juni 2009 (referenziert auf Seite 46).

[VSDF96] Sabine Volbracht, K. Shahrbabaki, Gitta Domik und Gregor Fels. „Perspective viewing, Anaglyph stereo or Shutter glass stereo?“ In: *Proceedings of the IEEE Symposium on Visual Languages*. Sep. 1996, S. 192–193 (referenziert auf Seite 159).

[War08] Colin Ware. *Visual Thinking for Design*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008 (referenziert auf Seite 37).

[War12] Colin Ware. *Information Visualization: Perception for Design*. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012 (referenziert auf Seite 37).

[WCJ98] Ulrika Wiss, David Carr und Håkan Jonsson. „Evaluating Three-Dimensional Information Visualization Designs: a Case Study of Three Designs“. In: *Proceedings of the IEEE Conference on Information Visualization*. Juli 1998, S. 137–144 (referenziert auf Seite 36).

[WCK11a] Jan Wolter, Bastian Cramer und Uwe Kastens. „Animation of Tile-Based Games Automatically Derived from Simulation Specificati-

ons“. In: *Computer Science and Information Systems*, 8(2) (2011). extended Journal Paper, S. 501–516 (referenziert auf Seite 43).

[WCK11b] Jan Wolter, Bastian Cramer und Uwe Kastens. „Towards Three-Dimensional Visual Languages“. In: *Proceedings of Compilers, Programming Languages, Related Technologies and Applications (CoRTA)*. Sep. 2011, S. 282–287 (referenziert auf Seite 6).

[WF94] Colin Ware und Glenn Franck. „Viewing a Graph in a Virtual Reality Display is Three Times as Good as a 2D Diagram“. In: *Proceedings of the IEEE Symposium on Visual Languages*. Okt. 1994, S. 182–183 (referenziert auf Seite 51).

[WK14] Jan Wolter und Uwe Kastens. „Encapsulating Interaction Techniques of 3D Language Editors in Visual Patterns“. In: *Proceedings of the 7th International Symposium on Visual Information Communication and Interaction (VINCI)*. Aug. 2014, S. 68–77 (referenziert auf Seiten 7, 104, 141, 143).

[WK15] Jan Wolter und Uwe Kastens. „Generating 3D Visual Language Editors: Encapsulating Interaction Techniques in Visual Patterns“. In: *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)* (2015). to appear (referenziert auf Seiten 7, 104, 141).

[WL07] Richard Wettel und Michele Lanza. „Visualizing Software Systems as Cities“. In: *Proceedings of the IEEE International Workshop on Visualizing Software for Understanding and Analysis*. Juni 2007, S. 92–99 (referenziert auf Seite 36).

[Wol11] Jan Wolter. *Ein Konzept zur Implementierung visueller dreidimensionaler Sprachen*. Masterarbeit, Universität Paderborn. Mai 2011 (referenziert auf Seiten 25, 26).

[Wol12] Jan Wolter. „DEViL3D – A Generator Framework for Three-Dimensional Visual Languages“. In: *Proceedings of the International Workshop on Visual Languages and Computing (VLC) in conjunction with the 18th International Conference on Distributed Multimedia Systems (DMS)*. Aug. 2012, S. 171–176 (referenziert auf Seite 6).

[Wol13a] Jan Wolter. „Specifying Generic Depictions of Language Constructs for 3D Visual Languages“. In: *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Sep. 2013, S. 139–142 (referenziert auf Seiten 6, 73, 90, 137).

[Wol13b] Jan Wolter. „Visual Representation of 3D Language Constructs Specified by Generic Depictions“. In: *The Computing Research Repository (CoRR)*, abs/1311.5126 (2013). `http://arxiv.org/abs/1311.5126` (referenziert auf Seite 73).

[Wol14] Jan Wolter. „Layout Requirements of a 3D Molecular Editor Specified with DEViL3D“. In: *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Juli 2014, S. 223–224 (referenziert auf Seiten 6, 95).

[Yam96] Kakuya Yamamoto. „3D-Visulan: A 3D Programming Language for 3D Applications“. In: *The third Pacific Workshop on Distributed Multimedia Systems (DMS)*. Juni 1996, S. 199–206 (referenziert auf Seite 16).

[YF96] Masoud Yazdani und Lindsey Ford. „Reducing the cognitive requirements of visual programming“. In: *Proceedings of the IEEE Symposium on Visual Languages*. Sep. 1996, S. 255–262 (referenziert auf Seite 22).

[YN04] Andy Yeh und Rod Nason. „VRMath: A 3D Microworld for Learning 3D Geometry“. In: *Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications*. Juni 2004, S. 2183–2191 (referenziert auf Seite 22).

[Zha12] Kang Zhang. „Using visual languages in management“. In: *Journal of Visual Languages and Computing*, 23(6) (2012), S. 340–343 (referenziert auf Seite 163).

[ZZC01] Kang Zhang, Da-Qian Zhang und Jiannong Cao. „Design, Construction, and Application of a Generic Visual Language Generation Environment“. In: *IEEE Transactions on Software Engineering*, 27(4) (Apr. 2001), S. 289–307 (referenziert auf Seite 38).