Faszination Programmierung

Freude mit Graphik und vielfältigen Konstrukten

Hinrich E. G. Bonin[1]

24. August 2006

[1]Prof. Dr. rer. publ. Dipl.-Ing. Dipl.-Wirtsch.-Ing. Hinrich E. G. Bonin, Universität Lüneburg, Fakultät III, Volgershall 1, D-21339 Lüneburg, Germany.

Inhaltsverzeichnis

1 Vorspann **11**
- 1.1 Zusammenfassung . . . 12
- 1.2 Vorwort . . . 13
- 1.3 Notation . . . 17

2 Ist programmieren eine Kunst? **19**
- 2.1 Was ist programmieren? . . . 20
  - 2.1.1 Definition: Programm . . . 20
  - 2.1.2 Prozess der Realisierung von Qualität . . . 22
- 2.2 Kreativität & Disziplin . . . 22

3 PostScript^{TM}-Paradigma — Kommandos überall **25**
- 3.1 Notation von Kommandos . . . 27
  - 3.1.1 Postfix-Notation . . . 27
  - 3.1.2 Präfix-Notation . . . 28
- 3.2 PostScript-Kostproben . . . 29
  - 3.2.1 Einfache Linie . . . 29
  - 3.2.2 Einfaches Dreieck . . . 32
  - 3.2.3 Grau ausgefülltes Dreieck . . . 32
  - 3.2.4 Hello World! . . . 33
  - 3.2.5 Zusammengesetzte Kostprobe . . . 35
- 3.3 Stack . . . 36
  - 3.3.1 Last-In-First-Out (LIFO) —*Stack*: *Push* und *Pop* 36
  - 3.3.2 *Graphic State Stack*-Beispiel . . . 38
- 3.4 Datentypen . . . 39
- 3.5 Clipping Region . . . 41

4 Java^{TM}-Paradigma — Objekte überall 45
4.1 J2SE (Java 2 Standard Edition) 46
4.2 Virtual Universe — Java3D-Paket 53
4.3 Java3D-Kostproben 58
4.3.1 `HelloUniverse` 58
4.3.2 `HelloWorld` 70
4.3.3 `SimpleFigure3D` — 3 Fälle 75
4.3.4 Durchsichtiger Zylinder 97
4.3.5 Drehender Text 103
4.3.6 Konstruierte Geometrie 114
4.3.7 `DataSharing` 121
4.3.8 `ExampleWavefrontLoad` 128

5 Objekt-Verhalten — `Behavior` 139
5.1 Anregung und Aktion 140
5.2 *Custom Behavior Class* 141
5.3 `KeyPressRotation` 142
5.4 `Navigation` 150
5.5 `ObjectWithMouse` 158

6 Beispielprojekt Jagdhund 165
6.1 1. Ansatz mit `GeometryInfo` — `Wachtel1` 166
6.2 Klasse `Wachtel1` 170

7 Ausblick 191

A Übungen 193
A.1 UML-Klassensymbol programmieren 194
A.2 PostScript-Kontur erläutern 194
A.3 PostScript-Quellcode interpretieren 195
A.3.1 Graphik zeichnen 196
A.3.2 Graphik klassifizieren 196
A.4 Java3D-Programm erläutern 196
A.4.1 Konstruktor erläutern 198
A.4.2 Parameteränderung erläutern 198
A.5 Objekt & Referenz 200
A.5.1 Kopieproblem erläutern 201
A.5.2 Einhaltung von Notationsregeln prüfen 201

A.6 Interface & Inheritance 201
A.6.1 Interface implementieren 204
A.6.2 Zugriff auf Klassenvariable 204
A.7 Lokale Klasse 204
A.7.1 Lokale Klasse erläutern 205
A.7.2 Ergebnis der Java-Applikation `Think` 205

B Lösungen zu den Übungen **207**

C Quellen **211**
C.1 Literaturverzeichnis 212
C.2 Web-Quellen 215

D Hinweise zu Faszination Programmierung **217**
D.1 Werkzeuge zum Manuskript 217
D.2 Liste der Fonts 217

E Glossar **221**

F Abkürzungen und Akronyme **223**

G Index **225**

Abbildungsverzeichnis

3.1 PostScript: Einfache Linie 31
3.2 PostScript: Einfaches Dreieck 32
3.3 PostScript: Graues Dreieck 33
3.4 PostScript: Gedrehter Schriftzug 34
3.5 PostScript: `Hello World!` mit grauem Dreieck 35
3.6 PostScript: `Hello World!` mit umrandetem Dreieck . 37
3.7 PostScript: *Octagon*-Figur 40
3.8 Path als Clipping Region 43

4.1 Beispiel: Java API 47
4.2 `Text`-Applet mit `appletviewer` 53
4.3 Eingabefenster von `Text`-Applet 53
4.4 `Text`-Applet mit Browser *Firefox 1.0.7* 54
4.5 *Scene Graph* — erstes Beispiel 55
4.6 *Scene Graph* — Symbole 56
4.7 Beispiel: `HelloUniverse` 59
4.8 Beispiel: `HelloUniverse` — vereinfacht 63
4.9 Beispiel: `HelloWorld` 71
4.10 Beispiel: `SimpleFigure3Da` 76
4.11 Hintergrundbild: `brickwork.jpg` 81
4.12 Beispiel: `SimpleFigure3Db` — Textur `brickwork.jpg` 82
4.13 Hintergrundbild: `code.jpg` 87
4.14 Beispiel: `SimpleFigure3Db` — Textur `code.jpg` . 88
4.15 Beispiel: `SimpleFigure3Dc` 96
4.16 Beispiel: `Transparency` 98
4.17 Beispiel: Drehender Text 104

4.18 Hintergrundbild: `AutomatonR110.eps` 105
4.19 TriangleFanArray — *Vertices* $v_{0..4}$ 114
4.20 Beispiel: Konstruierte Geometrie — YoYo 116
4.21 Beispiel: `DataSharing` 124
4.22 Beispiel: `ExampleWavefrontLoad` — Objekt in Cinema4D 129
4.23 Beispiel: `ExampleWavefrontLoad` — Objekt in Java3D 131
4.24 Beispiel: `ExampleWavefrontLoad` — Komplexes Objekt 138

5.1 Beispiel: `KeyPressRotation` — Hintergrundtextur . 143
5.2 Beispiel: `KeyPressRotation` 143
5.3 Beispiel: `Navigation` 150
5.4 Beispiel: `ObjectWithMouse` 159

6.1 Beispiel: `Wachtel1` — Dreiecke 167
6.2 Beispiel: `Wachtel1` — Ausgangspolygone 169

A.1 UML Klassensymbol 194
A.2 Java3D-Graphik 199

B.1 Zwei sich überdeckende Halbkreise 208
B.2 UML Klassensymbol 208

D.1 Liste der Fonts von `DVIPS` 218
D.2 Liste der Fonts von `CorelDraw12` 219

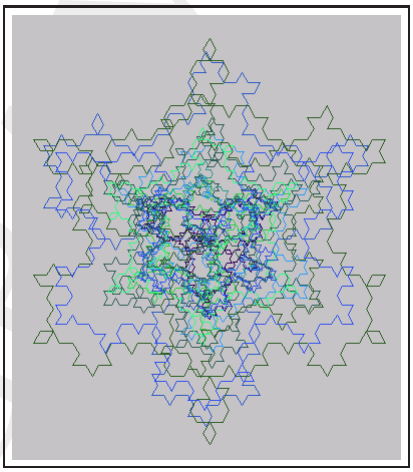
Tabellenverzeichnis

1.1 Internet Smileys 18

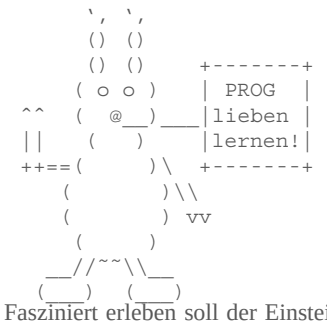
5.1 Objekt-Verhalten: Anregung und Aktion 141
5.2 Klasse `KeyNavigatorBehavior` — Wirkungen der Tasten 151

Kapitel 1

Vorspann



1.1 Zusammenfassung



Wie in vielen Hochschulen, so erfolgt auch in der Universität Lüneburg derzeit die Einführung in die *Grundlagen der Programmierung* (PROG) für Informatiker und Wirtschaftsinformatiker primär auf Basis der Programmiersprache Java^{TM} von Sun Microsystems, Inc. USA. Ziel ist es, programmieren als einen systematischen Konstruktionsvorgang zu vermitteln.

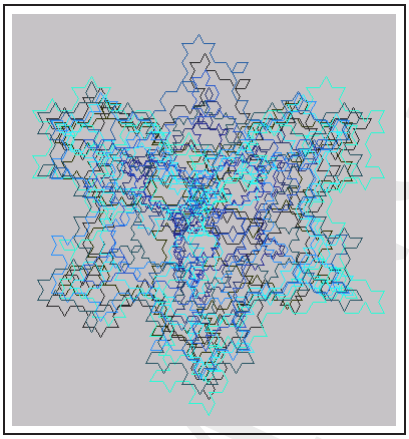
Fasziniert erleben soll der Einsteiger die konkrete Entwicklung von Algorithmen und Datenstrukturen als ein diszipliniertes Vorgehen :-). Zum Einordnen und Verstehen des *Objekt-Orientierte Paradigmas* (Denkmodells) wird die imperativ-orientierte Programmierung in PostScript^{TM} skizziert.

Das Fach *Programmierung* verlangt vom Einsteiger auf seinem Weg zum Versteher viel Geduld und Ausdauer :-). Der harte Weg läßt sich dann erfolgreich durchlaufen, wenn programmieren viel Freude macht und Faszinaion vermittelt. Deshalb sind die Beispiele in Faszination Programmierung aus dem Bereich Graphik gewählt worden.

Klar ist, das programmieren erlernen Sie zunächst am besten durch „abkupfern“ und spielen mit gelungenen Programmen. Klar ist aber auch, gute eigene Programme setzen ein tiefes Verständnis der tragfähigen Modelle und Konzepte der jeweiligen Programmiersprache voraus. Faszination Programmierung vermittelt daher das imperative und das Objekt-Orientierte Paradigma (Denkmodell).

Faszination Programmierung[1] versucht ein solches Verständnis Schritt für Schritt aufzubauen. Bei den Beispielen, Übungen und Musterlösungen geht es primär um ein Begreifen und Umgehen mit der Komplexität, die einem Programm innewohnt. Plakativ formuliert möchte Faszination Programmierung Ihnen helfen Java^{TM} als ein Akronym für *Just a valid application* zu verstehen ;-).

[1] Hinweis: Dieses Dokument ist ein Entwurf und wird laufend fortgeschrieben. Die aktuelle Version befindet sich unter:
http://as.uni-lueneburg.de/publikation/progall.pdf.
Anmerkungen und Kommentare schicken Sie bitte an:
bonin@uni-lueneburg.de
(Remark: This is a **draft document** and continues to be revised. The latest version can be found at
http://as.uni-lueneburg.de/publikation/progall.pdf.
Please send comments to bonin@uni-lueneburg.de)

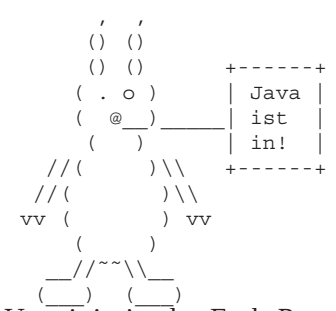


1.2 Vorwort

Kunst ist nicht nur die statische Errungenschaft
der Meister der Vergangenheit.
Kunst ist die kreative Dynamik-Qualität
der Künstler von heute.[2]
Robert M. Pirsig ↪ [Glover03] p. vii

Sie wollen und/oder müssen sich mit der Programmierung befassen? Das ist gut so! Schreiben von Programmen kann viel Freude machen. Welche Programmiersprache Sie dazu am besten nutzen sollten hängt weitgehend vom jeweiligen Zeitgeist ab :-). Um 1980 war es *List Processing* (LISP) (↪ [Bonin91b]). Derzeit ist es Java^{TM}. Ob nun Java^{TM} noch im Jahre 2010 relevant ist oder sich in einer Rolle wie heute LISP befindet, ist unerheblich. Es geht um die Frage wie kann effektiv ein Begreifen der Java^{TM} innewohnenden Modelle und Konzepte ermöglicht werden.

[2] *„Art is not just the static achievements of the masters of the past. Art is the creative Dynamic Quality of the artist of the present.“*



Ziel ist es, ein Spektrum von Möglichkeiten der Notation von Quellcode aufzuzeigen. Deshalb wird neben Java auch die Programmierung in PostScript beispielhaft skizziert.

Unstrittig ist das Fach *Programmierung* ein Kern der großen Disziplin Informatik. Es verlangt vom Einsteiger auf seinem Weg zum Versteher viel Geduld und Ausdauer :-). Erwerben muß er fundiertes Wissen über Schwerpunktthemen wie zum Beispiel:

- Syntax und Semantik elementarer Konstrukte (Sequenz, Alternative, Iteration, Rekursion),
- Repräsentation von Daten (Klassenkonstrukt, Variablen, Datentyp),
- Kommunikation von Objekten (Methoden, Signaturen),
- Erzeugung von Objekten (Vererbung, Interface),
- Handhabung von Klassenmengen (Pakete, Archive, Zugriffsrechte),
- Gebräuchliche Konstruktionen (Pattern) und
- allgemeine Konstruktionsempfehlungen (Bezeichner, Dokumentationsregeln)

Dieser harte Weg läßt sich nur dann erfolgreich durchlaufen, wenn programmieren letztlich Freude macht und Faszinaion vermittelt. Deshalb sind die Beispiele in Faszination Programmierung primär aus dem Bereich Graphik gewählt worden.

Faszination Programmierung wendet sich an alle, die mittels schöner Bilder die Lust nicht verlieren wollen, damit sie die harte Arbeit des Begreifens von Programmen meistern. Dabei spielt Ihr Alter keine Rolle, denn nach den neueren Erkenntnissen der Hirnforschung verfügt das Hirn über eine hohe Plastizität und die Fähigkeit der Neurogenese, bei der neue Nervenzellen in bestehende Verschaltungen eingefügt werden. Dank dieser Hirneigenschaften *kann Hans also durchaus noch lernen, was Hänschen nicht gelernt hat* :-) — auch wenn es mit den Jahren deutlich schwerer fällt.

Der Anfänger muß viele Konstrukte erlernen und bewährte Konstruktionen nachbauen. Ebensowenig wie zum Beispiel Autofahren allein aus Büchern erlernbar ist, wird niemand zum „Progamming-Wizard“

(deutsch: Hexenmeister) durch das Nachlesen von erläuterten Beispielen. Das intensive Durchdenken der Beispiele, im Dialog mit einer passenden Entwicklungsumgebung, vermittelt jedoch im Sinne der gezogenen Analogie, unstrittig die Kenntnis der Straßenverkehrsordnung und ermöglicht ein erfolgreiches Teilnehmen am Verkehrsgeschehen — auch im Großstadtverkehr. Für diesen Lernprozeß wünsche ich Ihnen, der „Arbeiterin“ oder dem „Arbeiter“, viel Freude.

Faszination Programmierung ist konzipiert als ein Buch zum Selbststudium und für Lehrveranstaltungen. Mit den umfassenden Quellenangaben und vielen Vor- und Rückwärtsverweisen, dient es auch als Nachschlagewerk und Informationslieferant für Spezialfragen. Jedoch ist es kein umfassendes Java-Kompendium.

Faszination Programmierung wurde im Jahre 2005 begonnen und zwar nachdem der JAVA^{TM}–COACH (↪ [Bonin04b]) — ein mehr als 500 Seiten starkes Buch — weitgehend fertiggestellt und ein erstes Manuskript über Kunst und Programmierung enstanden war. Während der Fortschreibung eines Buches, lernt man erfreulicherweise stets dazu. Das hat jedoch auch den Nachteil, daß man laufend neue Unzulänglichkeiten erkennt. Schließlich ist es, trotz solcher Schwächen, der Öffentlichkeit zu übergeben. Ich bitte Sie daher im voraus um Verständnis für Unzulänglichkeiten. Willkommen sind Ihre konstruktiven Vorschläge, um die Unzulänglichkeiten Schritt für Schritt weiter zu verringern. Ihre Vorschläge werden mit Ihrer Zustimmung über den Web-Server:

```
http://as.uni-lueneburg.de
```

verfügbar gemacht. Dort finden Sie auch aktuelle Ergänzungen.

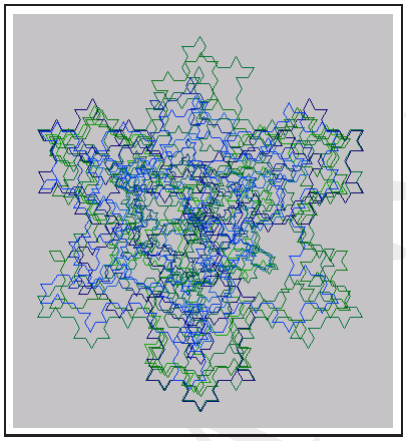
Danksagung

Für das Interesse und die Durchsicht einer der ersten Fassungen danke ich meinem Kollegen Herrn Dipl.-Ing. Christian Wagner. Für ein gründliches Korrekturlesen danke ich Herrn Dipl.-Kfm. Nobert Tschritter.

Ohne die kritischen Diskussionen mit Studierenden im Rahmen der Lehrveranstaltungen wäre Faszination Programmierung nicht in dieser Form entstanden. Ihnen möchte ich an dieser Stelle ganz besonders danken.

Lüneburg, 24. August 2006

```
<Erfasser>
     <Verfasser>
          Hinrich E. G. Bonin
     </Verfasser>
</Erfasser>
```



1.3 Notation

In Faszination Programmierung wird erst gar nicht der Versuch unternommen, die weltweit übliche Informatik-Fachsprache Englisch zu übersetzen. Es ist daher teilweise „mischsprachig“: Deutsch und Englisch. Aus Lesbarkeitsgründen sind nur die männlichen Formulierungen genannt; die Leserinnen seien implizit berücksichtigt. So steht das Wort „Programmierer“ hier für Programmiererin und Programmierer.

Englische Fachbegriffe

Für die Notation der Modelle und Algorithmen werden auch im Text PostScript^{TM} und Java^{TM} genutzt. Beispielsweise wird im Fall von Java zur Kennzeichnung einer Methode — also eines „aktivierbaren“ Objektteils — eine leere Liste an den Bezeichner angehängt, zum Beispiel `createSceneGraph()`.

Ein Programm (Quellcode) ist in der Schriftart `Typewriter` dargestellt. Ausgewiesene Zeilennummern in einer solchen Programmdarstellung sind kein Bestandteil des Quellcodes. Sie dienen zur Vereinfachung der Erläuterung.

`Typewriter`

Primär aus Sicherheitsgründen und aus didaktischen Gründen werden die zu importiernden Java-Klassen explizit genannt und nicht nur ihr Paket. So umfasst der Kopf einer Klasse beispielsweise:

```
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Box;
import com.sun.j3d.utils.geometry.Sphere;
```

`:-)`	Your basic smiley. This smiley is used to inflect a sarcastic or joking statement since we can't hear voice inflection over e-mail.
`;-)`	Winky smiley. User just made a flirtatious and/or sarcastic remark. More of a ¨don't hit me for what I just said¨ smiley.
`:-(`	Frowning smiley. User did not like that last statement or is upset or depressed about something.
`:-I`	Indifferent smiley. Better than a `:-(` but not quite as good as a `:-)`.
`:->`	User just made a really biting sarcastic remark. Worse than a `;-)`.
`>:->`	User just made a really devilish remark.
`>;->`	Winky and devil combined. A very lewd remark was just made.

Legende:
Quelle ↪ `http://members.aol.com/bearpage/smileys.htm`
(online 21-Nov-2003)

Tabelle 1.1: Internet Smileys

statt einfach:

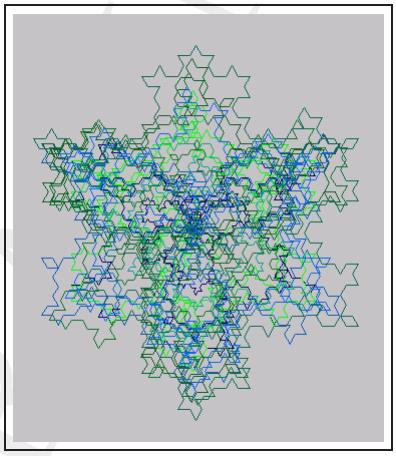
```
import com.sun.j3d.utils.geometry.*;
```

Hinweis:
Die Programme sind über einen längeren Zeitraum entstanden. Zu Beginn gab es noch die Institution *Fachhochschule Nordost Niedersachsen* (FHNON) `:-)`. Daher sind einige Java-Programme im *Package* `de.fhnon.as.`xxx ausgewiesen.
Das *Package* für den Zeitraum nach der Fusion zur Universität Lüneburg wurde `de.unilueneburg.as.`xxx benannt. Auf den üblichen Bindestrich zwichen `uni` und `lueneburg` mußte verzichtet werden, da andernfalls Fehler auftreten.

PS: Ab und zu werden zur Aufmunterung und zum Schmunzeln im Text *Internet Smileys* benutzt. Ihre Bedeutung erläutert Tabelle 1.1 S. 18.

Kapitel 2

Ist programmieren eine Kunst?



Kunst ist Mathematik![1]
Alexej von Jawlensky, 1864–1941, russ.-dt. Maler
zitiert in ↪ [Mäckler00] S. 27

[1] In Walter Dexel; Der Bauhausstil — ein Mythos, Starnberg (Josef Keller Verlag), 1976, S. 39

techne

„Der handwerksbedingte Ursprung des Kunstbegriffs manifestiert sich in dem griechischen Wort *techne*, in dem Aristoteles die Fähigkeit ausgedrückt sieht, ein Produkt herzustellen, mit dem Wissen um das innewohnende Prinzip. Gelingt die Arbeit, wird von *können* als *Kunst* gesprochen; auch die Heil- und Kriegskunst zählen in der Antike zu den *artes*.“ :-) (↪ [Mäckler00] S. 9)

Trainingsplan

Das Kapitel „Ist programmieren eine Kunst?“ gibt einen Überblick über:

- die Frage: Was ist programmieren?, und
  ↪ Seite 20 . . .
- das Zusammenwirken aus Kreativität und strikt diszipliniertem Vorgehen.
  ↪ Seite 22 . . .

2.1 Was ist programmieren?

Wenn programmieren den gesamten Vorgang zur Schaffung eines Progamms umfasst, dann stellt sich die Frage: Was ist ein Programm? — oder anders formuliert, was ist eine allgemein anerkannte Definition für den Begriff *Programm*.

2.1.1 Definition: Programm

Leider ist der Begriff *Programm* sehr unscharf (*fuzzy*) und wird zusätzlich in vielfältigen Zusammenhängen verwendet auch unabhängig von Rechnern. Zum Beispiel spricht man vom Programm der politischen Partei XY oder vom Programm der Bundesregierung :-(. Üblicherweise und ganz allgemein thematisiert der Begriff *Programm* einen geplanten

Ablauf, der von Menschen oder Rechnern abgearbeitet werden soll.[2] Wir unterstellen, dass ein solcher Ablauf irgendwie ausgeschreiben sein muss und definieren dann:

> **Ein Programm[3] ist ein im Voraus festgesetzter Ablauf, der so notiert ist, dass er von Menschen und/oder Rechnern vollzogen werden kann.**

Algorithmus

So gesehen entspricht der Begriff *Programm* dem Begriff *Algorithmus*. Die Formulierung eines Algorithmus unterliegt jedoch keinen so engen Sprachvorschriften, wie die eines Programms. Dieses muss exakt die *Semiotik*, das heißt,

- die *Syntax*[4] ($\approx$ Formvorschriften für die Sprachkonstrukte),
- die *Semantik*[5] ($\approx$ Bedeutung der Sprachkonstrukte) und
- die *Pragmatik*[6] ($\approx$ korrekte Anwendungskontext der einzelnen Sprachkonstrukte),

der gewählten Programmiersprache einhalten. Ein Algorithmus kann auf beliebige Art formuliert sein, wenn er nur im Voraus einen bestimmten Ablauf festlegt. Anders formuliert: Ein und derselbe Algorithmus kann in verschiedenen Programmiersprachen formuliert werden.

Im Zusammenhang mit Rechnern versteht man unter einem Programm eine Einheit zur Erfüllung von vordefinierten Aufgaben. Ein Programm kann in sehr unterschiedlichen Größenordnungen vorkommen. Aus der Größenperspektive wird ein Programm als Baustein, Modul, Procedur, Funktion etc. bezeichnet. Wirken mehrere Einheiten (Programme) zusammen spricht man in der Regel von Software.

---

[2] Zum Beispiel $\hookrightarrow$ `http://de.wikipedia.org/wiki/Programm` (online 06-Oct-2005)

[3] französisch: *programme* $\equiv$ schriftliche Bekanntmachung, Tagesordnung; griechisch: *prógramma* $\equiv$ Vorgeschriebenes, Vorschrift

[4] Griechisch *Syntaxis* $\equiv$ Lehre vom Satzbau, Satzlehre

[5] Semantik $\equiv$ Bedeutungslehre

[6] Pragmatik von griechisch *pragma* $\equiv$ Tat; Sachkunde, besonders die Geschäftsordnung im Staatsdient

2.1.2 Prozess der Realisierung von Qualität

Bei der Schaffung eines Programms (P) – oder allgemeiner von Software – geht es um die Realisierung der fallspezifisch notwendigen Qualität (Q). Sie ergibt sich primär aus den folgenden Eigenschaften eines Programms:

$P(Q)$

1. *Leistungsfähigkeit*,
   das heißt, das Programm erfüllt die gewünschten Anforderungen.
2. *Zuverlässigkeit*,
   das heißt, das Programm arbeitet auch bei ungewöhnlichen Bedienungsmaßnahmen und bei Ausfall gewisser Komponenten weiter und liefert aussagekräftige Fehlermeldungen (Robustheit),
3. *Durchschaubarkeit & Wartbarkeit*,
   das heißt, das Programm kann auch von anderen Programmierern als dem Autor verstanden, verbessert und auf geänderte Verhältnisse eingestellt werden,
4. *Portabilität & Anpassbarkeit*,
   das heißt, das Programm kann ohne großen Aufwand an weitere Anforderungen angepasst werden,
5. *Ergonomie & Benutzerfreundlichkeit*,
   das heißt, das Programm ist leicht zu handhaben,
6. *Effizienz*,
   das heißt, das Programm benötigt möglichst wenig Ressourcen.

Aus der Perspektive dieser Eigenschaften betrachtet, läßt sich ein Programm, im Kontext von Rechnern, als die Umsetzung eines Algorithmus mit hinreichender Qualität betrachten. Wir halten fest, nicht jede Folge von Befehlen, die „läuft“ und irgendetwas macht, kann den Ehrennamen Programm bekommen, denn es fehlen ihr wesentliche Eigenschaften (Qualitätsansprüche).

2.2 Kreativität & Disziplin

Man braucht Ideen, also Kreativität, wenn es gilt Konstrukte einer gewählten Programmiersprache so zu notieren, das ein Programm entsteht,

das die erforderlichen Qualitätsansprüche erfüllt. Kurz und holzschnittartig formuliert: *Der Programmierer gehört zur Gilde der Kreativen.*

Bei komplexen Zusammenhängen reicht aber Kreativität nicht aus. Die schönen Ideen kommen selten zielorientiert genau zur rechten Zeit. Vielmehr gilt es, durch ein systematisches Vorgehen, die Kreativität zu kanalisieren, also durch die Einhaltung einer bewährten Schrittfolge, das kreative Denken auf die jeweils angebrachten Themen (Punkte) zu konzentrieren. Kurz und holzschnittartig formuliert: *Der Programmierer gehört zur Gilde der Buchhalter*, weil dieser gewohnt ist, diszipliniert nach Regeln, komplexe Fälle zu bearbeiten.

Wenn man Programmieren anhand von Graphiken (Bildern) erlernen will, dann sollte das Ergebnis einer Programmausführung (hoffentlich) Kunst sein. Was Kunst ist, dürfte eine kaum klärbare philosophische Frage sein. Zwei Zitate mögen dazu jedoch etwas einstimmen, also Diskussionsstoff liefern.

> „Wer meint, dass man Bilder außerhalb der ästhetischen Erfahrung erzeugen kann, der bewegt sich auf demselben Grat wie die Alchimisten oder wie diejenigen, die nach dem Perpetuum Mobile suchen. Aber was heißt nun ästhetische Erfahrung? . . . Ästhetische Erfahrung ist die Selbstkonstituierung des Einzelnen in der Praxis des Lebens (egal in welcher Form dieser Praxis — als Jäger, Sammler, Bauer, Arbeiter, Wissenschaftler, Lehrer, Pfarrer, Politiker usw.) nach dem universellen Gesetz der Selbstoptimierung.“
> ↪ [Nadin03] S. 119.

> „der künstler als programmierer *schafft werke als klassen von werken*. er denkt grundsätzlich, wenn er schafft. er denkt an alle bilder, die ein inneres band verbindet. er denkt an bilder als klassen, die es berechenbar zu realisieren gilt.“ :-)
> ↪ [Nake03] S. 139. (Hinweis: Im Orginal klein geschrieben.)

Ein Kunstwerk behält seine einzigartige Schönheit, auch wenn man den Algorithmus genau versteht, mit dem es erzeugt wurde. Beispielsweise verliert eine Fuge von Bach keinesfalls ihre Faszination, nur weil man exakt weiß, wie sie konstruiert ist. Klar, die Musikwissenschaft kann zu Bachs Fuge einiges fundiert sagen, aber zur Erklärung ihrer individuellen Faszination bleibt sie überfragt ;-).

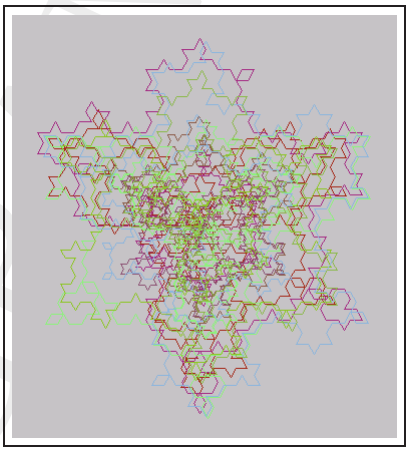
Ob nun ein Programm (P) selbst, also der notierte Code in der jeweiligen Programmiersprache (*Quellcode*), als ein Kunstwerk betrachtbar ist, mag mancher bestreiten. Wer jedoch wirklich vom Programmieren fasziniert ist, der erkennt einen gelungenen Quellcode durchaus als Kunstwerk an.

P **= Kunst?**

Kapitel 3

PostScript^{TM}-Paradigma — Kommandos überall

PostScript has become
an established part of worldwide
graphics, design, publishing, and printing.
(↪ [McGilton/Campione92] p. xiii)



Die Sprache PostScript^{TM}[1], seit $\approx$ 1985 entwickelt von Adobe Systems Incorporated, integriert drei Punkte:

[1] PostScript is a registered trademark of Adobe Systems Incorporated.
Post $\equiv$ Vorsilbe mit der Bedeutung „nach, hinter“
Skript $\equiv$ „Geschriebenes“, schriftliche Ausarbeitung, Nachschrift einer Hochschulvorlesung, Drehbuch

- *PostScript* ist eine universelle Programmiersprache (*General-Purpose Programming Language*) mit leistungsfähigen Graphikbausteinen
(*Built-in Graphics Primitives*).
- *PostScript* ist eine Sprache zur dynamischen Seitenbeschreibung (*Page Description Language*).
- *PostScript* ist ein interaktives System zur Steuerung von Druckern und Bildschirmen (*System for Controlling Raster Output Devices*)
- *PostScript* ist ein Austauschformat für Seitenbeschreibungen, unabhängig von speziellen Anwendungen und Geräten (*Application- and Device-Independent Interchange Format*)

Trainingsplan

Das Kapitel „PostScript^{TM}-Paradigma — Kommandos überall“ gibt einen Überblick über:

- die Notation von Kommandos (Befehlen),
↪ Seite 20 . . .
- einen Programmiereinstieg anhand von einigen PostScript-Kostproben,
↪ Seite 29 . . .
- die Aufgaben von *Stacks* mit *Push*- und *Pop*-Operationen,
↪ Seite 36 . . .
- die elementaren Datentypen und
↪ Seite 39 . . .
- die Möglichkeiten der Gestaltung von (Druck-)Seiten anhand eines Beispiels mit *Clipping Region*.
↪ Seite 41 . . .

3.1 Notation von Kommandos

Syntax, Semantik und Pragmatik sind prägende Aspekte jeder Programmiersprache, daher gestalten sie auch die Art und Weise wie wir in PostScript formulieren und notieren. Wie üblich, so sind sie, auch bei PostScript, nicht völlig neu erfunden, sondern basieren auf verschiedenen Programmiersprachen. Die PostScript-Syntax orientiert sich primär an der der Programmiersprache *Forth*[2] und damit an der Postfix-Notation, auch *Umgekehrten Polnischen Notation* genannt, das heißt, der Operator folgt seinen Operanden.

3.1.1 Postfix-Notation

Der Begriff Postfix-Notation (UPN ≡ *Umgekehrte Polnischen Notation*) klingt zunächst kompliziert und möglichweise unnatürlich. Damit wird jedoch nur die Reihenfolg von Substantiv (Objekt) und Verb (Operation) thematisert. Man formuliert in UPN beispielsweise einen Alltagsvorgang folgendermaßen:

1. Hände waschen
2. Brot schneiden
3. Brot belegen

und nicht:

1. Wasche Hände!
2. Schneide Brot!
3. Belege Brot!

Postfix-Notation

Man gehe zu einer Stelle auf einer Seite, die durch die Koordinaten mit $x = 150$ Einheiten und $y = 200$ Einheiten definiert ist. Diese Aufgabe wird in PostScript, also in seiner Postfix-Notation, folgendermaßen notiert:

[2] *Forth* wurde von Charles H. Moore $\approx$ 1969 entwickelt. *Forth* ist eine Progammiersprache (incl. Betriebssystem & Entwicklungsumgebung) deren Hauptdatenstruktur der *Stack* ist. *Forth*-Konstrukte werden in der *Umgekehrten Polnischen Notation* formuliert. Ein Interpreter für *Forth* kann sehr ressourcenschonend gebaut werden. *Forth* eignet sich daher besonders gut für die Programmierung von kleinen Rechnern (z. B. Microcontrollern).

```
150 200 moveto
```

Wollte man beispielsweise zur Zahl 20 die Zahl 40 addieren und das Ergebnis anschließend durch 2 teilen, dann stellt sich diese Aufgabe in PostScript folgendermaßen dar:

```
20 40 add 2 div
```

3.1.2 Präfix-Notation

Wären die beiden genannten Beispielaufgaben in der gegensätzlichen Notation, der sogenannten Präfix-Notation (Prefix-Notation), auch *Polnische Notation*[3] genannt, zu formulieren, wie sie beispielsweise LISP (*List Processing*) nutzt, dann wäre zu notieren:

```
(moveto 150 200)
```

und

```
(div (add 20 40) 2)
```

Das Datenmodell in PostScript verfügt über Bausteine einer modernen Programmiersprache wie zum Beispiel Zahlen, Zeichenketten (*Strings*) und Matrizen (*Arrays*). Zur Manipulation von Programm und Daten ist bedeutsam, das PostScript beides quasi gleich, als Daten behandelt (↪ Abschnitt 3.4 S. 39). An dieser Stelle halten wir nur fest, die Sprache PostScript verfügt über weitreichende Möglichkeiten und ist für den Soforteinstieg hinreichend einfach zu nutzen — letztlich aufgrund ihrer interpretativen Ausführung.

Es gibt viele Programme zur Ausführung (Interpretation) von PostScript-Quellcode, beispielsweise die Open-Source-Software `GSview` von *Ghostgum Software Pty. Ltd.* oder das lizenzpflichtige Softwarepaket `Corel` von *Corel Corporation* oder die Programme in vielen Laserdruckern, beispielsweise im Drucker `HP ColorLaserJet4600PS`.

[3] Die *Polnische Notation* auch Łukasiewicz-Notation genannt verdankt ihren Namen dem polnischen Mathematiker Jan Łukasiewicz, der sie 1920 vorstellte. Diese Notation, bei der der Operator vor den Operanden steht, bedarf keiner Klammerung, da die Präzedenz der Operationen klar ist. Der gleiche Effekt wird erreicht, wenn der Operator nicht vor den Operanden, sondern danach steht. Diese Postfix-Notation wurde später ebenfalls von Łukasiewicz entwickelt.

3.2 PostScript-Kostproben

3.2.1 Einfache Linie

Als Einstieg soll der PostScript-Quellcode für die Darstellung einer einfachen Linie notiert werden. Dazu nutzt man seinen Lieblingseditor, zum Beispiel `Emacs` oder `Microsoft Word` oder ... . Mit diesem erzeugt man eine Datei mit dem Namen `examplePSa` und dem Suffix `ps` (Extension).

PostScript *LanguageLevel*

Jede PostScript-Quellcode-Datei beginnt mit einer Zeile, die definiert, das es sich um PostScript handelt (`%!PS`) und um welchen Sprachlevel es geht. Man unterscheidet in der Entwicklung von PostScript Phasen, die man als Sprachlevel (*LanguageLevel*[4]) bezeichnet. Im folgenden wird aus Einfachheitsgründen generell der Level `Adobe-3.0` angenommen. Die Datei `examplePSa.ps` enthält daher folgende erste Zeile:

```
%!PS-Adobe-3.0
```

DIN A4 Diese Linie soll auf einer DIN A4 Seite dargestellt werden. Eine DIN A4 Seite ist $21cm$ breit und $29,70cm$ hoch. In PostScript-Einheiten ausgedrückt, ist die DIN A4 Seite $595PSunit$ breit und $842PSunit$ hoch. Der Mittelpunkt M der Seite liegt dann bei $M(298, 421)$. Ein Zentimeter entspricht damit $28,35PSunit$ und ein Inch $72PSunit$.

DIN A4

Den Anfangspunkt (A) der Linie setzt man mit dem Operator `moveto`. Mit dem Operator `lineto` zieht man die Linie bis zum gewünschten Endpunkt (E). Die Beispielslinie soll in PostSript-Einheiten gehen von $A(150, 200)$ bis $E(300, 400)$. Dazu wird in der Datei `examplePSa.ps` notiert:

```
150 200 moveto
300 400 lineto
```

[4] Im Computerjargon auch *Magic Number* genannt.

Aus holzschnittartiger, stark vereinfachter Sicht basiert das PostScript-Modell auf Graphik-Objekten deren Konturen spezifiziert werden. Mit dem PostScript-Konstrukt `closepath` beendet man die Spezifikation einer Kontur und geht zum Ausgangspunkt zurück. Mit dem PostScript-Konstrukt `stroke`[5] wird die Kontur als Linie dargestellt. Hat man beispielsweise ein Fläche spezifiziert dann wird mit `stroke` ihr Rand (↪ z. B. S. 32) dargestellt und mit dem PostScript-Konstrukt `fill`[6] ihre Fäche (↪ z. B. S. 33). Die Datei `examplePSa.ps` wird daher ergänzt um folgende Zeilen:

stroke

```
closepath
stroke
```

Die Darstellung eines spezifizierten Objektes vollzieht das PostScript-Konstrukt `showpage`. Im Sinne des Anspruchs, ein Programm nicht nur lauffähig zu machen, sondern eine hinreichende Qualität zu erreichen, kommentiert man den bisherigen PostScript-Quellcode mit den Konstrukten:

```
%%Creator:
%%Title:
%%CreationDate:
%%EndComments
...
%%EOF
```

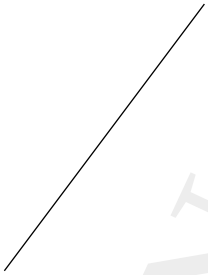
Damit ist die erste simple Graphik, eine einfache Linie von $A(150, 200)$ bis $E(300, 400)$, vollständig programmiert und kann nun von einem PostScript-Interpretor abgearbeitet werden.

Die Abbildung 3.1 S. 31 zeigt das Ergebnis, nachdem die Datei examplePSa.ps in die *Encapsulated PostScript* Datei `examplePSa.eps` konvertiert wurde und dann in ein LaTeX-Dokument eingebunden wurde. Aus diesem LaTeX-Dokument wurde über das Programm `DVIPS` eine PostScript-Datei erzeugt, die auch den PostScript-Quellcode von `examplePSa.eps` enthält.

EPS

PS⟹EPS Die Konvertierung von PostScript zu *Encapsulated Post-*

[5] `stroke` ≡ mit einem Strich kennzeichnen
[6] tt fill ≡ (sich) füllen, ausfüllen



Legende:
PostScript-Quellcode ↪ S. 31

Abbildung 3.1: PostScript: Einfache Linie

Script (EPS) geschieht durch die Einfügung einer Positionierungsangabe der Graphik als Rechteck (*BoundingBox*).

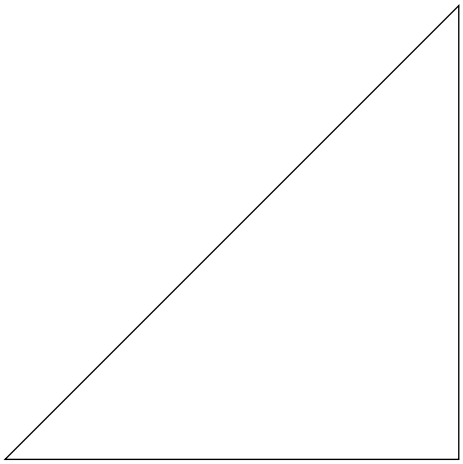
Die Datei `examplePSa.eps` enthält daher die beiden Kopfzeilen:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 148 199 302 401
```

und sonst den Inhalt der Datei `examplePSa.ps`. Die `BoundingBox` wird durch ihre linke untere Ecke (LU) und ihre rechte obere Ecke (RO) in Form von $PSunit$-Angaben spezifiziert. In diesem Fall also mit den Punkten $LU(148, 199)$ und $RO(302, 401)$.

PostScript-Quellcode `examplePSa`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Einfache Linie!
%%CreationDate: 08-Oct-2005
%%EndComments
150 200 moveto
300 400 lineto
closepath
stroke
showpage
%%EOF
```



Legende:
PostScript-Quellcode ↪ S. 32

Abbildung 3.2: PostScript: Einfaches Dreieck

3.2.2 Einfaches Dreieck

PostScript-Quellcode `examplePSb`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Einfaches Dreieck!
%%CreationDate: 08-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
4 cm 4 cm moveto
16 cm 16 cm  lineto
16 cm  4 cm lineto
closepath
stroke
showpage
%%EOF
```

3.2.3 Grau ausgefülltes Dreieck

Legende:
PostScript-Quellcode ↪ S. 32

Abbildung 3.3: PostScript: Graues Dreieck

PostScript-Quellcode `examplePSc`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Grau ausgefuelltes Dreieck!
%%CreationDate: 08-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
4 cm 4 cm moveto
0.8 setgray
16 cm 16 cm  lineto
16 cm  4 cm lineto
closepath
fill
showpage
%%EOF
```

3.2.4 Hello World!

Legende:
PostScript-Quellcode ↪ S. 33

Abbildung 3.4: PostScript: Gedrehter Schriftzug

PostScript-Quellcode `examplePSd`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Hello World!
%%CreationDate: 08-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
/Palatino-Roman findfont
83 scalefont
setfont
4 cm 4 cm moveto
0.4 setgray
45 rotate
(Hello World!) show
showpage
%%EOF
```

Legende:
PostScript-Quellcode ↪ S. 35

Abbildung 3.5: PostScript: `Hello World!` mit grauem Dreieck

3.2.5 Zusammengesetzte Kostprobe

PostScript-Quellcode `examplePSe`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Hello World mit grauem Dreieck!
%%CreationDate: 08-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
4 cm 4 cm moveto
0.8 setgray
16 cm 16 cm  lineto
16 cm  4 cm lineto
closepath
fill
/Palatino-Roman findfont
83 scalefont
```

```
setfont
4 cm 4 cm moveto
0.4 setgray
45 rotate
(Hello World!) show
showpage
%%EOF
```

3.3 Stack

Bisher wurde ein PostScript-Objekt entweder ausgefüllt dargestellt (`fill`) oder es wurde seine Kontur als Strich gezeichnet (`stroke`). Wollte man beispielsweise das graue Dreieck in der Abblidung 3.5 S. 35 umranden, dann könnte man es nochmal notieren und mit `stroke` spezifizieren. Ein solche Lösung hätte den Nachteil, dass der PostScript-Quellcode sehr lang und damit wenig durchschaubar würde (Mangel an Transparenz!). Bei der besseren Lösung (↪ Abbildung 3.6 S. 37; Quellcode S. 38) hält man einen spezifizierten Zustand der Graphik fest und setzt dann später wieder auf diesen auf. Zum Festhalten des Graphikzustandes dient das PostScript-Konstrukt `gsave`, zum Wiederaufsetzen das Konstrukt `grestore`. Beide Konstrukte arbeiten mit dem *Graphik State Stack*, wobei `gsave` eine *Push*-Operation und `grestore` eine *Pop*-Operation durchführt.

gsave

grestore

3.3.1 Last-In-First-Out (LIFO) —*Stack*: *Push* und *Pop*

Ein klassischer *Stack* (≡ Stapelspeicher) ist eine Datenstruktur mit festgelegen Operationen zu ihrer Manipulation. Das Hinzufügen von Daten erfolgt ausschließlich, indem diese Daten auf den Stapel „oben aufgelegt“ werden (*Push*-Operation). Das Ändern des Stapelspeichers ist nur möglich durch Entnahme der Daten, die auf dem Stapel „oben aufliegen“ (*Pop*-Operation). Hinweis: Zur Effizenzsteigerung ist bei manchem *Stack*, wie auch bei PostScript, der Lesezugriff auf tiefer liegende Daten möglich. Dadurch wird ein Umschichten eines Stapels, mit Hilfe eines anderen Stapels vermieden.

Programmiersprachen wie Java, C oder FORTRAN verstecken ihr Stack-Modell vor der Manipulation durch den Programmierer. PostScript jedoch ermöglicht, wie bei interpretierten Sprachen üblich, die direk-

Legende:
PostScript-Quellcode ↪ S. 38

Abbildung 3.6: PostScript: `Hello World!` mit umrandetem Dreieck

te Kontrolle und den Zugriff auf seine Stacks. Bei PostScript sind vier Stacks zu unterscheiden:

- *Operand Stack* wird verwendet als Speicher der Operanden für fast alle PostScript Operatoren.
- *Dictionary Stack* wird verwendet als Speicher für Objekte, die mittels ihres Namens auffindbar sind (*Naming Contexts*, *Name-Lookup-Mechanisms*).
- *Graphic State Stack* wird verwendet als Speicher, für Zustände der Graphik, wie sie beispielsweise im folgenden Quellcode (S.38) mit `gsave` und `grestore` genutzt werden.
- *Execution Stack* wird verwendet als Speicher für ausführbare Prozeduren (*Executable Procedures*), insbesondere zum Zeitpunkt ihrer Ausführung.

3.3.2 *Graphic State Stack*-Beispiel

In diesem PostScript-Beispiel wird die Stärke der Umrandungslinien, mit der sie dann vom `stroke`-Konstrukt dargestellt werden, explizit gesetzt (`setlinewidth`). Das Zusammentreffen dieser Linien wird mit dem Konstrukt `setlinejoin` gestaltet. Dieser Wert des Arguments bestimmt den Grad der „Abrundung“.

PostScript-Quellcode `examplePSf`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Hello World mit umrandetem Dreieck!
%%CreationDate: 08-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
4 cm 4 cm moveto
16 cm 16 cm  lineto
16 cm  4 cm lineto
closepath
gsave
0.8 setgray
fill
grestore
0.9 setgray
4.5 cm setlinewidth
2 setlinejoin
stroke
/Palatino-Roman findfont
83 scalefont
setfont
4 cm 4 cm moveto
0.4 setgray
45 rotate
(Hello World!) show
showpage
%%EOF
```

3.4 Datentypen

Ein PostScript-Stack kann Elemente von unterschiedlichen Datentypen speichern. Dabei unterscheidet man einfache Datentypen (*Simple Data Types*) und zusammengesetzte Datentypen (*Composite Data Types*).

Simple Data Types Beispiele:

```
true    % Boolean Value
7       % Integer Value
-13     % Integer Value
3.141   % Real Value
8#377   % Radix Value
        % (Octal representation of 255)
```

Composite Data Types Beispiele:

```
 (Nachhaltigkeit!) % Character String
<0123456789ABCDEF>% Hexadecimal String
[ 1 2 3 4 5 ]     % Array Object
{ 72 mul }        % Executable Array Object
```

Die Werte in einer Matrix (*Array*) können auch das Ergebnis einer Auswertung beim Einleseprozess sein. Zum Beispiel wird aus dem folgenden notierten Eingabedaten:

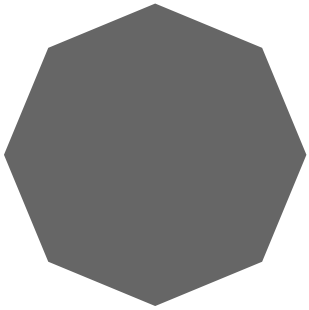
```
[ 7 2 mul 2 2 mul ]
```

ein *Array* mit nur zwei Elementen:

```
[ 14 4 ]
```

Das Beispiel `octagonPS` (↪Abbildung 3.7 S. 40, Quellcode S. 40) nutzt diese Array-Eigenschaft zur Berechnung der Eckpunkte des Achtecks.

```
[  45 cos radius mul    45 sin radius mul ]
  % 1. Eckpunkt rechts oben im Gegenuhrzeiger
```



Legende:
PostScript-Quellcode ↪ S. 40

Abbildung 3.7: PostScript: *Octagon*-Figur

Mit den Konstrukt `get` wird auf das Array zugegriffen, wobei in PostScript das erste Element mit den Index 0 adressiert wird (*zero-based*). Das Konstrukt `aload` transferiert die Elemente eines Array zum *Operand Stack*. Im Beispiel also die berechneten x,-y-Werte eines Eckpunktes. Das Konstrukt `pop` entfernt des oberste Element des Stacks, hier vom Datentyp Array.

`aload`

PostScript-Quellcode `octagonPS`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Octagon
%%CreationDate: 08-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
/radius 4 cm def
/cornerpoints [
[  45 cos radius mul    45 sin radius mul ]
[  90 cos radius mul    90 sin radius mul ]
[ 135 cos radius mul   135 sin radius mul ]
[ 180 cos radius mul   180 sin radius mul ]
[ 225 cos radius mul   225 sin radius mul ]
[ 270 cos radius mul   270 sin radius mul ]
[ 315 cos radius mul   315 sin radius mul ]
```

```
[ 360 cos radius mul   360 sin radius mul ]
] def
%%EndProlog
4.2 cm 4.2 cm translate
cornerpoints 0 get aload pop moveto
cornerpoints 1 get aload pop lineto
cornerpoints 2 get aload pop lineto
cornerpoints 3 get aload pop lineto
cornerpoints 4 get aload pop lineto
cornerpoints 5 get aload pop lineto
cornerpoints 6 get aload pop lineto
cornerpoints 7 get aload pop lineto
closepath
0.4 setgray
fill
showpage
%%EOF
```

3.5 Clipping Region

Die Spezifikation eines Pfades (*Path*) lässt sich auch zum Schneiden eines komplexen Seitenausschnittes (*Clipping*[7] *Region*) benutzen. An die Stelle des PostScript-Konstruktes `stroke` oder `fill` tritt dann das Konstrukt `clip`, wie das folgende Beispiel (↪ PostScript-Quellcode S. 42) zeigt. Der Ausschnitt ist dort als Ellipse spezifiziert. Die Ellipse wird als Kreisbogen von 0 bis 360 Grad und einem unterschiedlichen Skalierungsfaktor in Richtung der Breite (x-Werte) und Höhe (y-Werte) definiert und zwar folgendermaßen:

```
1.2 1.8 scale
0 0 2.75 inch 0 360 arc
```

Der Skalierungsfaktor der x-Werte ist 1.2, der der y-Werte 1.8. Das PostSript-Konstrukt `arc` hat folgende Syntax:

$$x_{Mittelpunkt}\ y_{Mittelpunkt}\ Radius\ Start_{Winkel}\ End_{Winkel}\ \texttt{arc}$$

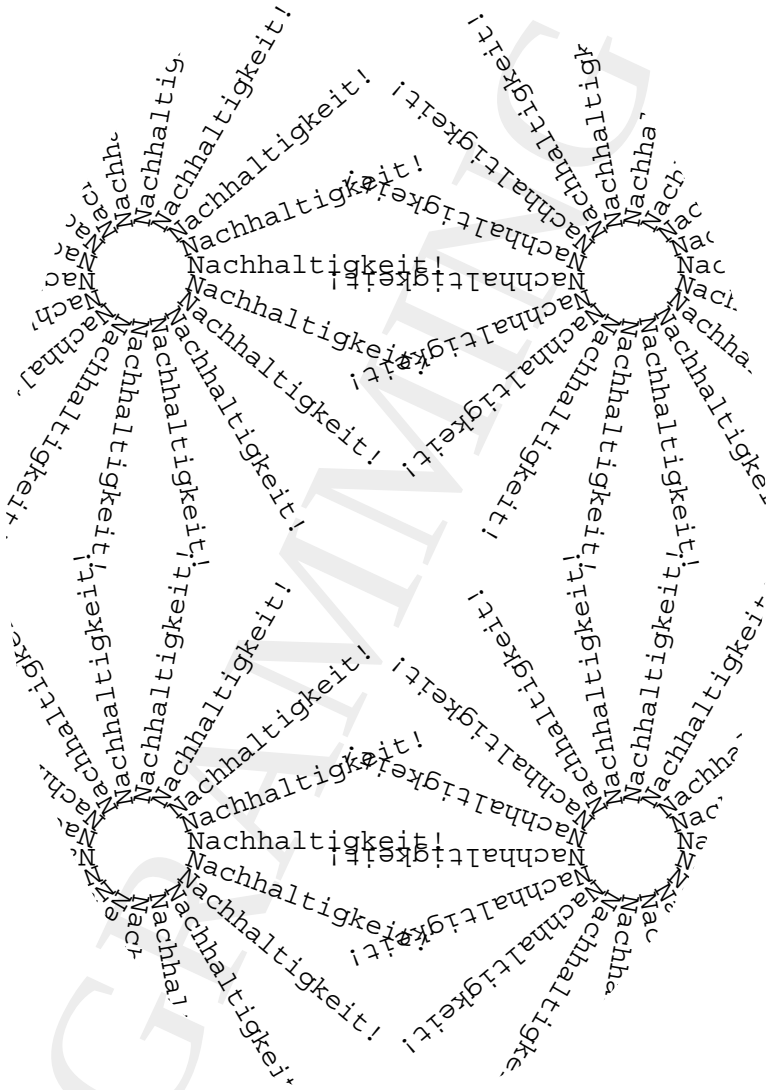
Die Werte für $Start_{Winkel}$ und End_{Winkel} werden in Grad von der x-Achse in Gegen-Uhrzeiger-Richtung ermittelt. Im Beispiel ist ein Kreisbogen im Mittelpunkt (0,0) mit einem Radius von $2.75inch$ von 0^0 bis

[7] Clipping ≡ (Be)Schneiden, Stutzen, Ausschnitt

360^0 angeben, also ein voller Kreis, der durch die unterschiedliche Skalierung der Achsen zu einer Ellipse wird.

PostScript-Quellcode `clippingRegionPS`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Using a path as clipping region
%%CreationDate: 06-Oct-2005
%%EndComments
%%BeginProlog
/inch { 72 mul } def
%%EndProlog
matrix currentmatrix
4.25 inch 5.5 inch translate
1.2 1.8 scale
0 0 2.75 inch 0 360 arc
closepath
clip
newpath
setmatrix
/Courier findfont
18 scalefont
setfont
/TextOnCircle {
    gsave
    translate
    18 {
        0.4 inch 0 moveto
        (Nachhaltigkeit!) show
        20 rotate
    } repeat
    grestore
} def
2.125 inch 7.75 inch TextOnCircle
6.375 inch 7.75 inch TextOnCircle
2.125 inch 2.75 inch TextOnCircle
6.375 inch 2.75 inch TextOnCircle
showpage
%%EOF
```

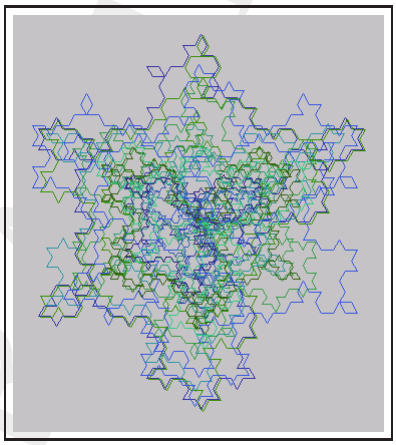


Legende:
Idee für dieses Beispiel entnommen aus [McGilton/Campione92] p. 34
PostScript-Quellcode ↪ S. 42

Abbildung 3.8: Path als Clipping Region

Kapitel 4

Java^{TM}-Paradigma — Objekte überall



Unter den Fehlleistungen der Programmierung wird ständig und überall gelitten :-) : Zu kompliziert, zu viele Mängel, nicht übertragbar, zu teuer, zu spät und so weiter. *The Java Factor*[1] soll es besser machen. Der Hoffnungsträger basiert auf einer beinahe konsequent objektorientierten Programmierung. Sie soll die gewünschte Qualität ermöglichen. Dabei wird Qualität durch die Leistungsfähigkeit, Zuverlässigkeit,

Just a valid application

[1] Titelüberschrift von *Communications of the ACM*, June 1998, Volume 41, Number 6.

Durchschaubarkeit & Wartbarkeit, Portabilität & Anpassbarkeit, Ergonomie & Benutzerfreundlichkeit und Effizienz beschrieben.

Faszination Programmierung schwärmt wohl vom Glanz der „Java-Philosophie“, ist aber nicht euphorisch eingestimmt. Es wäre schon viel erreicht, wenn Sie, liebe Programmiererin, lieber Programmierer, nach dem Arbeiten mit diesem Buch Java^{TM} als ein Akronym für *Just a valid application* betrachten können.

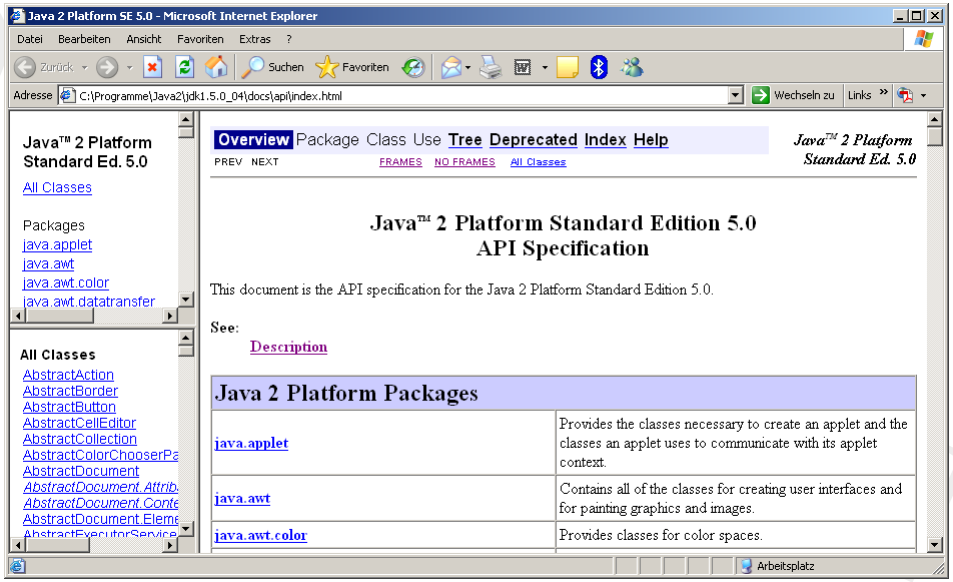
Trainingsplan

Das Kapitel „Java^{TM}-Paradigma — Objekte überall“ gibt einen Überblick über:

- die Java 2 Standard Edition (J2SE)
  ↪ Seite 46 . . .
- die Java3D-Welt (*Virtual Universe* und
  ↪ Seite 53 . . .
- einige Java3D-Kostproben.
  ↪ Seite 58 . . .

4.1 J2SE (Java 2 Standard Edition)

Zum Arbeiten mit Java^{TM} benötigt man die Beschreibung der vorgegebenen Klassen und Schnittstellen (*Interfaces*) der jeweiligen Version. Arbeitet man beispielsweise mit der *Java^{TM} 2 Platform Standard Edition 5.0* dann findet man üblicherweise unter dem Pfad `docs` die benötigte API-Beschreibung (*Application Programming Interface*). Beispielsweise befindet sich diese Dialoghilfe auf meinem Rechner unter:
`file:///C:/Programme/Java2/jdk1.5.0_04/docs/api/index.html`
Die Abbildung 4.1 S. 47 zeigt die Einstiegsseite diese Java-Dialoghilfe.



Legende:
Dialoghilfe für Java^{TM}

Abbildung 4.1: Beispiel: Java API

Üblicherweise werden die ersten Java-Schritte mit ganz einfachen Beispielprogrammen vollzogen. Man erfährt dabei, was der Java-Compiler `javac` und die sogenannte *Java Virtual Machine* `java` machen. Solche elementaren Grundkenntnisse vermittelt mein Manuskript JAVA^{TM}–COACH (↪ [Bonin04b]) `:-)`. Da wir der Informatiktradition folgend schon das obligatorische Einstiegsprogramm „Hello World!“ in Postscript realisiert haben (↪ Abbidung 3.4 S. 34), kann hier unmittelbar eine etwas komplexere erste Kostproben präsentiert werden.

Texteingabe und 2D-Darstellung Mit Hilfe eines Java-Applets, das heißt, einer Java-Klasse eingebunden in eine Web-Seite, wird ein Text in einem Dialogfenster erfasst (↪ Abbildung 4.3 S. 53) und anschließend in einer farbigen Ellipse dargestellt (↪ Abbildung 4.4 S. 54).

Klasse `Text`

```
/**
 *  Applet example
 *
 *@author     Bonin
 *@version    1.2
 */
package de.unilueneburg.as.figure2D;

import java.applet.Applet;
import java.awt.Color;
import java.awt.Font;
import java.awt.font.FontRenderContext;
import java.awt.font.TextLayout;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.util.Random;
import javax.swing.JOptionPane;

public class Text extends Applet
{
   final int LARGE_SIZE = 36;

   private Color color1;
```

```
private Color color2;
private Font largeFont;
private String input;


public Text()
{
   super();
   System.out.println("Text()");

}


public void init()
{
   System.out.println("init()");

   float r1;

   float g1;

   float b1;
   float r2;
   float g2;
   float b2;
   Random generator = new Random();

   /*
    *  Applet Parameters
    *  <param name="..." value="..." />
    */
   r1 = Float.parseFloat(getParameter("red"));
   g1 = Float.parseFloat(getParameter("green"));
   b1 = Float.parseFloat(getParameter("blue"));
   color1 = new Color(r1, g1, b1);

   /*
    *  Second color generated randomly.
    *  nextDouble() returns a random floating-point
    *  number between 0 (inclusive) and 1 (exclusive).
    */
   r2 = (float) generator.nextDouble();
```

```
      g2 = (float) generator.nextDouble();
      b2 = (float) generator.nextDouble();
      color2 = new Color(r2, g2, b2);

      /*
       *  Modal dialog (Swing toolkit)
       *  Waits until the user has entered a string
       *  and clicks on the "OK" button.
       */
      input = JOptionPane.showInputDialog(
            "Your text?");
      if (input.length() == 0)
      {
         input = "No input!";
      }
      largeFont = new Font(
            "Courier", Font.ITALIC, LARGE_SIZE);
   }


   public void paint(Graphics g)
   {

      System.out.println("paint(g)");

      Graphics2D g2 = (Graphics2D) g;

      /*
       *  The font render context is an object, that knows how
       *  to transform letter shapes (which are described as
       *  curves) into pixels.
       */
      FontRenderContext context = g2.getFontRenderContext();

      /*
       *  The TextLayout object gets typographic measurements
       *  of the string TEXT.
       */
      TextLayout layout =
            new TextLayout(input, largeFont, context);

      float xTextWidth = layout.getAdvance();
```

```
      float yTextHeight =
            layout.getAscent() + layout.getDescent();
      float xLeft = (getWidth() - xTextWidth) * 0.5F;
      float yTop = (getHeight() - yTextHeight) * 0.5F;
      float yBase = yTop + layout.getAscent();

      Ellipse2D.Float egg =
            new Ellipse2D.Float(
            xLeft, yTop, xTextWidth, yTextHeight);
      Rectangle2D.Float box =
            new Rectangle2D.Float(
            xLeft, yTop, xTextWidth, yTextHeight);

      g2.setColor(color1);
      g2.fill(egg);
      g2.setColor(color2);
      g2.draw(box);

      g2.setFont(largeFont);
      g2.drawString(input, xLeft, yBase);
   }


   public void stop()
   {
      System.out.println("stop()");

   }


   public void destroy()
   {
      System.out.println("destroy()");
   }
}
```

HTML-Datei `Text.html` **mit Applet über** `<object>`**-Element**

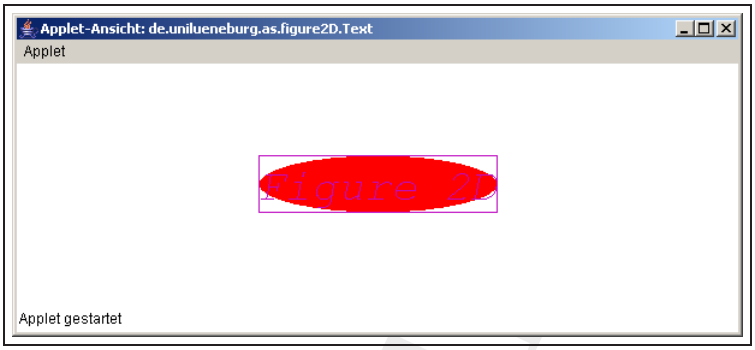
```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Bonin Version 1.0  -->
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<meta http-equiv="Content-Type"
  content="text/html; charset=utf-8" />
<title>Little Applet</title>
</head>
<body>
<h1>Your Text:</h1>
<object
  codetype="application/java"
  classid="java:de.unilueneburg.as.figure2D.Text.class"
  code="de.unilueneburg.as.figure2D.Text"
  width="600" height="200"
  alt="Java: Just A Valid Application">
  <param name="red" value="1.0" />
  <param name="green" value="0.0" />
  <param name="blue" value="0.0" />
</object>
<p>Copyright bonin@uni-lueneburg.de</p>
</body>
</html>
```

Protokoll `Text.log`

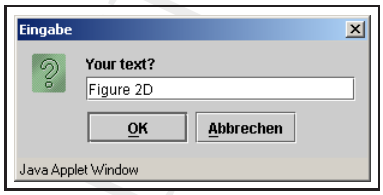
```
d:\bonin\prog\code>java -version
java version "1.5.0_04"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_04-b05)
Java HotSpot(TM) Client VM
  (build 1.5.0_04-b05, mixed mode, sharing)

d:\bonin\prog\code>javac de/unilueneburg/as/figure2D/Text.java

d:\bonin\prog\code>appletviewer Text.html
Text()
init()
paint(g)
paint(g)
stop()
destroy()

d:\bonin\prog\code>
```

Das Ergebnis mit dem *Appletviewer* zeigt die ↪ Abbildung 4.2, S. 53.



Legende:
Modal dialog ↪ Abbildung 4.3, S. 53

Abbildung 4.2: `Text`-Applet mit `appletviewer`



Legende:
Modal dialog: Texteingabe `Figure 2D`

Abbildung 4.3: Eingabefenster von `Text`-Applet

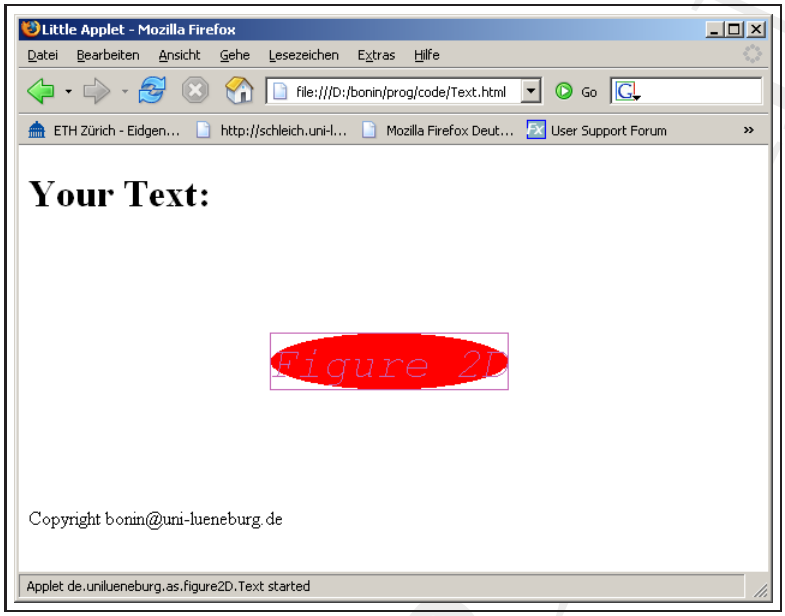
Das Ergebnis mit dem Browser *Firefox 1.0.7*[2] zeigt die ↪ Abbildung 4.4, S. 54. Das Fenster für die Eingabe zeigt die ↪ Abbildung 4.3, S. 53

4.2 Virtual Universe — Java3D-Paket

Eine Java3D-Welt (*Virtual Universe*) wird vom sogenannten *Scene Graph* erzeugt. Der *Scene Graph* setzt sich zusammen aus Objekten (Instanzen der Java3D-Klassen). Er definiert die Geometrie, den Klang, den Ort, die Orientierung und das Erscheinungsbild (*Appearance*) der sicht-

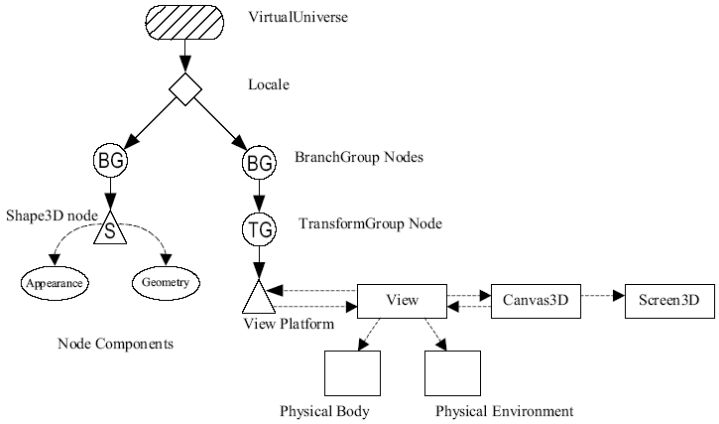
Scene Graph

[2] Mozilla/5.0 (Windows; U; Windows NT 5.1; de-DE; rv:1.7.12) Gecko/20050919 Firefox/1.0.7.



Legende:
Modal dialog ↪ Abbildung 4.3, S. 53

Abbildung 4.4: `Text`-Applet mit Browser *Firefox 1.0.7*



Legende:

Inhalt	≡	linke Teilgraph von `Locale`	(*content branch graph*)
Sichtweise	≡	rechte Teilgraph von `Locale`	(*viewing branch graph*)

Quelle ↪ *The Java 3D Tutorial, Chapter 1, p. 1–5*

Erläuterung der Symbole ↪ Abbildung 4.6 S. 56

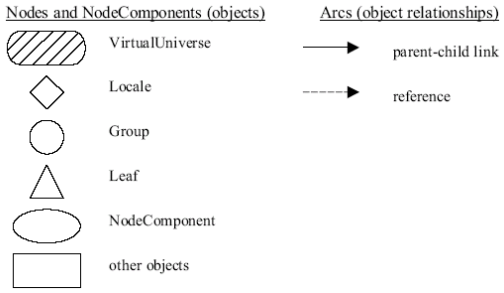
Abbildung 4.5: *Scene Graph* — erstes Beispiel

und hörbaren Objekte. Als Graph setzt er sich aus Knoten und Kanten zusammen.[3] Ein Knoten (*node*) ist ein „Datenelement“ und eine Kante (*arc*) ist eine Beziehung (*Relationship*) zwischen Datenenelementen. Eine Kante bildet daher die Beziehung zwischen Instanzen von Java3D-Klassen ab. Die Abbildung 4.5 S. 55 zeigt ein erstes Beispiel für einen einfachen *Scene Graph*. Die verwendeten Symbole erläutert Abbildung 4.6

Eine übliche Beziehung in einem Graphen ist die Eltern-Kind-Beziehung (*parent-child relationship*). Dabei kann ein Gruppenknoten eine Menge von Kindknoten haben, aber nur einen Elternknoten. In einem solchen Baum hat ein Blatt(knoten) (*leaf node*) keine Kindknoten.

Eine andere Beziehung ist die Referenz. Eine Referenz assoziiert ein *NodeComponent object*, also ein Bestandteil eines Objektes, mit einem *Scene Graph*-Knoten. Die *NodeComponent*-Objekte definieren Attribu-

[3] Mathematisch betrachtet ist der *Scene Graph* ein gerichteter azyklischer Graph (*directed acyclic graph* (DAG)).



Legende:
Quelle ↪ *The Java 3D Tutorial, Chapter 1, p. 1–4*
Einfaches Beispiel ↪ Abbildung 4.5 S. 55

Abbildung 4.6: *Scene Graph* — Symbole

te der Geometrie und des Erscheinungsbildes, die benutzt werden, um das sichtbare Objekt zu berechnen (*render*n).

Ein *Scene Graph Tree* wird gebildet durch Bäume mit einer Wurzel aus *Locale*-Objekte. Die *NodeComponents* und ihre Referenzpfeile sind nicht Teil dieser Bäume. Somit gibt es stets nur einen Weg von der Wurzel (*root*) eines Baums zu einem Blatt (*leaf*). Diesen Weg bezeichnet man als den *leaf nodes scene graph path*. So ein Weg spezifiziert vollständig die Zustandsinformation eines Baumblattes. Die Zustandsinformation umfasst den Ort, die Orientierung und die Grösse des sichtbaren Objektes.

Locale

Ein *Locale*-Objekt stellt einen Referenzpunkt im *Virtual Universe* bereit. Es ist quasi ein Orientierungspunkt (*landmark*) um den Ort von sichtbaren Objekten im Universum festzulegen.

Ein *BranchGroup*-Objekt ist die Wurzel eines Teilgraphen (*subgraph* auch *branch graph* (BG)). Es gibt zwei Kategorien von Teilgraphen:

- *content branch graph* und
  Er spezifiziert die Geometrie, das Erscheinungsbild, Verhalten, den Ort, den Klang und das Licht.
- *viewing branch graph*
  Er spezifiziert den *viewing*-Ort und die -Richtung.

Die Klasse `SimpleUniverse` aus dem Paket `com.sun.j3d.utils.universe` vereinfacht die Java3D-Programmierung wesentlich, da sie einen schon definierten *view branch graph* nutzt. Ihr Konstruktor erzeugt einen *Scene Graph*, der das *VirtualUniverse*-Objekt und *Locale*-Objekte sowie einen vollständigen *view branch graph* umfasst.

`SimpleUniverse`

Mit dieser Vereinfachung programmieren wir unsere ersten Java3D-Kostproben nach folgender Vorgehensweise:

1. Schritt: Erzeugen einer dreidimensionalen „Leinwand“, also eines `Canvas3D`-Objektes
2. Schritt: Erzeugen eines `SimpleUniverse`-Objektes mit Referenz zu unserem `Canvas3D`-Objekt
3. Schritt: Anpassen (*customizing*) des `SimpleUniverse`-Objektes
4. Schritt: Konstruieren *content branch*
5. Schritt: Kompilieren *content branch graph*
6. Schritt: Einfügen *content branch graph* in *Locale* des *SimpleUniverse*

Das Java3D-API umfasst mehr als hundert Klassen, die im Paket `javax.media.j3d` zusammengefasst sind. Üblichwerweise werden diese Klassen als *Java3D core classes* bezeichnet. Dieses Kernklassenpaket enthält die notwendigen *low-level* Klassen. Es wird ergänzt durch das Paket `com.sun.j3d.utils`, das als *Java3D utility classes* bezeichnet wird. Die *utility classes* bilden eine leistungsfähige Ergänzung zum Kern. Sie umfassen folgende Kategorien:

- Laden von Inhalt (*Content Loaders*),
- Konstruktion des *Scene Graph*,
- Geometrie und
- Komfort (*convenience utilities*).

Zusätzlich zu diesen beiden Paketen werden Klassen der Pakete `java.awt` und `javax.vecmath` genutzt. Das erste enthält das *Abstract Windowing Toolkit* (AWT). Es dient zum Erzeugen eines Fensters, in

AWT

dem das Ergebnis angezeigt werden kann. Das zweite enthält die Klassen der Vektormathematik, also Punkte, Vektoren, Matrizen und andere mathematischen Objekte.

Im folgenden bezeichnen wir mit „sichtbaren Objekt“ (*visual object*) ein „Objekt im *Scene Graph*“, zum Beispiel eine Kugel (*sphere*) oder einen Quader (*cube*). Der Begriff „Objekt“ verweist stets auf eine Instanz einer Klasse, während der Begriff „Inhalt“ benutzt wird, um sich auf ein sichtbares Objekt im *Scene Graph* als Ganzes zu beziehen.

4.3 Java3D-Kostproben

In der Softwareentwicklung wird traditionell mit einem Programm „Hello World!“ gestartet (↪ PostScript-Beispiel 3.2.4 S. 33). Dieses zeigt üblicherweise eine entsprechende Zeichenkette auf dem jeweiligen Standardausgabegerät. In der Java3D-Welt wäre eine solche Zeichenkette kein passender Einstieg, sondern eher eine „Beleidigung“ für die Java3D-Schöpfer `:-)`.

Wir steigen daher mit dem Programm „Hello Universe“ ein. Anschließend bringen wir dann auch die obligatorische „Hello World!“-Meldung allerdings in einer grafisch anspruchsvolleren Form (↪ Abschnitt 23 S. 71).

4.3.1 `HelloUniverse`

„Hello Universe“ zeigt einen Zylinder im Universum und zwar mit den Polygonlinien und ihren Eckpunkten (*Vertices*) (↪ Abbildung 4.7 S. 59). Sicherlich ist der Quellcode zunächst nicht einfach nachvollziehbar. In diesem Kapitel werden die genutzten Java3D-Konstrukte nach und nach eingehend erläutert. Also keine Sorge — gleich wird es einfacher und es kommt die Zeit, in der Sie diesen Einstieg gut verstehen.

Abbildung 4.7: Beispiel: `HelloUniverse`

Legende:
Quellcode ↪ S. 66

Die Superklasse für `HelloUniverse` (↪ Quellecode S.66) ist ein Applet. Üblicherweise ist ein Applet ein (kleines) Java-Programm, das konzipiert wurde um nicht eigenständig zu laufen, sondern eingebettet in eine andere Anwendung. In der Regel ist diese andere Anwendung ein marktüblicher Web-Browser. Ein Applet erfordert die Implementation von Methoden wie `init()` und `destroy()`. In der ersten gestalten wir unser `SimpleUniverse`, in der zweiten wenden wir die Methode `cleanup()` auf unser `SimpleUniverse` an. Um das Applet auch als eigenständige Java-Applikation aufrufen zu können, bauen wir eine `main`-Methode ein und nutzen darin die Klasse `Mainframe`. Den *Scene Graph* konstruieren wir mit der Methode `createSceneGraph()`. Damit hat unsere Klasse `HelloUniverse` folgende Grundstruktur:

create-Scene-Graph()

Grundstrukur von `HelloUniverse`

```
public class HelloUniverse extends Applet
{
   private SimpleUniverse u = null;

   private HelloUniverse() {...};

   public BranchGroup createSceneGraph()
   {

      ...
      Cylinder cylinder = new Cylinder(...)
      ...

   }


   public void init()
   {
      u = new SimpleUniverse(...);
      ...
      u.addBranchGraph(this.createSceneGraph());

   }
```

```
    public void destroy()
    {
       u.cleanup();
    }


    public static void main(String[] args)
    {
       new MainFrame(new HelloUniverse(), 300, 400);
    }

}
```

Mit Hilfe eines Objektes der Klasse `Canvas3D` lösen wir die Aufgabe eine dreidimensionale Szene auf der graphischen Benutzungsoberfläche[4] darzustellen. Zur Konstruktion dieses Objektes sind Daten über die konkrete graphische Konfiguration erforderlich. Diese erhalten wir mit Hilfe der Methode `getPreferredConfiguration()` von der Klasse `SimpleUniverse`. Außerdem gilt es, die Position des Beobachters, also die Position der `ViewingPlatform`, zu spezifizieren. Ohne Spezifikation liegt diese Position im Koordinatennullpunkt, also bei $P(0,0,0)$. Mit der Methode `setNominalViewingTransform()` modifizieren wir diese Position, so dass sich der Beobachter etwas entfernt vom Nullpunkt des `SimpleUniverse` befindet.[5] Die `init`-Methode ist damit wie folgt konstruiert:

Detail `init()`

```
    public void init()
    {
       this.setLayout(new BorderLayout());
       GraphicsConfiguration config =
             SimpleUniverse.
                getPreferredConfiguration();
       Canvas3D c = new Canvas3D(config);
       this.add("Center", c);
       u = new SimpleUniverse(c);
```

[4] Häufig auch als „Benutzeroberfläche“ bezeichnet — obwohl die Oberfläche des Benutzers Haut ist.

[5] Die *Nominal Viewing Distance* beträgt ungefähr $2,4m$. In dieser Entfernung füllen Objekte von einer Höhe bzw. Breite von $2m$ die „Bildplatte“ (*image plate*).

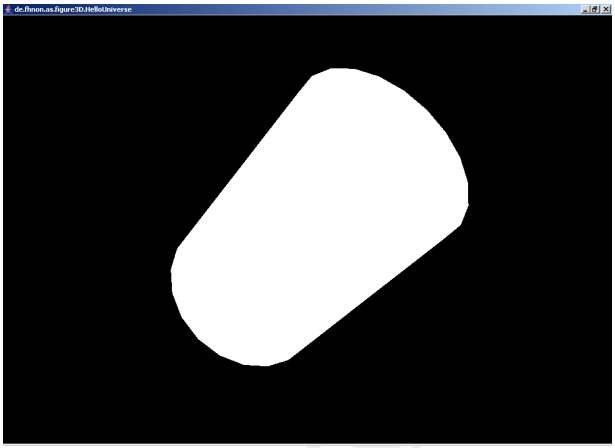
```
        u.getViewingPlatform().
            setNominalViewingTransform();
        u.addBranchGraph(
            this.createSceneGraph());

    }
```

Mit der Methode `createSceneGraph()` erzeugen wir ein Objekt `bg` der Klasse `BranchGroup`, modifizieren dieses durch das Hinzufügen eines Zylinders, dessen Ausrichtung und Aussehen vorher spezifiziert wurde und geben `bg` aus Performance-Gründen in compilierter Form[6] als Wert zurück.

Detail `createSceneGraph()`

```
    public BranchGroup createSceneGraph()
    {
        BranchGroup bg = new BranchGroup();

        // Ausrichtung
        TransformGroup tg = new TransformGroup();
        ...
        tg.setTransform(...);

        // Aussehen
        Appearance app = new Appearance();
        ...
        Material m = new Material(...);
        // Licht einschalten
        m.setLightingEnable(true);
        app.setMaterial(m);

        ...

        // Zylinder mit Aussehen erzeugen
        Cylinder cylinder =
              new Cylinder(
              0.3f, // radius
              0.9f, // height
```

[6]Hinweis: Mit der Compilierung unterliegt der *Scene Graph* einigen Einschränkungen. Beispielsweise lassen sich die compilierten Elemente nur noch bedingt verändern.



Legende:
Quellcode ↪ S. 66
— Ergebnis mit vereinfachter Methode `createSceneGraph()` (↪ S. 62)

Abbildung 4.8: Beispiel: `HelloUniverse` — vereinfacht

```
            Primitive.GENERATE_NORMALS,
            20,   // xDivision
            7,    // yDivision
            app   // Aussehen
            );

        // Zylinder transformieren
        tg.addChild(cylinder);

        // Zylinder in BranchGroup-Knoten einfügen
        bg.addChild(tg);
        bg.compile();
        return bg;
    }
```

Das Ergebnis einer solchen Methode `createSceneGraph()` zeigt Abbildung 4.8 S. 63. Dieses Ergebnis ist kaum beeindruckender als eine übliche „Hello World“ Textausgabe. Wir ergänzen daher die Methode `createSceneGraph()` durch eine Modifikation der Instanz `app`.

Unsere Änderung des Aussehens bringt ein leichteres Verstehen für

die Polygone, aus denen der Zylinder konstruiert ist und zwar insbesondere für deren Spitzen („Ecken“), den sogenannten *Vertices*. Wir stellen den Zylinder jetzt mit sichtbaren *Vertices* dar und zwar aus 20 Abschnitten für den Umfang des Grundkreises (`xDivision`-Wert) und 7 Abschnitten zur Einteilung der Zylindermantelfläche (`yDivision`-Wert).

Detail `PolygonAttributes`

```
...
      PolygonAttributes polyAtt =
            new PolygonAttributes();
      polyAtt.setPolygonMode(
            PolygonAttributes.POLYGON_LINE);
      polyAtt.setCullFace(
            PolygonAttributes.CULL_NONE);

      app.setPolygonAttributes(polyAtt);
...
```

Der Wert `PolygonAttributes.POLYGON_LINE` sorgt dafür, das nur die Verbindungslinien zwischen den einzelnen *Vertices* gezeichnet werden. Das Ergebnis ist daher eine Darstellung als „Drahtgitter“. Mit dem Wert `PolygonAttributes.` Mit `CULL_NONE` verhindern wir ein *Face-Culling*. Die Basis der Java3D-Maschine, meist `OpenGL` oder `DirectX`, führt ein *Face-Culling* durch, das heißt, bestimmte *Faces*, also Polygone, werden entfernt. Beispielsweise wird beim *Backface-Culling*, die Rückseite eines Objektes nicht gezeichnet.

Face-Culling

Zum Fertigstellen der Klasse `HelloUniverse` bedarf es dann nur noch der Ergänzung des selbst gewählten Paketnamens, hier `de.uniluebeburg.as.figure3D` und der Importierung der genutzten Klassen, also folgender Ergänzungen:

Detail *Pakete*

```
package de.fhnon.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.applet.MainFrame;
```

```
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.universe.SimpleUniverse;

import javax.media.j3d.Appearance;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.Material;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.Transform3D;
import javax.media.j3d.PolygonAttributes;

import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;
```

Nicht jedes Konstrukt in der Klasse `HelloUniverse` ist damit hinreichend erläutert. Beispielsweise bedarf die Klasse `AxisAngle4f` einer Erklärung. Sie ermöglicht eine Drehung um ein definiertes Koordinatensystem. Die Details sind in der *Java3D-API*[7] wie folgt erläutert: *A four-element axis angle represented by single-precision floating point* x, y, z, angle *components. An axis angle is a rotation of angle (radians) about the vector (x,y,z).*

Beim Konstruktor `AxisAngle4f` wird der Winkel, um den gedreht werden soll, über den vierten Parameter angegeben, hier um 90 Grad und zwar mit der Angabe `(float) Math.toRadians(90)`. Die Drehachse selbst legen die ersten drei Parameter fest. Sie sind für die jeweils gewünschte Achse auf den Wert 1f zu setzen, hier also eine Drehung um die x- und die y-Achse.

Auch die Klasse `Material` mit ihren mannigfaltigen Optionen bedarf noch einer eingehenden Erläuterung. Hier seien nur ganz holzschnittartig die Parameter des genutzten Konstruktors skizziert:

- `ambientColor` spezifiziert die Farbe, die mit dem gleichmäßigen Umgebungslicht korrespondiert.

[7] In meiner Installation unter
↪ `file:///D:/bonin/myJava/j3dapi/index.html`.
Quelle für die Java3D-API
↪ `http://java3d.virtualworlds.de` (online 20-Apr-2004)

- `emissiveColor` spezifiziert die Farbe, in der das 3D-Objekt selbst leuchtet.
- `diffuseColor` spezifiziert die Farbe, die reflektiert wird, wenn das Objekt beleuchtet wird.
- `specularColor` spezifiziert die Farbe des Glanzpunktes des Objektes.
- `shininess` ist ein Faktor im Bereich von 1..128, der das Reflexionsverhalten des Materials bestimmt.

Aus Vereinfachungsgründen ist hier stets die Farbe weiß gewählt. Die Klasse `com.sun.j3d.utils.applet.MainFrame` ermöglicht den Aufruf eines Applets als Java-Applikation, wobei der Wert:

- des ersten Parameters das Applet ist, hier durch seinen Konstruktor `HelloUniverse()` erzeugt,
- des zweiten Parameters die Breite des Applets in Pixel, hier 300 und
- des dritten Parameters die Höhe in Pixel, hier 400, angibt.

Klasse `HelloUniverse`

```
/**
 *  "Java 3D Example" Hello Universe
 *
 *@author     Bonin
 *@version    1.1
 */

package de.unilueneburg.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.applet.MainFrame;

import com.sun.j3d.utils.geometry.Primitive;
```

```
import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.universe.SimpleUniverse;

import javax.media.j3d.Appearance;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.Material;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.Transform3D;
import javax.media.j3d.PolygonAttributes;

import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;

public class HelloUniverse extends Applet
{
   public SimpleUniverse u = null;


   public HelloUniverse()
   {
      super();
   }


   public BranchGroup createSceneGraph()
   {
      BranchGroup bg = new BranchGroup();

      // Zum Transformieren der untergeordneten Knoten
      TransformGroup tg = new TransformGroup();

      // Spezifiziert die Transformation, hier Drehung
      Transform3D t3d = new Transform3D();
      t3d.setRotation(
            new AxisAngle4f(
            1f,
            1f,
            0f,
            (float) Math.toRadians(90)));
      tg.setTransform(t3d);
```

```
Appearance app = new Appearance();

Color3f white = new Color3f(1.0f, 1.0f, 1.0f);
Color3f ambientColor = white;
Color3f emissiveColor = white;
Color3f diffuseColor = white;
Color3f specularColor = white;
float shininess = 64.0f;

Material m = new Material(
      ambientColor,
      emissiveColor,
      diffuseColor,
      specularColor,
      shininess);

m.setLightingEnable(true);
app.setMaterial(m);

PolygonAttributes polyAtt =
      new PolygonAttributes();
polyAtt.setPolygonMode(
      PolygonAttributes.POLYGON_LINE);
polyAtt.setCullFace(
      PolygonAttributes.CULL_NONE);

app.setPolygonAttributes(polyAtt);

Cylinder cylinder =
      new Cylinder(
      0.3f,
      0.9f,
      Primitive.GENERATE_NORMALS,
      20,
      7,
      app);

tg.addChild(cylinder);

bg.addChild(tg);
bg.compile();
return bg;
```

```
    }


    public void init()
    {
        this.setLayout(new BorderLayout());
        GraphicsConfiguration config =
              SimpleUniverse.getPreferredConfiguration();
        Canvas3D c = new Canvas3D(config);
        this.add("Center", c);
        u = new SimpleUniverse(c);
        u.getViewingPlatform().
              setNominalViewingTransform();
        u.addBranchGraph(
              this.createSceneGraph());

    }


    public void destroy()
    {
        u.cleanup();
    }


    public static void main(String[] args)
    {
        new MainFrame(
              new HelloUniverse(), 300, 400);
    }

}
```

Protokoll `HelloUniverse.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)
```

```
D:\bonin\artsprog\code>javac de/fhnon/as/figure3D/HelloUniverse.java

D:\bonin\artsprog\code>java de.fhnon.as.figure3D.HelloUniverse

D:\bonin\artsprog\code>appletviewer HelloUniverse.html

D:\bonin\artsprog\code>
```

4.3.2 HelloWorld

In der Java3D-Welt unterscheiden wir zwei Formen von Texten; einen Text einerseits konstruiert mit der Klasse `com.sun.j3d.utils.geometry.Text2D` und andererseits konstruiert mit der Klasse `javax.media.j3d.Text3D`. Ein `Text2D`-Objekt besteht aus rechteckigen Polygonen, wobei der Text durch Anwendung einer Textur realisiert ist. Im Abschnitt 4.3.5 S. 103 nutzen wir ein solches `Text2D`-Objekt. Es wird wie folgt konstruiert:

Text2D

```
Text2D textObject = new Text2D(
  java.lang.String text, // z.B. "Bonin"
  Color3f color, // z.B. new Color3f(0f, 0f, 0f)
  java.lang.String fontName, // z.B. "Serif"
  int fontSize, // z.B. 120
  int fontStyle); // z.B. Font.BOLD
```

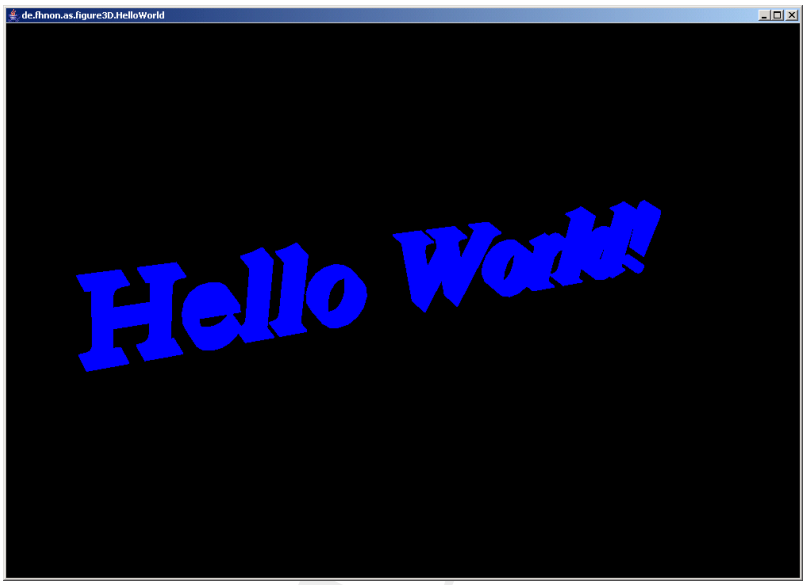
Ein `Text3D`-Objekt ist ein geometrisches 3D-Objekt. Die textliche Geometrie ist eine Extrusion[8] von einem „normalen“ Font. Daher wird zunächst ein entsprechender räumlicher Font auf der Basis eines üblichen AWT-Fonts konstruiert. Dies erfolgt mit der Klasse `javax.media.j3d.Font3D` folgendermaßen:

```
Font3D font3D = new Font3D(
  java.awt.Font font,
    // z.B. new Font("Times", Font.PLAIN, 1)
  FontExtrusion extrudePath);
    // z.B. new FontExtrusion()
```

Der 3D-Text entsteht dann mit Hilfe der Klasse `javax.media.j3d.Font3D` wie folgt:

Text3D

[8] Der Begriff *Extrusion* wird häufig im Zusammenhang mit Pressvorgängen bei der Metallverarbeitung genutzt.



Legende:
Quellcode ↪ S. 71

Abbildung 4.9: Beispiel: `HelloWorld`

```
Text3D textGeom = new Text3D(
  Font3D font3D,    // z.B. font3D siehe oben
  String string,    // z.B. new String("Hello World!")
  Piont3f position, // z.B. new Point3f(0.0f, 0.0f, 0.0f)
  int alignment,    // z.B. Text3D.ALIGN_CENTER
  int path);        // z.B. Text3D.PATH_RIGHT
```

Klasse `HelloWorld`

```
/**
 *  "Java 3D Example" Hello World
 *
 *@author     Bonin
 *@version    1.0
 */

package de.fhnon.as.figure3D;
```

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;
import java.awt.Font;

import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.universe.SimpleUniverse;

import javax.media.j3d.Appearance;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.Font3D;
import javax.media.j3d.FontExtrusion;
import javax.media.j3d.Material;
import javax.media.j3d.Shape3D;
import javax.media.j3d.Text3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.Transform3D;

import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;
import javax.vecmath.Point3f;

public class HelloWorld extends Applet
{
   private SimpleUniverse u = null;


   private HelloWorld()
   {
      super();
   }


   public BranchGroup createSceneGraph()
   {
      BranchGroup bg = new BranchGroup();
      // Zum Transformieren der untergeordneten Knoten
      TransformGroup tg = new TransformGroup();
```

```
        // Spezifiziert die Transformation
        Transform3D t3d = new Transform3D();
        // hier Drehung
        t3d.setRotation(
              new AxisAngle4f(
              1f,
              1f,
              0f,
              (float) Math.toRadians(45)));
        // Maßstab
        t3d.setScale(0.3);
        tg.setTransform(t3d);

        Font3D font3D = new Font3D(
              new Font("Times", Font.PLAIN, 1),
              new FontExtrusion());
        Text3D textGeom = new Text3D(
              font3D,
              new String("Hello World!"),
              new Point3f(0.0f, 0.0f, 0.0f),
              Text3D.ALIGN_CENTER,
              Text3D.PATH_RIGHT);

        Appearance app = new Appearance();

        Material m = new Material();
        m.setEmissiveColor(
              new Color3f(0.0f, 0.0f, 1.0f));
        m.setLightingEnable(true);
        app.setMaterial(m);

        Shape3D textObject = new Shape3D(textGeom, app);

        tg.addChild(textObject);

        bg.addChild(tg);
        bg.compile();
        return bg;
    }
```

```
   public void init()
   {
      this.setLayout(new BorderLayout());
      GraphicsConfiguration config =
            SimpleUniverse.getPreferredConfiguration();
      Canvas3D c = new Canvas3D(config);
      this.add("Center", c);
      u = new SimpleUniverse(c);
      u.getViewingPlatform().
            setNominalViewingTransform();
      u.addBranchGraph(
            this.createSceneGraph());

   }


   public void destroy()
   {
      u.cleanup();
   }


   public static void main(String[] args)
   {
      new MainFrame(
            new HelloWorld(), 1000, 700);
   }

}
```

Protokoll `HelloWorld.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/figure3D/HelloWorld.java

D:\bonin\artsprog\code>java
```

```
  de.fhnon.as.figure3D.HelloWorld

D:\bonin\artsprog\code>
```

4.3.3 `SimpleFigure3D` — 3 Fälle

Die Beispielsidee ist der Java3D-Distribution entnommen und wird auch von Daniel Selman beschrieben (↪ [Selman02] S. 30–36). Der Quellcode der folgenden drei Varianten (`a`, `b` und `c`) ist von mir gestaltet.

Angestrahlte Kugel — 1. Fall

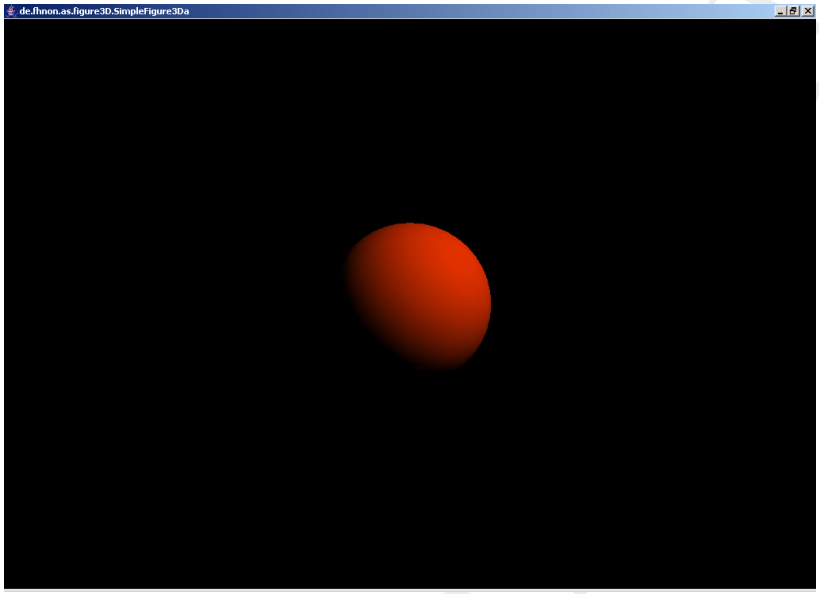
Wir strahlen mit einem Licht, erzeugt mit der Klasse `javax.media.j3d.DirectionalLight`, unser 3D-Objekt an. Das Licht wirkt nur auf das Objekt, wenn dieses in seinen spezifizierten Einflussgrenzen, den sogenannten *Influencing Bounds*, liegt. Diese Einflussgrenzen der jeweiligen Lichquelle spezifizieren wir hier ganz einfach als eine Kugel im Nullpunkt mit dem Radius $200m$. Diese Grenzkugel selbst ist nicht sichtbar. Sie begrenzt nur den Bereich unserer Lichtquelle.

Influencing Bounds

Unser Licht kommt aus einer Richtung, die wir mit Hilfe eines Vektors festlegen. Die Lichtquelle selbst befindet sich in unendlicher Entfernung. Ihre Lichtstrahlen laufen daher parallel. Es findet keine entfernungsabhängige Änderung des Lichtes statt.

Klasse `SimpleFigure3Da`

```
/**
 *  "Java 3D Example" Einfache Kugel
 *
 *@author     Bonin
 *@version    1.1
 */

package de.fhnon.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Sphere;
```



Legende:
Quellcode ↪ S. 75

Abbildung 4.10: Beispiel: `SimpleFigure3Da`

```
import com.sun.j3d.utils.universe.SimpleUniverse;

import com.sun.j3d.utils.applet.MainFrame;

import javax.media.j3d.Appearance;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.Material;
import javax.media.j3d.TransformGroup;

import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

public class SimpleFigure3Da extends Applet
{
   private SimpleUniverse u = null;


   private SimpleFigure3Da()
   {
      super();
   }


   public BranchGroup createSceneGraph()
   {
      BranchGroup bg = new BranchGroup();
      Appearance app = new Appearance();
      Color3f ambientC =
            new Color3f(0.9f, 0.2f, 1.0f);
      Color3f emissiveC =
            new Color3f(0.0f, 0.0f, 0.0f);
      Color3f diffuseC =
            new Color3f(0.9f, 0.2f, 1.0f);
      Color3f specularC =
            new Color3f(0.0f, 0.0f, 0.0f);
      float shininess = 80.0f;

      app.setMaterial(
```

```
            new Material(
            ambientC,
            emissiveC,
            diffuseC,
            specularC,
            shininess));
      Sphere sphere =
            new Sphere(
            0.2f, Primitive.GENERATE_NORMALS, 40, app);
      bg.addChild(sphere);
      return bg;
   }


   public void addLights(BranchGroup bg)
   {
      DirectionalLight light =
            new DirectionalLight(
            new Color3f(1.0f, 1.0f, 0.0f),
            new Vector3f(-1.0f, -1.0f, -1.0f));
      light.setInfluencingBounds(
            this.getBoundingSphere());
      bg.addChild(light);
   }


   public TransformGroup createBehaviors(
         BranchGroup bg)
   {
      TransformGroup objTrans =
            new TransformGroup();
      bg.addChild(objTrans);
      return objTrans;
   }


   BoundingSphere getBoundingSphere()
   {
      return new BoundingSphere(
            new Point3d(0.0, 0.0, 0.0), 200.0);
   }
```

```
    public void init()
    {
       this.setLayout(new BorderLayout());
       GraphicsConfiguration config =
             SimpleUniverse.getPreferredConfiguration();
       Canvas3D c = new Canvas3D(config);
       this.add("Center", c);
       u = new SimpleUniverse(c);
       u.getViewingPlatform().
             setNominalViewingTransform();

       BranchGroup bgRoot = new BranchGroup();
       TransformGroup tg = this.createBehaviors(bgRoot);
       tg.addChild(this.createSceneGraph());
       this.addLights(bgRoot);
       u.addBranchGraph(bgRoot);

    }


    public void destroy()
    {
       u.cleanup();
    }


    public static void main(String[] args)
    {
       new MainFrame(new SimpleFigure3Da(), 300, 400);
    }

}
```

Protokoll `SimpleFigure3Da.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
```

```
  (build 1.4.2-b28, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/figure3D/SimpleFigure3Da.java

D:\bonin\artsprog\code>java
  de.fhnon.as.figure3D.SimpleFigure3Da

D:\bonin\artsprog\code>
```

Angestrahlte Kugel mit Hintergrund — 2. Fall

Textur

Feine Strukturen und Farbenvielfalt zeichnen Objekte der realen Welt aus. Beispielsweise hat ein Ziegelmauerwerk eine große Menge von unterschiedlichen Farben und Steinstrukturen (↪ Abbildung 4.11 S. 81). Dieses hochkomplexe Muster orginalgetreu mit entsprechenden Polygonen und Farbzuweisungen wiederzugeben, wäre zu aufwendig, das heißt, der Ressourcenverbrauch wäre nicht akzeptabel. Man löst dieses Darstellungsproblem mit Hilfe von sogenannten Texturen. Eine Textur ist ein normales Foto (Bild) von der Oberfläche des realen Objektes. Wir legen dann eine solche Textur auf das spezifizierte 3D-Objekt. Für eine Objektdarstellung benötigen wir dann nur Polygone um das 3D-Objekt zu spezifizieren, aber nicht um seine Oberfläche zu beschreiben. Dazu reicht seine Textur. Damit reduzieren wir den Ressourcenverbrauch erheblich. Geboten ist dieses Verfahren insbesondere bei Bewegtbildern.

Das Oberflächenbild laden wir mit der Klasse `com.sun.j3d.utils.image.TextureLoader`. In welcher Lage und Position das Bild auf das Objekt gelegt werden soll, spezifizieren wir mit Hilfe der Klasse `javax.media.j3d.TexCoordGeneration`. Mit dem ersten Parameterwert `SPHERE_MAP` sorgen wir für eine sphärische Projektion. Mit dem zweiten Parameterwert `TEXTURE_COORDINATE_2` spezifizieren wir eine zweidimensionale Textur. Mit der Klasse `javax.media.j3d.TextureAttributes` beeinflussen wir den Vorgang zum Beispiel im Hinblick auf die Qualität. Mit dem Parameterwert `TextureAttributes.NICEST` geben wir der Qualität Vorzug vor der Geschwindigkeit (`FASTEST`).

Klasse `SimpleFigure3Db`

```
/**
```

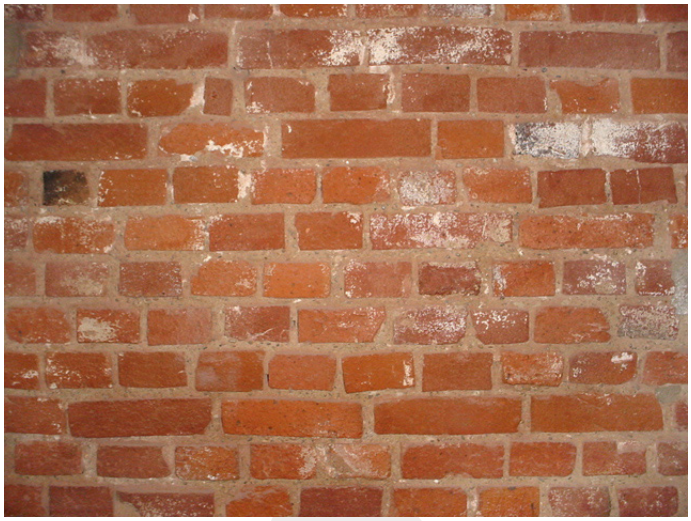
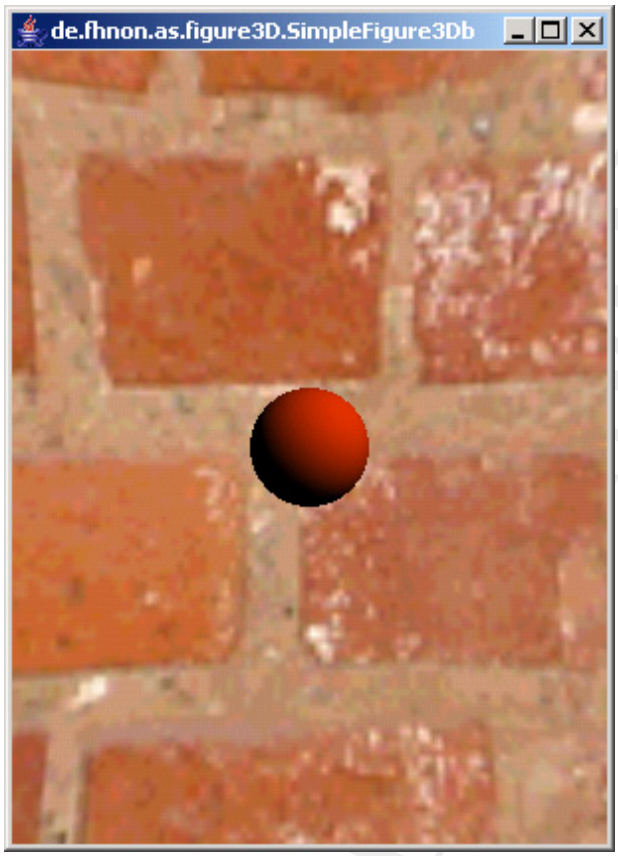


Abbildung 4.11: Hintergrundbild: `brickwork.jpg`

```
 *  "Java 3D Example" Kugel mit Hintergrund
 *
 *@author     Bonin
 *@version    1.1
 */

package de.fhnon.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Sphere;

import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.universe.SimpleUniverse;

import com.sun.j3d.utils.applet.MainFrame;

import javax.media.j3d.Appearance;
```



Legende:
Quellcode ↪ S. 80
Textur ↪ Abbildung 4.11 S. 81

Abbildung 4.12: Beispiel: `SimpleFigure3Db` — Textur `brick-work.jpg`

```
import javax.media.j3d.Background;

import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.Material;
import javax.media.j3d.TexCoordGeneration;

import javax.media.j3d.Texture;
import javax.media.j3d.TextureAttributes;

import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;

import javax.vecmath.Color3f;
import javax.vecmath.Color4f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

public class SimpleFigure3Db extends Applet
{
   private SimpleUniverse u = null;


   private SimpleFigure3Db()
   {
      super();
   }


   public BranchGroup createSceneGraph()
   {
      BranchGroup bg = new BranchGroup();
      Appearance app = new Appearance();
      Color3f ambientC =
            new Color3f(0.9f, 0.2f, 1.0f);
      Color3f emissiveC =
            new Color3f(0.0f, 0.0f, 0.0f);
      Color3f diffuseC =
            new Color3f(0.9f, 0.2f, 1.0f);
      Color3f specularC =
```

```
            new Color3f(0.0f, 0.0f, 0.0f);
      float shininess = 80.0f;

      app.setMaterial(
            new Material(
            ambientC,
            emissiveC,
            diffuseC,
            specularC,
            shininess));
      Sphere sphere =
            new Sphere(
            0.2f, Primitive.GENERATE_NORMALS, 40, app);
      bg.addChild(sphere);
      return bg;
   }


   public BranchGroup createBackground()
   {
      BranchGroup bg =
            new BranchGroup();
      Background back = new Background();
      back.setApplicationBounds(
            getBoundingSphere());
      BranchGroup bgGeometry =
            new BranchGroup();
      Appearance app = new Appearance();

      Texture tex = new TextureLoader(
            "de/fhnon/as/figure3D/brickwork.jpg",
            null).getTexture();
      app.setTexture(tex);

      app.setTexCoordGeneration(
            new TexCoordGeneration(
            TexCoordGeneration.SPHERE_MAP,
            TexCoordGeneration.TEXTURE_COORDINATE_2));
      app.setTextureAttributes(
            new TextureAttributes(
            TextureAttributes.REPLACE,
            new Transform3D(),
```

```
            new Color4f(),
            TextureAttributes.NICEST));

      Sphere sphere = new Sphere(1.0f,
            Primitive.GENERATE_TEXTURE_COORDS |
            Primitive.GENERATE_NORMALS_INWARD, 40, app);

      bgGeometry.addChild(sphere);
      back.setGeometry(bgGeometry);
      bg.addChild(back);
      return bg;
   }


   public void addLights(BranchGroup bg)
   {
      DirectionalLight light =
            new DirectionalLight(
            new Color3f(1.0f, 1.0f, 0.0f),
            new Vector3f(-1.0f, -1.0f, -1.0f));
      light.setInfluencingBounds(
            this.getBoundingSphere());
      bg.addChild(light);
   }


   public TransformGroup createBehaviors(
         BranchGroup bg)
   {
      TransformGroup objTrans =
            new TransformGroup();
      bg.addChild(objTrans);
      return objTrans;
   }


   BoundingSphere getBoundingSphere()
   {
      return new BoundingSphere(
            new Point3d(0.0, 0.0, 0.0), 200.0);
   }
```

```
   public void init()
   {
      this.setLayout(new BorderLayout());
      GraphicsConfiguration config =
            SimpleUniverse.getPreferredConfiguration();
      Canvas3D c = new Canvas3D(config);
      this.add("Center", c);
      u = new SimpleUniverse(c);
      u.getViewingPlatform().
            setNominalViewingTransform();
      u.addBranchGraph(this.createBackground());

      BranchGroup bgRoot = new BranchGroup();
      TransformGroup tg = this.createBehaviors(bgRoot);
      tg.addChild(this.createSceneGraph());
      this.addLights(bgRoot);
      u.addBranchGraph(bgRoot);

   }


   public void destroy()
   {
      u.cleanup();
   }


   public static void main(String[] args)
   {
      new MainFrame(new SimpleFigure3Db(), 300, 400);
   }

}
```

Protokoll `SimpleFigure3Db.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
```

```
public void init()
{
   this.setLayout(new BorderLayout());
   GraphicsConfiguration config =
         SimpleUniverse.getPreferredConfiguration();
   Canvas3D c = new Canvas3D(config);
   this.add("Center", c);
   u = new SimpleUniverse(c);
   u.getViewingPlatform().
         setNominalViewingTransform();
   u.addBranchGraph(this.createBackground());

   BranchGroup bgRoot = new BranchGroup();
   TransformGroup tg = this.createBehaviors(bgRoot);
   tg.addChild(this.createSceneGraph());
   this.addLights(bgRoot);
   u.addBranchGraph(bgRoot);

}
```

Abbildung 4.13: Hintergrundbild: `code.jpg`

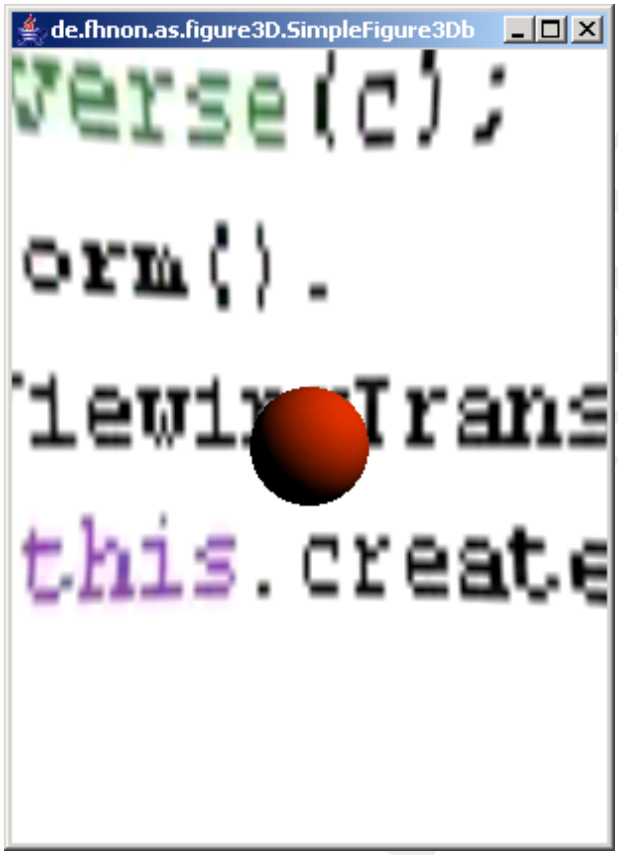
```
  (build 1.4.2-b28, mixed mode)

D:\bonin\artsprog\code>javac de/fhnon/as/figure3D/SimpleFigure3Db.java

D:\bonin\artsprog\code>java de.fhnon.as.figure3D.SimpleFigure3Db

D:\bonin\artsprog\code>
```

Wir ändern die Bilddatei zur Schaffung der Textur um die Auswirkungen auf das Ergebnis zu verdeutlichen. Nun nutzen wir die Datei `code.jpg` (↪ Abbildung 4.13 S. 87). Das Ergebnis zeigt ↪ Abbildung 4.14 S. 88.

Animierte Kugel — 3. Fall

Klasse `SimpleFigure3Dc`

```
/**
 *  "Java 3D Example"
 *  Animierte Kugel mit
 * Zylinder und Kegel erweitert
 *
 *@author     Bonin
 *@version    1.1
 */
```



Legende:
Quellcode ↪ S. 80
Textur ↪ Abbildung 4.13 S. 87

Abbildung 4.14: Beispiel: `SimpleFigure3Db` — Textur `code.jpg`

```
package de.fhnon.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.geometry.Cone;
import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Sphere;

import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.universe.SimpleUniverse;

import com.sun.j3d.utils.applet.MainFrame;

import javax.media.j3d.Alpha;
import javax.media.j3d.Appearance;
import javax.media.j3d.Background;

import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.Material;
import javax.media.j3d.PositionInterpolator;
import javax.media.j3d.TexCoordGeneration;
import javax.media.j3d.Texture;
import javax.media.j3d.TextureAttributes;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;

import javax.vecmath.Color3f;
import javax.vecmath.Color4f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

/**
 *  Description of the Class
 *
 *@author     bonin
```

```
 *@created    14. September 2004
 */
public class SimpleFigure3Dc extends Applet {
        private SimpleUniverse u = null;


        /**
         *  Constructor for the SimpleFigure3Dc object
         */
        private SimpleFigure3Dc() {
                super();
        }


        /**
         *  Description of the Method
         *
         *@return    Description of the Return Value
         */
        public BranchGroup createSceneGraph() {
                BranchGroup bg = new BranchGroup();
                Appearance app = new Appearance();
                Color3f ambientC =
                                new Color3f(0.9f, 0.2f, 1.0f);
                Color3f emissiveC =
                                new Color3f(0.0f, 0.0f, 0.0f);
                Color3f diffuseC =
                                new Color3f(0.9f, 0.2f, 1.0f);
                Color3f specularC =
                                new Color3f(0.0f, 0.0f, 0.0f);
                float shininess = 80.0f;

                app.setMaterial(
                                new Material(
                                ambientC,
                                emissiveC,
                                diffuseC,
                                specularC,
                                shininess));

                Sphere sphere =
                                new Sphere(
```

```
                                0.1f,
                                Primitive.GENERATE_NORMALS,
                                40,
                                app);
                Cylinder cylinder =
                                new Cylinder(
                                0.05f, 0.2f,
                                Primitive.GENERATE_NORMALS,
                                40, 40,
                                app);
                Cone cone =
                                new Cone(
                                0.05f, 0.6f,
                                Primitive.GENERATE_NORMALS,
                                40, 40,
                                app);
                bg.addChild(sphere);
                bg.addChild(cylinder);
                bg.addChild(cone);
                return bg;
        }


        /**
         *  Description of the Method
         *
         *@return    Description of the Return Value
         */
        public BranchGroup createBackground() {
                BranchGroup bg =
                                new BranchGroup();
                Background back = new Background();
                back.setApplicationBounds(
                                getBoundingSphere());
                BranchGroup bgGeometry =
                                new BranchGroup();
                Appearance app = new Appearance();

                Texture tex = new TextureLoader(
                "de/fhnon/as/figure3D/brickwork.jpg",
                                null).getTexture();
                app.setTexture(tex);
```

```
        app.setTexCoordGeneration(
          new TexCoordGeneration(
          TexCoordGeneration.SPHERE_MAP,
          TexCoordGeneration.TEXTURE_COORDINATE_2));
        app.setTextureAttributes(
                        new TextureAttributes(
                        TextureAttributes.REPLACE,
                        new Transform3D(),
                        new Color4f(),
                        TextureAttributes.NICEST));

        Sphere sphere = new Sphere(1.0f,
          Primitive.GENERATE_TEXTURE_COORDS |
          Primitive.GENERATE_NORMALS_INWARD, 40, app);

        bgGeometry.addChild(sphere);
        back.setGeometry(bgGeometry);
        bg.addChild(back);
        return bg;
}


/**
 *  Adds a feature to the Lights
 *  attribute of the SimpleFigure3Dc object
 *
 *@param  bg  The feature to be added
 * to the Lights attribute
 */
public void addLights(BranchGroup bg) {
        DirectionalLight light =
          new DirectionalLight(
          new Color3f(1.0f, 1.0f, 0.0f),
          new Vector3f(-1.0f, -1.0f, -1.0f));
        light.setInfluencingBounds(
                        this.getBoundingSphere());
        bg.addChild(light);
}


/**
```

```
     *  Description of the Method
     *
     *@param  bg  Description of the Parameter
     *@return     Description of the Return Value
     */
    public TransformGroup createBehaviors(
                    BranchGroup bg) {
            TransformGroup objTrans =
                            new TransformGroup();

            objTrans.setCapability(
              TransformGroup.ALLOW_TRANSFORM_WRITE);
            Transform3D xAxis = new Transform3D();
            xAxis.rotY(Math.PI / 2);
            xAxis.rotZ(Math.PI / 8);
            Alpha xAlpha = new Alpha(-1,
                            Alpha.DECREASING_ENABLE |
                            Alpha.INCREASING_ENABLE,
                            1000, 1000, 5000,
                            1000, 1000, 10000,
                            2000, 4000);
            PositionInterpolator posInt =
              new PositionInterpolator(xAlpha,
                            objTrans,
                            xAxis, -0.8f, 0.8f);
            posInt.setSchedulingBounds(
                            getBoundingSphere());
            objTrans.addChild(posInt);

            bg.addChild(objTrans);
            return objTrans;
    }


     /**
     *  Gets the boundingSphere
     *  attribute of the SimpleFigure3Dc object
     *
     *@return    The boundingSphere value
     */
    BoundingSphere getBoundingSphere() {
            return new BoundingSphere(
```

```
                    new Point3d(0.0, 0.0, 0.0), 200.0);
}


/**
 *  Description of the Method
 */
public void init() {
        this.setLayout(new BorderLayout());
        GraphicsConfiguration config =
          SimpleUniverse.getPreferredConfiguration();
        Canvas3D c = new Canvas3D(config);
        this.add("Center", c);
        u = new SimpleUniverse(c);
        u.getViewingPlatform().
                        setNominalViewingTransform();
        u.addBranchGraph(this.createBackground());

        BranchGroup bgRoot = new BranchGroup();
        TransformGroup tg = this.createBehaviors(bgRoot);
        tg.addChild(this.createSceneGraph());
        this.addLights(bgRoot);
        u.addBranchGraph(bgRoot);

}


/**
 *  Description of the Method
 */
public void destroy() {
        u.cleanup();
}


/**
 *  The main program for the SimpleFigure3Dc class
 *
 *@param  args  The command line arguments
 */
public static void main(String[] args) {
        new MainFrame(new SimpleFigure3Dc(), 1000, 700);
```

```
        }
}
```

Protokoll `SimpleFigure3Dc.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/figure3D/SimpleFigure3Dc.java

D:\bonin\artsprog\code>java
  de.fhnon.as.figure3D.SimpleFigure3Dc

D:\bonin\artsprog\code>
```

Legende:
Quellcode ↪ S. 87

Abbildung 4.15: Beispiel: `SimpleFigure3Dc`

4.3.4 Durchsichtiger Zylinder

Transparency

Die Transparenz von Objekten ist für die 3D-Maschine eine rechenintensive Aufgabe. Um die Transparenzfrage zu entscheiden, bedarf es der Festlegung, welche Objekte durch ein Objekt hindurchscheinen sollen und wie deren Farbe sich als resultierende Farbe der Objekte errechnet. Ein Objekt, welches hinter einem durchsichtigen Objekt liegt, soll vom durchsichtigen Objekt in seinem Erscheinungsbild beeinflusst werden. Ein undurchsichtiges Objekt, welches vor einem durchsichtigen Objekt liegt, soll unbeeinflusst dargestellt werden. Es gilt daher, die Reihenfolge der Objekte in der räumlichen Tiefe zu bestimmen. Diese Reihenfolge nennt man die *Z-Order*. Sie ergibt sich aus der Position eines Objektes auf der z-Achse der 3D-Welt.

Wir spezifizieren die Durchsichtigkeit des Zylinders wie folgt:

Detail `TransparencyAttributes`

```
cylinderApp.setTransparencyAttributes(
      new TransparencyAttributes(
      TransparencyAttributes.NICEST,
      0.6f));
```

Der erste Parameterwert des Konstruktors, hier `TransparencyAttributes.NICEST`, definiert die Qualität des Ergebnisses. Um die Berechnungszeit zu verkürzen, wäre der Wert `TransparencyAttributes.FASTEST` zu wählen. Der zweite Parameterwert legt den Grad der Durchsichtigkeit des Objektes fest. Dabei steht der Wert $1.0f$ für ein völlig durchsichtiges Objekt, was faktisch selbst nicht sichtbar ist. Der Wert $0.0f$ spezifiziert ein völlig undurchsichtiges Objekt.

Abbildung 4.16: Beispiel: `Transparency`

Legende:
Quellcode ↪ S. 99

Klasse `Transparency`

```
/**
 *  "Java 3D Example" Durchsichtiges Objekt
 *
 *@author     Bonin
 *@version    1.0
 */

package de.fhnon.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Sphere;
import com.sun.j3d.utils.universe.SimpleUniverse;

import com.sun.j3d.utils.applet.MainFrame;

import javax.media.j3d.AmbientLight;
import javax.media.j3d.Appearance;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.Material;
import javax.media.j3d.PolygonAttributes;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.TransparencyAttributes;

import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

public class Transparency extends Applet
{
   private SimpleUniverse u = null;
```

```
private Transparency()
{
   super();
}


public BranchGroup createSceneGraph()
{
   BranchGroup bg = new BranchGroup();
   TransformGroup sphereTG = new TransformGroup();
   Transform3D sphereT3D = new Transform3D();

   sphereT3D.setTranslation(new Vector3f(1f, 0f, -1.5f));
   sphereTG.setTransform(sphereT3D);

   Appearance sphereApp = new Appearance();
   Appearance cylinderApp = new Appearance();

   sphereApp.setMaterial(new Material(
         new Color3f(0.1f, 0.1f, 0.1f),
         new Color3f(0.0f, 0.0f, 0.0f),
         new Color3f(0.8f, 0.8f, 0.8f),
         new Color3f(0.6f, 0.6f, 0.6f),
         100f));
   sphereApp.setPolygonAttributes(new PolygonAttributes(
         PolygonAttributes.POLYGON_LINE,
         PolygonAttributes.CULL_NONE,
         0));
   Sphere sphere =
         new Sphere(
         0.5f, Sphere.GENERATE_NORMALS, 40, sphereApp);
   bg.addChild(sphere);

   cylinderApp.setMaterial(new Material(
         new Color3f(0.0f, 0.0f, 1.0f),
         new Color3f(0.0f, 0.0f, 0.0f),
         new Color3f(1.0f, 0.0f, 0.0f),
         new Color3f(1.0f, 1.0f, 1.0f),
         100f));
   cylinderApp.setTransparencyAttributes(
         new TransparencyAttributes(
         TransparencyAttributes.NICEST,
```

```
            0.6f));
      Cylinder cylinder =
            new Cylinder(
            0.4f, 1f,
            Cylinder.GENERATE_NORMALS,
            40, 1,
            cylinderApp);
      bg.addChild(cylinder);

      bg.compile();
      return bg;
   }


   public void addLights(BranchGroup bg)
   {
      AmbientLight aLight = new AmbientLight(
            new Color3f(1f, 1f, 1f));

      DirectionalLight dLight =
            new DirectionalLight(
            new Color3f(1.0f, 1.0f, 0.0f),
            new Vector3f(-0.5f, -0.5f, -1.0f));

      aLight.setInfluencingBounds(
            this.getBoundingSphere());
      dLight.setInfluencingBounds(
            this.getBoundingSphere());

      bg.addChild(aLight);
      bg.addChild(dLight);
   }


   public TransformGroup createBehaviors(
         BranchGroup bg)
   {
      TransformGroup objTrans =
            new TransformGroup();
      bg.addChild(objTrans);
      return objTrans;
   }
```

```
   BoundingSphere getBoundingSphere()
   {
      return new BoundingSphere(
            new Point3d(0.0, 0.0, 0.0), 100000.0);
   }



   public void init()
   {
      this.setLayout(new BorderLayout());
      GraphicsConfiguration config =
            SimpleUniverse.getPreferredConfiguration();
      Canvas3D c = new Canvas3D(config);
      this.add("Center", c);
      u = new SimpleUniverse(c);
      u.getViewingPlatform().
            setNominalViewingTransform();

      BranchGroup bgRoot = new BranchGroup();
      TransformGroup tg = this.createBehaviors(bgRoot);
      tg.addChild(this.createSceneGraph());
      this.addLights(bgRoot);
      u.addBranchGraph(bgRoot);

   }


   public void destroy()
   {
      u.cleanup();
   }


   public static void main(String[] args)
   {
      new MainFrame(new Transparency(), 1000, 700);
   }

}
```

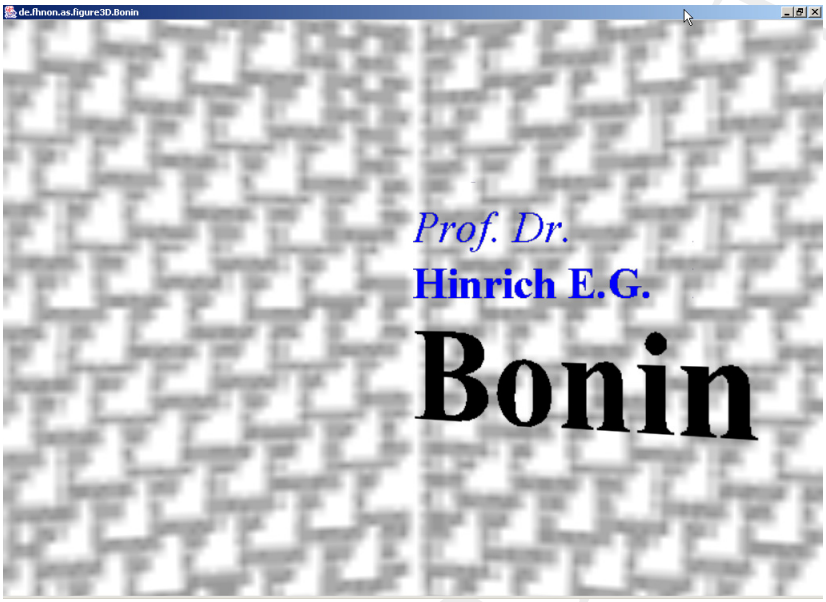
Protokoll `Transparency.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/figure3D/Transparency.java

D:\bonin\artsprog\code>java
  de.fhnon.as.figure3D.Transparency

D:\bonin\artsprog\code>
```

4.3.5 Drehender Text

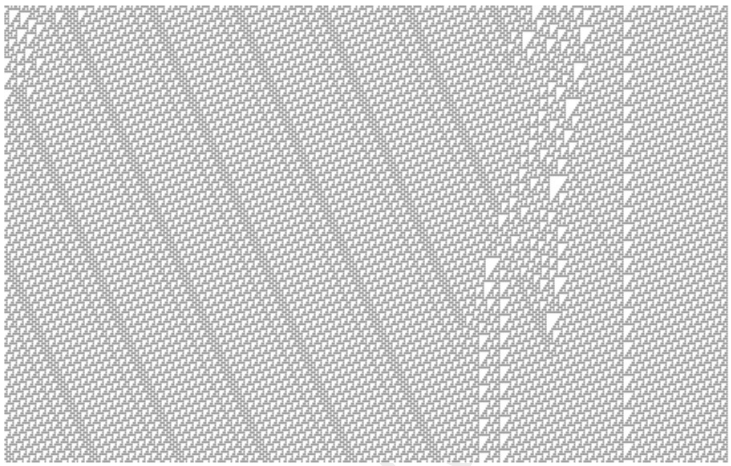
Die Beispielsidee ist der Java3D-Distribution entnommen. Der Quellcode ist modifiziert und ergänzt. Mit Drücken der Taste ↑ wird der Text größer. Die Taste ↓ verkleinert ihn. Mit der Taste ← wird der Text nach links verschoben. Die Taste → verschiebt ihn nach rechts.

```
/**
 *  Beispiel "Drehender Text"
 *  Idee aus Sun Demo-Sammlung entnommen; siehe:
 *  \j2sdk1.4.2\demo\java3d\Text2D\Text2DTest.java
 *  Copyright (c) 1996-2002 Sun Microsystems.
 *  Quellcode modified
 *
 *@author     Bonin
 *@version    1.0
 */
package de.fhnon.as.figure3D;
import java.applet.Applet;
import java.awt.GraphicsConfiguration;
import java.awt.BorderLayout;
import java.awt.Font;
```



Legende:
Quellcode ↪ S. 103

Abbildung 4.17: Beispiel: Drehender Text



Legende:
Quelle zur Erzeugung des Hintergrundbildes ↪ [Bonin04a].

Abbildung 4.18: Hintergrundbild: `AutomatonR110.eps`

```
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.Text2D;
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Sphere;
import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.universe.SimpleUniverse;

import javax.media.j3d.Alpha;
import javax.media.j3d.Appearance;
import javax.media.j3d.Background;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.PolygonAttributes;
import javax.media.j3d.RotationInterpolator;
import javax.media.j3d.Shape3D;
import javax.media.j3d.Texture;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;

import javax.vecmath.Point3d;
```

```
import javax.vecmath.Color3f;
import javax.vecmath.Vector3f;

public class Bonin extends Applet
{

   private SimpleUniverse u = null;


   public BranchGroup createSceneGraph()
   {

      BranchGroup objRoot = new BranchGroup();

      TransformGroup objScale = new TransformGroup();
      Transform3D t3d = new Transform3D();
      t3d.setScale(0.4);
      objScale.setTransform(t3d);
      objRoot.addChild(objScale);

      TransformGroup objTrans = new TransformGroup();
      objTrans.setCapability(
            TransformGroup.ALLOW_TRANSFORM_WRITE);

      BoundingSphere bounds =
            new BoundingSphere(
            new Point3d(0.0, 0.0, 0.0), 100.0);

      TransformGroup textTranslationGroup;
      Transform3D textTranslation;
      float yPos = -0.5f;
      Shape3D textObject = new Text2D("Bonin",
            new Color3f(0f, 0f, 0f),
            "Serif",
            120,
            Font.BOLD);

      Appearance app = textObject.getAppearance();

      PolygonAttributes pa = app.getPolygonAttributes();
      if (pa == null)
      {
```

```
   pa = new PolygonAttributes();
}
pa.setCullFace(PolygonAttributes.CULL_NONE);
if (app.getPolygonAttributes() == null)
{
   app.setPolygonAttributes(pa);
}
objTrans.addChild(textObject);

textTranslation = new Transform3D();
textTranslation.setTranslation(
      new Vector3f(0f, yPos, 0f));
textTranslationGroup = new TransformGroup(
      textTranslation);
textTranslationGroup.addChild(objTrans);
objScale.addChild(textTranslationGroup);
yPos += 0.5f;

textObject = new Text2D("Hinrich E.G.",
      new Color3f(0f, 0f, 1f),
      "Serif",
      40,
      Font.BOLD);
textTranslation = new Transform3D();
textTranslation.setTranslation(
      new Vector3f(0f, yPos, 0f));
textTranslationGroup = new TransformGroup(
      textTranslation);
textTranslationGroup.addChild(textObject);
objScale.addChild(textTranslationGroup);
yPos += 0.2f;

textObject = new Text2D("Prof. Dr.",
      new Color3f(0f, 0f, 1f),
      "Serif",
      40,
      Font.ITALIC);
textTranslation = new Transform3D();
textTranslation.setTranslation(
      new Vector3f(0f, yPos, 0f));
textTranslationGroup =
      new TransformGroup(textTranslation);
```

```
      textTranslationGroup.addChild(textObject);
      objScale.addChild(textTranslationGroup);
      yPos += 0.5f;

      Transform3D yAxis = new Transform3D();
      Alpha rotationAlpha =
            new Alpha(-1,
            Alpha.INCREASING_ENABLE,
            0, 0,
            4000, 0, 0,
            0, 0, 0);

      RotationInterpolator rotator =
            new RotationInterpolator(
            rotationAlpha, objTrans, yAxis,
            0.0f, (float) Math.PI * 2.0f);
      rotator.setSchedulingBounds(bounds);
      objTrans.addChild(rotator);

      return objRoot;
   }


   public BranchGroup createBackground()
   {
      BranchGroup backgroundGroup =
            new BranchGroup();
      Background back = new Background();
      back.setApplicationBounds(
            getBoundingSphere());
      BranchGroup bgGeometry =
            new BranchGroup();
      Appearance app = new Appearance();
      Texture tex = new TextureLoader(
            "./de/fhnon/as/figure3D/AutomatonR110.jpg",
            this).getTexture();
      app.setTexture(tex);
      Sphere sphere = new Sphere(1.0f,
            Primitive.GENERATE_TEXTURE_COORDS |
            Primitive.GENERATE_NORMALS_INWARD, app);
      bgGeometry.addChild(sphere);
      back.setGeometry(bgGeometry);
```

```
      backgroundGroup.addChild(back);
      return backgroundGroup;
   }


   BoundingSphere getBoundingSphere()
   {
      return new BoundingSphere(
            new Point3d(0.0, 0.0, 0.0), 200.0);
   }


   public Bonin() { }


   public void init()
   {
      setLayout(new BorderLayout());
      GraphicsConfiguration config =

      SimpleUniverse.getPreferredConfiguration();

      Canvas3D c = new Canvas3D(config);
      add("Center", c);

      BranchGroup scene = createSceneGraph();
      u = new SimpleUniverse(c);
      MoverBehavior navigator =
            new MoverBehavior(
            u.getViewingPlatform().
            getViewPlatformTransform());
      scene.addChild(navigator);

      scene.compile();

      u.getViewingPlatform().
            setNominalViewingTransform();
      u.addBranchGraph(createBackground());
      u.addBranchGraph(scene);
   }
```

```
   public void destroy()
   {
      u.cleanup();
   }



   public static void main(String[] args)
   {
      new MainFrame(new Bonin(), 256, 256);
   }
}
```

Klasse `MoverBehavior`

```
/**
 *  Beispiel "Drehender Text" Idee aus Sun
 *  Demo-Sammlung entnommen, siehe
 *  \j2sdk1.4.2\demo\java3d\Text2D\MoverBehavior.java
 *  Copyright (c) 1996-2002 Sun Microsystems,
 *
 *@author     Bonin
 *@version    1.0
 */
package de.fhnon.as.figure3D;

import java.awt.event.KeyEvent;
import java.awt.AWTEvent;
import javax.media.j3d.*;
import java.util.Enumeration;
import javax.vecmath.*;

/*
 *  Mover behavior class -
 *  used to allow viewer to move using arrow keys
 */
class MoverBehavior extends Behavior
{
   WakeupOnAWTEvent w1 =
       new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED);
   WakeupCriterion[] w2 = {w1};
   WakeupCondition w = new WakeupOr(w2);
```

```
TransformGroup viewTransformGroup;
/*
 *  holds current rotation radians
 */
double rotation = 0.0;


public void initialize()
{
   /*
    *  Establish initial wakeup criteria
    */
   wakeupOn(w);
}


/**
 *  Override Behavior's stimulus method to
 *  handle the event.
 *
 *@param  criteria  Description of the
 *      Parameter
 */
public void processStimulus(Enumeration criteria)
{
   WakeupOnAWTEvent ev;
   WakeupCriterion genericEvt;
   AWTEvent[] events;

   while (criteria.hasMoreElements())
   {
      genericEvt = (WakeupCriterion)
          criteria.nextElement();
      if (genericEvt instanceof WakeupOnAWTEvent)
      {
         ev = (WakeupOnAWTEvent) genericEvt;
         events = ev.getAWTEvent();
         processManualEvent(events);
      }
   }
   /*
    *  Set wakeup criteria for next time
```

```
     */
    wakeupOn(w);
}


/**
 *  Process a keyboard event to move or rotate
 *  the viewer.
 *
 *@param  events  Description of the Parameter
 */
void processManualEvent(AWTEvent[] events)
{

   for (int i = 0; i < events.length; ++i)
   {
      if (events[i] instanceof KeyEvent)
      {
         KeyEvent event = (KeyEvent) events[i];
         if (event.getKeyCode()
               == KeyEvent.VK_EQUALS)
         {
            continue;
         }
         Transform3D t = new Transform3D();
         viewTransformGroup.getTransform(t);
         Vector3f viewDir =
               new Vector3f(0f, 0f, -1f);
         Vector3f translation = new Vector3f();
         t.get(translation);
         t.transform(viewDir);
         if (event.getKeyCode()
               == KeyEvent.VK_UP)
         {
            translation.x += viewDir.x;
            translation.y += viewDir.y;
            translation.z += viewDir.z;
         } else if (event.getKeyCode()
               == KeyEvent.VK_DOWN)
         {
            translation.x -= viewDir.x;
            translation.y -= viewDir.y;
```

```
                translation.z -= viewDir.z;
            } else if (event.getKeyCode()
                    == KeyEvent.VK_RIGHT)
            {
                rotation += -.1;
            } else if (event.getKeyCode()
                    == KeyEvent.VK_LEFT)
            {
                rotation += .1;
            }
            t.rotY(rotation);
            t.setTranslation(translation);
            viewTransformGroup.setTransform(t);
        }
    }
}


/**
 *  Constructor
 *
 *@param  trans  Description of the Parameter
 */
public MoverBehavior(TransformGroup trans)
{
    viewTransformGroup = trans;
    Bounds bound =
          new BoundingSphere(
          new Point3d(0.0, 0.0, 0.0), 10000.0);
    this.setSchedulingBounds(bound);
}
}
```

Protokoll `Bonin.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)
```

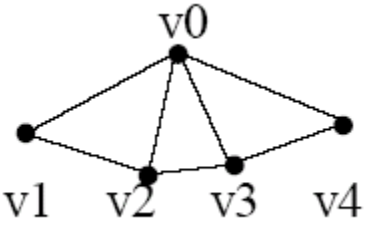


Abbildung 4.19: TriangleFanArray — *Vertices* $v_{0..4}$

```
D:\bonin\artsprog\code>javac de/fhnon/as/figure3D/*.java

D:\bonin\artsprog\code>java de.fhnon.as.figure3D.Bonin

D:\bonin\artsprog\code>
```

4.3.6 Konstruierte Geometrie

Wir konstruieren unser 3D-Objekt, eine Yo-Yo-Spindel, als Instanz der Klasse `javax.media.j3d.Shape3D`. Diese Beispielidee „Yo-Yo“ wurde dem Java3D-Tutorial[9] entnommen. Seine spezifizierte Geometrie übergeben wir als Parameter des Konstruktors. Diese Geometrie spezifizieren wir auf der Basis der Klasse `javax.media.j3d.TriangleFanArray`. Diese hat folgenden Konstruktor:

TriangleFanArray

```
public TriangleFanArray(int vertexCount,
                        int vertexFormat,
                        int[] stripVertexCounts)
```

Ein TriangleFanArray-Skizze mit $v_{0..4}$ zeigt Abbildung 4.19 S. 114. Wir konstruieren unser Objekt aus vier *triangle fan strips* mit einer „Auflösung“ (N) von 30 *Vertices* ($v_{1..30}$). Deren Nullpunkte (v_0) setzen wir direkt ausgehend vom Koordinatennullpunkt und einer Verschiebung auf der z-Achse, wie folgt:

[9] Tutorial v1.6 (Java 3D API v1.2) — Chapter 2 (Creating Geometry) ↪ `http://java.sun.com/products/java-media/3D/learning/-tutorial/index.html` (online 29-Apr-2004)

```
coords[0 * (N + 1)] = new Point3f(0.0f, 0.0f, w);
coords[1 * (N + 1)] = new Point3f(0.0f, 0.0f, 0.0f);
coords[2 * (N + 1)] = new Point3f(0.0f, 0.0f, 0.0f);
coords[3 * (N + 1)] = new Point3f(0.0f, 0.0f, -w);
```

Die Koordinaten für $v_{1..30}$ ermitteln wir für jeden der vier *triangle fan strips* auf die gleiche Weise. Die jeweiligen x,y-Werte nach folgender Formel und den z-Wert setzen wir dann direkt auf w oder $-w$.

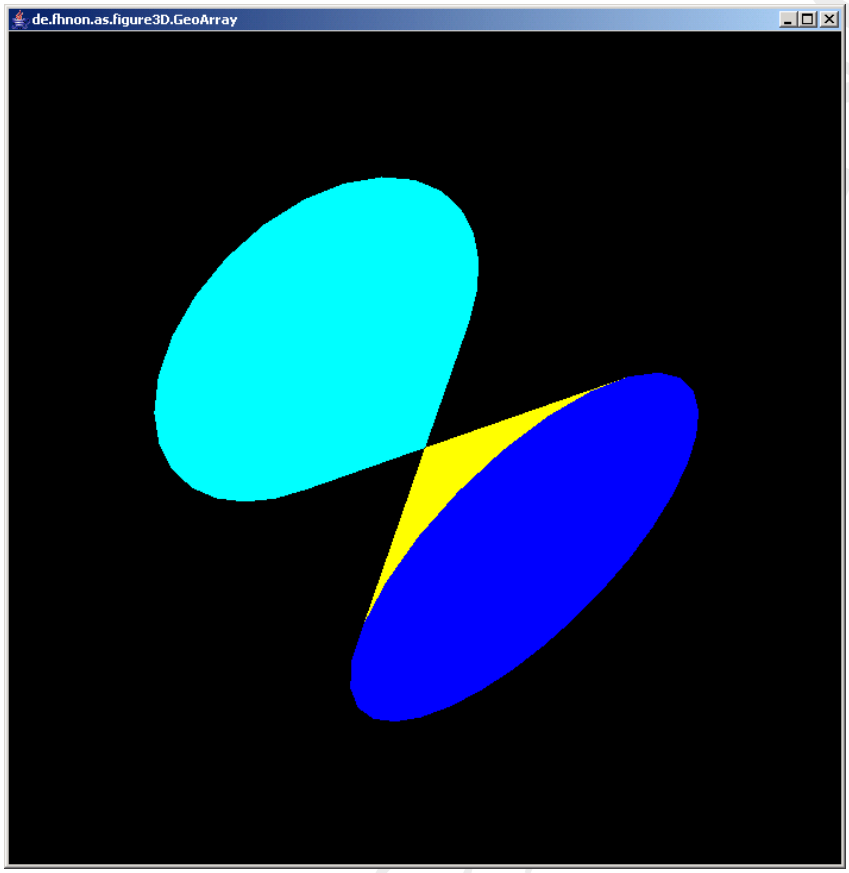
```
for (a = 0, n = 0; n < N;
     a = 2.0 * Math.PI / (N - 1) * ++n)
  {
     x = (float) (r * Math.cos(a));
     y = (float) (r * Math.sin(a));
...
}
```

Die Koordinaten legen wir hintereinander im Array `coords` ab. Mit dem Wert `stripVertexCounts`, ein `int`-Array, geben wir die Anzahl der *Vertices* für jeden der *triangle fan strips* an. Seine Länge entspricht der Anzahl der *strips*.

Um die vier *triangle fan strips* einfach zu erkennen, ordnen wir ihnen jeweils eine Farbe zu.

Klasse `GeoArray`

```
/**
 *  "Java 3D Example" Geometry Array
 *
 *@author     Bonin
 *@version    1.0
 */

package de.fhnon.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.geometry.Primitive;
```



Legende:
Quellcode ↪ S. 115

Abbildung 4.20: Beispiel: Konstruierte Geometrie — YoYo

```
import com.sun.j3d.utils.geometry.Sphere;
import com.sun.j3d.utils.universe.SimpleUniverse;

import com.sun.j3d.utils.applet.MainFrame;

import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.Geometry;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.TriangleFanArray;
import javax.media.j3d.Shape3D;

import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Point3f;
import javax.vecmath.Vector3f;

public class GeoArray extends Applet
{
   private SimpleUniverse u = null;


   private GeoArray()
   {
      super();
   }


   public BranchGroup createSceneGraph()
   {
      BranchGroup bg = new BranchGroup();
      TransformGroup tg = new TransformGroup();
      Transform3D t3d = new Transform3D();
      t3d.setRotation(
            new AxisAngle4f(
            1f,
            1f,
            0f,
```

```
            (float) Math.toRadians(60)));
      tg.setTransform(t3d);

      Shape3D shape3D =
            new Shape3D(this.yoyoGeometry());

      tg.addChild(shape3D);
      bg.addChild(tg);

      return bg;
   }


   public void addLights(BranchGroup bg)
   {
      DirectionalLight light =
            new DirectionalLight(
            new Color3f(1.0f, 1.0f, 0.0f),
            new Vector3f(-1.0f, -1.0f, -1.0f));
      light.setInfluencingBounds(
            this.getBoundingSphere());
      bg.addChild(light);
   }


   public TransformGroup createBehaviors(
         BranchGroup bg)
   {
      TransformGroup objTrans =
            new TransformGroup();
      bg.addChild(objTrans);
      return objTrans;
   }


   BoundingSphere getBoundingSphere()
   {
      return new BoundingSphere(
            new Point3d(0.0, 0.0, 0.0), 200.0);
   }
```

```
private Geometry yoyoGeometry()
{
   TriangleFanArray tfan;
   final int N = 30;
   int vertexCount = 4 * (N + 1);
   Point3f coords[] = new Point3f[vertexCount];
   Color3f colors[] = new Color3f[vertexCount];
   Color3f blue = new Color3f(0.0f, 0.0f, 1.0f);
   Color3f yellow = new Color3f(1.0f, 1.0f, 0.0f);
   Color3f cyan = new Color3f(0.0f, 1.0f, 1.0f);
   Color3f magenta = new Color3f(1.0f, 0.0f, 1.0f);

   int stripVertexCounts[] =
         {N + 1, N + 1, N + 1, N + 1};

   float r = 0.5f;
   float w = 0.4f;
   int n;
   double a;
   float x;
   float y;
   /*
    *  set the central points
    *  for the four triangle fan strips
    */
   coords[0 * (N + 1)] =
         new Point3f(0.0f, 0.0f, w);
   coords[1 * (N + 1)] =
         new Point3f(0.0f, 0.0f, 0.0f);
   coords[2 * (N + 1)] =
         new Point3f(0.0f, 0.0f, 0.0f);
   coords[3 * (N + 1)] =
         new Point3f(0.0f, 0.0f, -w);
   colors[0 * (N + 1)] = blue;
   colors[1 * (N + 1)] = yellow;
   colors[2 * (N + 1)] = cyan;
   colors[3 * (N + 1)] = magenta;

   for (a = 0, n = 0;
         n < N;
         a = 2.0 * Math.PI / (N - 1) * ++n)
   {
```

```
            x = (float) (r * Math.cos(a));
            y = (float) (r * Math.sin(a));
            coords[0 * (N + 1) + n + 1] =
                  new Point3f(x, y, w);
            coords[1 * (N + 1) + N - n] =
                  new Point3f(x, y, w);
            coords[2 * (N + 1) + n + 1] =
                  new Point3f(x, y, -w);
            coords[3 * (N + 1) + N - n] =
                  new Point3f(x, y, -w);
            colors[0 * (N + 1) + N - n] = blue;
            colors[1 * (N + 1) + n + 1] = yellow;
            colors[2 * (N + 1) + N - n] = cyan;
            colors[3 * (N + 1) + n + 1] = magenta;
         }
         tfan = new TriangleFanArray(vertexCount,
               TriangleFanArray.COORDINATES |
               TriangleFanArray.COLOR_3,
               stripVertexCounts);
         tfan.setCoordinates(0, coords);
         tfan.setColors(0, colors);
         return tfan;
      }


      public void init()
      {
         this.setLayout(new BorderLayout());
         GraphicsConfiguration config =
               SimpleUniverse.getPreferredConfiguration();
         Canvas3D c = new Canvas3D(config);
         this.add("Center", c);
         u = new SimpleUniverse(c);
         u.getViewingPlatform().
               setNominalViewingTransform();

         BranchGroup bgRoot = new BranchGroup();
         TransformGroup tg = this.createBehaviors(bgRoot);
         tg.addChild(this.createSceneGraph());
         this.addLights(bgRoot);
         u.addBranchGraph(bgRoot);
```

```
    }


    public void destroy()
    {
       u.cleanup();
    }


    public static void main(String[] args)
    {
       new MainFrame(new GeoArray(), 700, 700);
    }

}
```

Protokoll `GeoArray.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac de/fhnon/as/figure3D/GeoArray.java

D:\bonin\artsprog\code>java de.fhnon.as.figure3D.GeoArray

D:\bonin\artsprog\code>
```

4.3.7 `DataSharing`

In diesem Beispiel greifen wir unser Startbeispiel, also die Klasse `HelloUniverse` (↪ Abschnitt 22 S. 66) wieder auf. Allerdings sind jetzt zwei Zylinder darzustellen. Wir wollen jedoch nicht zweimal den Konstruktor `Cylinder(...)` aufrufen, sondern nur unser Zylinderobjekt zweimal darstellen und zwar einmal gedreht und einmal nicht gedreht.

Zu diesem *Data Sharing* nutzen wir die Klassen `SharedGroup` und `Link` des Paketes `javax.media.j3d`. Zwei `Link`-Knoten zeigen beide auf den `SharedGroup`-Knoten, der die eine `Cylinder`-

Instanz aufnimmt. Der folgende Quellcode zeigt, dass die `SharedGroup`-Instanz, hier `sg`, die Verbindung von den beiden `Link`-Instanzen, hier `link1toSg` und `link2toSg`, zum gemeinsamen Objekt, hier `cylinder`, darstellt.

Detail `SharedGroup` **&** `Link`

```
public BranchGroup createSceneGraph()
{
   BranchGroup bg = new BranchGroup();

   TransformGroup tg = new TransformGroup();

   SharedGroup sg = new SharedGroup();
   Link link1toSg = new Link(sg);
   Link link2toSg = new Link(sg);

   Transform3D t3d = new Transform3D();
   t3d.setRotation(...);
   tg.setTransform(t3d);

   ...

   Cylinder cylinder = new Cylinder(...);

   ...

   sg.addChild(cylinder);
   tg.addChild(link1toSg);
   bg.addChild(tg);
   bg.addChild(link2toSg);
   bg.compile();
   return bg;
}
```

Gegenüber dem Startbeispiel wurden weitere Änderungen vorgenommen. So wurde der Zylinder nur noch mit vier Unterteilungen des Grundkreises, also mit $xDivision = 4$, spezifiziert. Das Ergebnis (↪ Abbildung 4.21 S. 124) zeigt daher den Zylinder als Quader.

Auch die Unterteilung des Zylindermantels wurde reduziert und zwar auf $yDivision = 2$. Zusätzlich wurde die Linienbreite vergrößert (auf den Wert `4.0f`) und mit dem Parameter `PolygonAttributes.-`

`CULL_BACK` wurde die Darstellung der hinteren Polygone unterdrückt.

Abbildung 4.21: Beispiel: `DataSharing`

Legende:
Quellcode ↪ S. 125

Klasse `DataSharing`

```
/**
 *  "Java 3D Example" Data Sharing
 *
 *@author     Bonin
 *@version    1.0
 */

package de.fhnon.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.applet.MainFrame;

import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.universe.SimpleUniverse;

import javax.media.j3d.Appearance;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.LineAttributes;
import javax.media.j3d.Link;
import javax.media.j3d.Material;
import javax.media.j3d.SharedGroup;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.Transform3D;
import javax.media.j3d.PolygonAttributes;

import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;

public class DataSharing extends Applet
{
   private SimpleUniverse u = null;


   public BranchGroup createSceneGraph()
   {
      BranchGroup bg = new BranchGroup();
```

```
TransformGroup tg = new TransformGroup();

SharedGroup sg = new SharedGroup();
Link link1toSg = new Link(sg);
Link link2toSg = new Link(sg);

Transform3D t3d = new Transform3D();
t3d.setRotation(
      new AxisAngle4f(
      1f,
      1f,
      0f,
      (float) Math.toRadians(90)));
tg.setTransform(t3d);

Appearance app = new Appearance();
Material m = new Material();
Color3f red = new Color3f(1.0f, 0.0f, 0.0f);
m.setEmissiveColor(red);
m.setLightingEnable(true);
app.setMaterial(m);

PolygonAttributes polyAtt =
      new PolygonAttributes();
polyAtt.setPolygonMode(
      PolygonAttributes.POLYGON_LINE);
polyAtt.setCullFace(
      PolygonAttributes.CULL_BACK);
app.setPolygonAttributes(polyAtt);

LineAttributes latt = new LineAttributes();
latt.setLineWidth(6.0f);
latt.setLineAntialiasingEnable(true);
app.setLineAttributes(latt);

Cylinder cylinder =
      new Cylinder(
      0.3f,
      0.9f,
      Primitive.GENERATE_NORMALS,
      4,
```

```
                2,
                app);

        sg.addChild(cylinder);
        tg.addChild(link1toSg);
        bg.addChild(tg);
        bg.addChild(link2toSg);
        bg.compile();
        return bg;
    }


    public void init()
    {
        this.setLayout(new BorderLayout());
        GraphicsConfiguration config =
                SimpleUniverse.getPreferredConfiguration();
        Canvas3D c = new Canvas3D(config);
        this.add("Center", c);
        u = new SimpleUniverse(c);
        u.getViewingPlatform().
                setNominalViewingTransform();
        u.addBranchGraph(
                this.createSceneGraph());

    }


    public void destroy()
    {
        u.cleanup();
    }


    public static void main(String[] args)
    {
        new MainFrame(
                new DataSharing(), 300, 400);
    }

}
```

Protokoll `DataSharing.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac de/fhnon/as/figure3D/DataSharing.java

D:\bonin\artsprog\code>java de.fhnon.as.figure3D.DataSharing

D:\bonin\artsprog\code>
```

4.3.8 `ExampleWavefrontLoad`

Um 3D-Daten in die Java3D-Welt importieren zu können, gibt es das Paket `com.sun.j3d.loaders`. Mit dessen Hilfe läßt sich eine eigene *File Loader Class* entwickeln, die das jeweilige Datenformat importieren kann. Für eine große Menge marktüblicher Datenformate exisitieren schon entsprechende Dateilader.

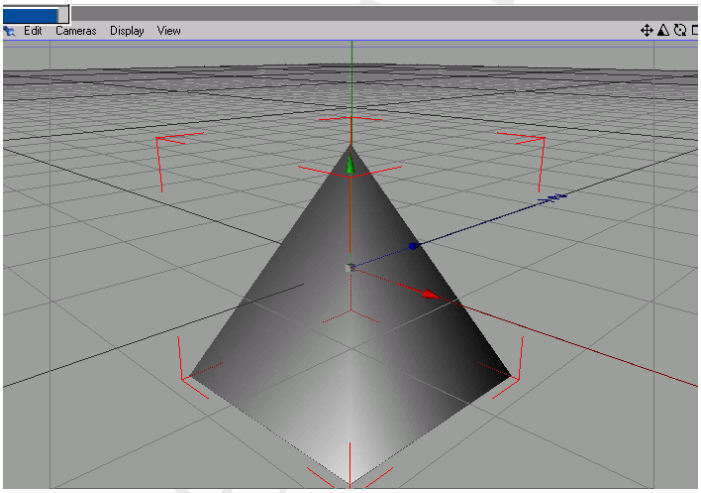
Wavefront

In diesem Beispiel importieren wir ein Objekt, gespeichert im *Wavefront*-Format. Eine solche Objektdatei ist eine ASCII-codierte Datei mit der Namenserweiterung `*.obj`. Prinzipiell kann sie eine Geometrie enthalten, bestehend aus Polygonen und *Free-form Geometry* (Kurven und Oberflächen). Letztere wird vom hier genutzten Lader nicht unterstützt.

Das zu importierende Objekt wurde mit dem Werkzeug *Cinema4D Release 8* der Firma *Maxon* Computer GmbH[10] erzeugt und dort mit der Option „Export Wavefront“ gespeichert.

Der eigenliche Vorgang des Ladens geschieht durch eine Instanz der entsprechenden *File Loader Class*, hier der Klasse `com.sun.j3d.loaders.objectfile.ObjectFile`. Deren Methode `load(`filename`)` gibt eine Instanz der Klasse `com.sun.j3d.loaders.Scene` zurück, falls nicht Ausnahmefälle beim Lesen und Auswerten (Parsen) eintreten. Das folgende Quellcodefragment zeigt die Konstruktion für den Ladevorgang:

[10] Homepage ↪ `http://www.maxon.net` (online 8-May-2004)



Legende:
Quellcode ↪ S. 130

Abbildung 4.22: Beispiel: `ExampleWavefrontLoad` — Objekt in Cinema4D

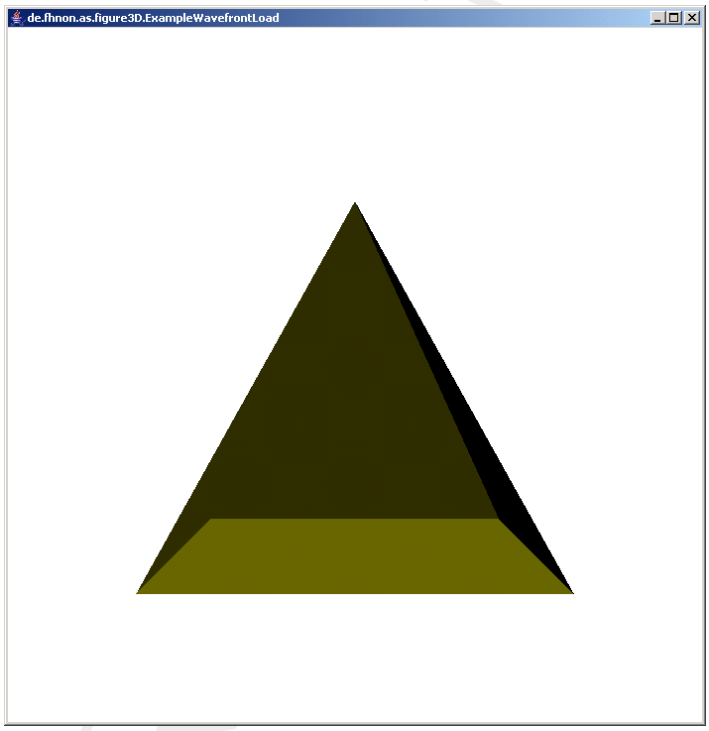
Detail `obj`**-Datei laden**

```
public BranchGroup createSceneGraph()
{
   BranchGroup bg = new BranchGroup();

   ObjectFile f = new ObjectFile();

   Scene s = null;

   try
   {
      s = f.load(filename);
   } catch (FileNotFoundException e)
       {...;
   } catch (ParsingErrorException e)
       {...;
   } catch (IncorrectFormatException e)
       { ...;
   }

   bg.addChild(s.getSceneGroup());

   return bg;
   }
```

Um das erfolgreiche Importieren zu verdeutlichen, wurde das Objekt in seiner Größe mit dem Konstrukt `t3d.setScale(0.005)` reduziert. Das Objekt im Werkzeug *Cinema4D Release* zeigt Abbildung 4.22 S. 129. Die ASCII-Datei `pyramide.obj` (↪ S. 135) zeigt es im exportierten Wavefront-Format. Das Importergebnis zeigt Abbildung 4.23 S. 131. Hinweis: Man beachte die Unterschiede in der Form und dem Erscheinungsbild.

Klasse `ExampleWavefrontLoad`

```
/**
 *  "Java 3D Example" Laden eines Objektes im Format
 *  Wavefront. Erzeugt wurde das Objekt mit Cinema4D
 *
 *@author     Bonin
 *@version    1.0
 */
```



Legende:
Quellcode ↪ S. 130

Abbildung 4.23: Beispiel: `ExampleWavefrontLoad` — Objekt in Java3D

```
package de.fhnon.as.figure3D;

import java.io.*;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.loaders.objectfile.ObjectFile;
import com.sun.j3d.loaders.ParsingErrorException;
import com.sun.j3d.loaders.IncorrectFormatException;
import com.sun.j3d.loaders.Scene;

import com.sun.j3d.utils.universe.SimpleUniverse;

import com.sun.j3d.utils.applet.MainFrame;

import javax.media.j3d.Appearance;
import javax.media.j3d.Background;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.Material;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.Transform3D;

import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

public class ExampleWavefrontLoad extends Applet
{
   private SimpleUniverse u = null;

   private String filename =
         "de/fhnon/as/figure3D/pyramide.obj";


   private ExampleWavefrontLoad()
   {
      super();
```

```
}


public BranchGroup createSceneGraph()
{
   BranchGroup bg = new BranchGroup();
   TransformGroup tg = new TransformGroup();
   Transform3D t3d = new Transform3D();
   t3d.setScale(0.005);
   tg.setTransform(t3d);

   ObjectFile f = new ObjectFile();

   Scene s = null;

   try
   {
      s = f.load(filename);
   } catch (FileNotFoundException e)
   {
      System.err.println(e);
      System.exit(1);
   } catch (ParsingErrorException e)
   {
      System.err.println(e);
      System.exit(1);
   } catch (IncorrectFormatException e)
   {
      System.err.println(e);
      System.exit(1);
   }

   System.out.println(s.getNamedObjects());

   tg.addChild(s.getSceneGroup());
   bg.addChild(tg);
   return bg;
}


public void addLights(BranchGroup bg)
{
```

```
      DirectionalLight light =
            new DirectionalLight(
            new Color3f(1.0f, 1.0f, 0.0f),
            new Vector3f(-1.0f, -1.0f, -1.0f));
      light.setInfluencingBounds(
            this.getBoundingSphere());
      bg.addChild(light);
   }


   public TransformGroup createBehaviors(
         BranchGroup bg)
   {
      TransformGroup objTrans =
            new TransformGroup();
      bg.addChild(objTrans);
      return objTrans;
   }


   BoundingSphere getBoundingSphere()
   {
      return new BoundingSphere(
            new Point3d(0.0, 0.0, 0.0), 200.0);
   }


   BranchGroup createBackground()
   {
      BranchGroup bg = new BranchGroup();
      Background background = new Background();
      background.setColor(1.0f, 1.0f, 1.0f);
      background.setApplicationBounds(
            getBoundingSphere());
      bg.addChild(background);
      return bg;
   }


   public void init()
   {
      this.setLayout(new BorderLayout());
```

```
      GraphicsConfiguration config =
            SimpleUniverse.getPreferredConfiguration();
      Canvas3D c = new Canvas3D(config);
      this.add("Center", c);
      u = new SimpleUniverse(c);
      u.getViewingPlatform().
            setNominalViewingTransform();

      BranchGroup bgRoot = new BranchGroup();
      TransformGroup tg = this.createBehaviors(bgRoot);
      tg.addChild(this.createSceneGraph());
      this.addLights(bgRoot);
      u.addBranchGraph(this.createBackground());
      u.addBranchGraph(bgRoot);

   }


   public void destroy()
   {
      u.cleanup();
   }


   public static void main(String[] args)
   {
      new MainFrame(new ExampleWavefrontLoad(), 700, 700);
   }

}
```

Wavefront-**Daten** pyramide.obj

```
# WaveFront *.obj file (generated by Cinema4D)

g Pyramid
v -100 -100 100
v 100 -100 100
v 100 -100 -100
v -100 -100 -100
v 0 100 0
```

```
vt 0 0 0
vt 1 0 0
vt 0 1 0
vt 1 0 0
vt 0 0 0
vt 1 1 0
vt 0 0 0
vt 1 0 0
vt 0 0 0
vt 1 0 0
vt 0.5 1 0

f 1/3 2/6 3/8 4/9
f 5/11 3/8 2/5
f 5/11 4/10 3/7
f 5/11 1/2 4/9
f 5/11 2/4 1/1
```

`*.obj`

In einer `*.obj`-Datei bedeutet:

- `v` float float float
  Eine Angabe einer Eckenposition im Raum (*vertex's geometric position*). Die erste Nennung in der Datei hat den Index 1, die folgendene werden der Reihe nach durchnummeriert.
- `vn` float float float
  Eine Angabe einer Normalen mit einer Indexierung analog zu `v`.
- `vt` float float
  Eine Angabe einer Textur-Koordinate mit einer Indexierung analog zu `v`.
- `f` int int int . . . oder
  `f` int/int int/int int/int . . . oder
  `f` int/int/int int/int/int int/int/int . . .
  Eine Angabe einer Fläche, die durch ein Polygon spezifiziert wird (*Polygonal Face*). Die Zahlen sind Indizes im Feld der *Vertex*-Positionen, Textur-Koordinaten und der Normalen. Die Anzahl der *Vertices* für ein Polygon ist nicht begrenzt. Hinweis: Wird beim `ObjectFile`-Konstruktor die Angabe `ObjectFile.TRIANGULATE` genutzt, dann wird jede Fläche vom Java3D-`Triangulator` verarbeitet. Näheres zur Klasse `Triangulator`
  ↪ Abschnitt 6.1 S. 166.

- g name
  Faces, die nach dieser benannten Gruppe vorkommen, werden als ein separates Shape3D-Objekt konstruiert. Dieses Objekt wird verknüpft mit der SceneGroup des Elternknotens. Mit der Methode getNamedObjects() erhält man diesen Gruppennamen.

Protokoll ExampleWavefrontLoad.log

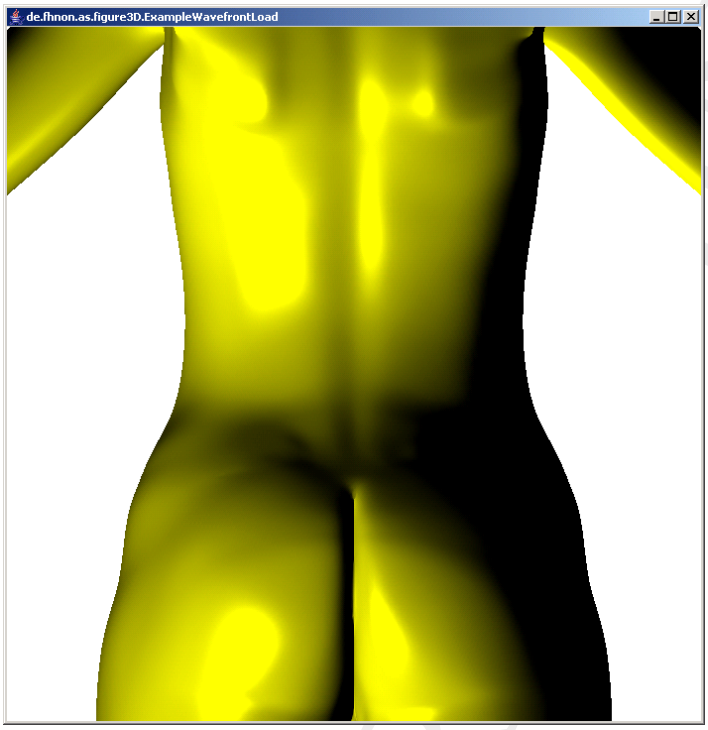
```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/figure3D/ExampleWavefrontLoad.java

D:\bonin\artsprog\code>java
  de.fhnon.as.figure3D.ExampleWavefrontLoad
{pyramid=javax.media.j3d.Shape3D@4ac00c}

D:\bonin\artsprog\code>
```

Wenn man komplexe Objekte, zum Beispiel aus der *Cinema4D*-Bibliothek (hier das Objekt *Human Meg*), im Wavefront-Format importieren will, dann reicht der Speicherplatz, den die JVM (*Java Virtual Maschine*) beim Standardaufruf reserviert, nicht aus. Um die Applikation ausführen zu können, werden die *HotSpot Memory Options* genutzt. Die Parameter -Xms und -Xmx stellen die *Heap Allocation* auf die benötigten Werte ein. Im Beispiel werden 512 MB allokiert:

Memory Options

```
>java -Xms512m -Xmx512m
  de.uni-lueneburg.as.figure3D.ExampleWavefrontLoad
```

-XmsSize setzt den initialen Java^{TM}-Heap-Wert und -XmxSize den maximalen Java^{TM}-Heap-Wert beim Start der Anwendung. Ein Beispiel mit diesem Aufruf zeigt Abbildung 4.24 S. 138.

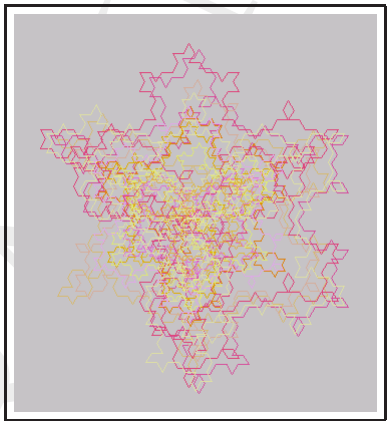


Legende:
Objekt *Human Meg* aus Cinema4D-Bibliothek.
Quellcode ↪ S. 130

Abbildung 4.24: Beispiel: `ExampleWavefrontLoad` — Komplexes Objekt

Kapitel 5

Objekt-Verhalten — `Behavior`



Objekte können sich für den Beobachter im *Virtual Universe* im Laufe der Zeit ändern, beispielsweise können sie sich bewegen. Eine Verhaltensänderung des Objektes kann durch den Benutzer angestoßen werden oder aufgrund einer Kollision mit einem anderen Objekt erfolgen.

Zusätzlich zu Ort und Richtung der Bewegung kann sich die Geometry des Objektes, das heißt, seine Form und seine Farbe ändern. Auch der Beobachter kann seinen Standort und seinen Blickwinkel ändern.

Dieses Kapitel skizziert anhand von einigen Beispielen solche Änderungen des Objekt-Verhaltens.

Trainingsplan

Das Kapitel „Objekt-Verhalten“ gibt einen Überblick über:

- die Möglichkeiten der Anregung und die daraufhin eintretende Aktion,
  ↪ Seite 140 . . .
- die *Custom Behavior Class*,
  ↪ Seite 141 . . .
- beschreibt ein Bespiel zur Modifizierung der *View Platform* per Tastendruck,
  ↪ Seite 150 . . .
- und per Mausklick.
  ↪ Seite 158 . . .

5.1 Anregung und Aktion

Soll ein grafisches Objekt sich im *Virtual Universe* bewegen, dann sind Konstrukte zur Interaktion, Animation und Navigation anzuwenden. Die Tabelle 5.1 S. 141 skizziert diese Begriffe im Java3D-Kontext.

Ein einfaches Beispiel ist eine Box, die sich in Zeitschritten von einer halben Sekunde permanent dreht, wenn vorher eine Taste gedrückt worden ist. Solch ein spezielles Verhalten von Objekten (*Scene-Graph*-Teil) programmieren man auf der Basis der abstrakten Klasse `Behavior`.

Anregung	**Aktion**			
Änderungs-grund (*Stimulus*)	**Objekt der Änderung**			
	`TransformGroup` (Ausrichtung/Ort)	`Geometry` (Form/Farbe)	*SceneGraph* (Hinzu/Entfernen)	`View` (Ort/Richtung)
Benutzer Taste, Maus	**Interaktion**	Anwendungsfall spezifisch		**Navigation**
Kollision	sichtbares Objekt ändert Ausrichtung / Ort	Erscheinung	verschwindet	Ansichtsänderung
Zeit	**Animation**			
Betrachtungsort	*Billboard*	*Level of Detail*	Anwendungsfall spezifisch	

Legende:
Einordnung der Begriffe: Interaktion, Animation und Navigation
Ähnlich *The Java 3D Tutorial — Chapter 4*
↪http://java.sun.com/products/java-media/3D/collateral/j3d_tutorial_ch4.pdf (online 04-May-2004)

Tabelle 5.1: Objekt-Verhalten: Anregung und Aktion

5.2 *Custom Behavior Class*

Die abstrakten Klasse `Behavior` spezifiziert die beiden folgenden Methoden, die wir in unserer Subklasse überschreiben:

- `void initialize()` **Start**
  Diese Methode wird einmal aufgerufen, wenn das Verhalten aktiviert wird. Sie übernimmt daher das Setzen des auslösenden Ereignisses, genannt *Trigger* und spezifiziert die Zustandsvariablen zu Beginn.
- `void processStimulus(java.util.Enumeration criteria)` **Stimulus**
  Diese Methode wird immer aufgerufen, wenn das Trigger-Ereignis für das spezifizierte Verhalten eintritt.

Mit der Instanzmethode `wakeupOn(WakeupCondition criteria)` der Klasse `Behavior` überwachen wir die Bedingung für das Auslösen des jeweiligen Triggers (`WakeupCondition`). Da die Klasse `javax.media.j3d.WakeupOnAWTEvent` eine Subklasse von `javax.` **Trigger**

media.j3d.WakeupCondition ist, können wir mit ihr ein WakeupCondition-Objekt konstruieren. Mit der Übergabe des Wertes von KeyEvent.KEY_PRESSED aus der Klasse java.awt.event.KeyEvent ist die Verknüpfung zur Tastatur realisiert. Mit der Klasse javax.media.j3d.WakeupOnElapsedTime konstruieren wir ein WakeupCondition-Objekt, das in 500 Millisekunden triggert. Der folgende Codeausschnitt skizziert diese Konstruktion:

Detail *Trigger*

```
MyBehavior(TransformGroup tg)
      {
         this.tg = tg;
      }
public void initialize()
{
   this.wakeupOn(
         new WakeupOnAWTEvent(
         KeyEvent.KEY_PRESSED));
}
public void processStimulus(Enumeration criteria)
{
...
   this.wakeupOn(
         new WakeupOnElapsedTime(500));
}
```

5.3 KeyPressRotation

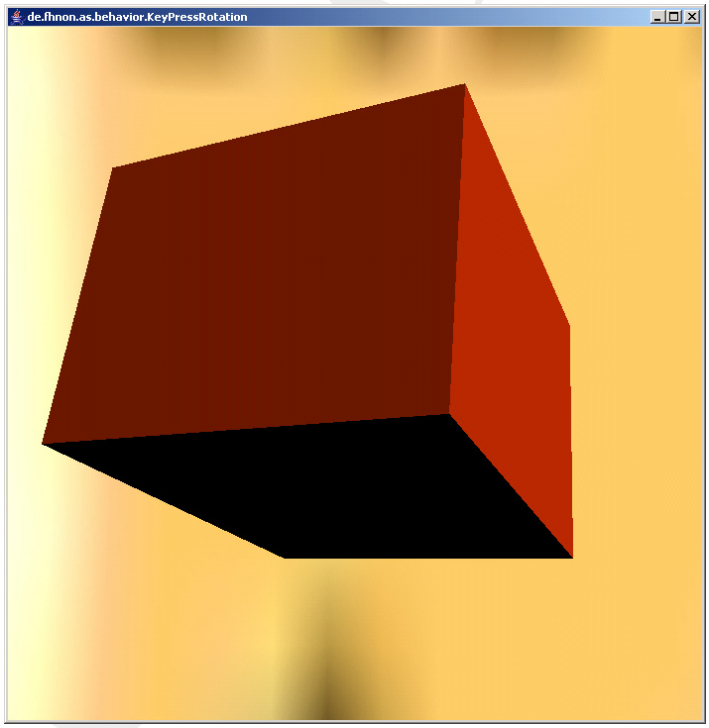
Als Basis haben wir die Klasse SimpleFigure3Db (↪ Abschnitt 4.3.3 S. 80) genutzt und die Kugel durch ein Objekt der Klasse com.sun.j3d.utils.geometry.Box ersetzt. Diese Konstruktion wurde um die innere Klasse MyBehavior ergänzt. Die Textur für den Hintergrund zeigt Abbildung 5.1 S. 143. Das Ergebnis, nach dem drücken einer beliebigen Taste und dem Warten von einigen Sekunden, zeigt Abbildung 5.2 S. 143.

Klasse KeyPressRoatation

```
/**
```

Legende:
Quellcode ↪ S. 142

Abbildung 5.1: Beispiel: KeyPressRotation — Hintergrundtextur



Legende:
Quellcode ↪ S. 142

Abbildung 5.2: Beispiel: KeyPressRotation

```
 *  "Java 3D Example" Rotation auf Tastendruck
 *
 *@author     Bonin
 *@version    1.0
 */

package de.fhnon.as.behavior;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;
import java.awt.event.KeyEvent;
import java.util.Enumeration;

import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Box;
import com.sun.j3d.utils.geometry.Sphere;

import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.universe.SimpleUniverse;

import com.sun.j3d.utils.applet.MainFrame;

import javax.media.j3d.Appearance;
import javax.media.j3d.Background;
import javax.media.j3d.Behavior;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.Material;
import javax.media.j3d.TexCoordGeneration;
import javax.media.j3d.Texture;
import javax.media.j3d.TextureAttributes;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.WakeupOnAWTEvent;
import javax.media.j3d.WakeupOnElapsedTime;

import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;
import javax.vecmath.Color4f;
```

```
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

public class KeyPressRotation extends Applet
{
   private SimpleUniverse u = null;


   public class MyBehavior extends Behavior
   {
      private int angle = 0;

      private TransformGroup tg;
      private Transform3D t3d = new Transform3D();


      MyBehavior(TransformGroup tg)
      {
         this.tg = tg;
      }


      public void initialize()
      {

         this.wakeupOn(
               new WakeupOnAWTEvent(
               KeyEvent.KEY_PRESSED));
      }


      public void processStimulus(
           Enumeration criteria)
      {

      angle += 5;

      t3d.setRotation(
               new AxisAngle4f(
               1f,
               1f,
               0f,
```

```
            (float) Math.toRadians(angle)));

      tg.setTransform(t3d);
      this.wakeupOn(
            new WakeupOnElapsedTime(500));
   }

}



private KeyPressRotation()
{
   super();
}


public BranchGroup createSceneGraph()
{
   BranchGroup bg = new BranchGroup();
   TransformGroup tg = new TransformGroup();
   tg.setCapability(
         TransformGroup.ALLOW_TRANSFORM_WRITE);

   Appearance app = new Appearance();

   app.setMaterial(
         new Material(
         new Color3f(0.9f, 0.2f, 1.0f),
         new Color3f(0.0f, 0.0f, 0.0f),
         new Color3f(0.9f, 0.2f, 1.0f),
         new Color3f(0.0f, 0.0f, 0.0f),
         60.0f));

   Box box =
         new Box(
         0.4f, 0.5f, 0.6f,
         Primitive.GENERATE_NORMALS, app);

   tg.addChild(box);
   bg.addChild(tg);
```

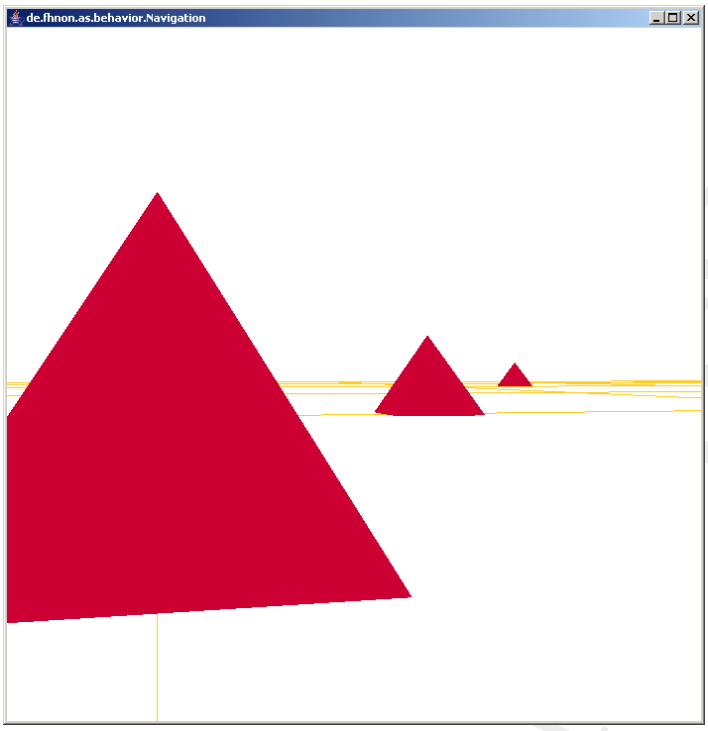
```
      MyBehavior keyPressBehavior =
            new MyBehavior(tg);
      keyPressBehavior.setSchedulingBounds(
            new BoundingSphere());
      bg.addChild(keyPressBehavior);

      bg.compile();
      return bg;
   }


   public BranchGroup createBackground()
   {
      BranchGroup bg =
            new BranchGroup();
      Background back = new Background();
      back.setApplicationBounds(
            getBoundingSphere());
      BranchGroup bgGeometry =
            new BranchGroup();
      Appearance app = new Appearance();

      Texture tex = new TextureLoader(
            "de/fhnon/as/behavior/validxhtml.jpg",
            null).getTexture();
      app.setTexture(tex);

      app.setTexCoordGeneration(
            new TexCoordGeneration(
            TexCoordGeneration.SPHERE_MAP,
            TexCoordGeneration.TEXTURE_COORDINATE_2));
      app.setTextureAttributes(
            new TextureAttributes(
            TextureAttributes.REPLACE,
            new Transform3D(),
            new Color4f(),
            TextureAttributes.NICEST));

      Sphere sphere = new Sphere(1.0f,
            Primitive.GENERATE_TEXTURE_COORDS |
            Primitive.GENERATE_NORMALS_INWARD, 40, app);
```

```
      bgGeometry.addChild(sphere);
      back.setGeometry(bgGeometry);
      bg.addChild(back);
      return bg;
   }


   public void addLights(BranchGroup bg)
   {
      DirectionalLight light =
            new DirectionalLight(
            new Color3f(1.0f, 1.0f, 0.0f),
            new Vector3f(-1.0f, -1.0f, -1.0f));
      light.setInfluencingBounds(
            this.getBoundingSphere());
      bg.addChild(light);
   }


   public TransformGroup createBehaviors(
         BranchGroup bg)
   {
      TransformGroup objTrans =
            new TransformGroup();
      bg.addChild(objTrans);
      return objTrans;
   }


   BoundingSphere getBoundingSphere()
   {
      return new BoundingSphere(
            new Point3d(0.0, 0.0, 0.0), 200.0);
   }


   public void init()
   {
      this.setLayout(new BorderLayout());
      GraphicsConfiguration config =
            SimpleUniverse.getPreferredConfiguration();
      Canvas3D c = new Canvas3D(config);
```

```
      this.add("Center", c);
      u = new SimpleUniverse(c);
      u.getViewingPlatform().
            setNominalViewingTransform();
      u.addBranchGraph(this.createBackground());

      BranchGroup bgRoot = new BranchGroup();
      TransformGroup tg = this.createBehaviors(bgRoot);
      tg.addChild(this.createSceneGraph());
      this.addLights(bgRoot);
      u.addBranchGraph(bgRoot);

   }


   public void destroy()
   {
      u.cleanup();
   }


   public static void main(String[] args)
   {
      new MainFrame(new KeyPressRotation(), 700, 700);
   }

}
```

Protokoll KeyPressRotation.log

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/behavior/KeyPressRotation.java

D:\bonin\artsprog\code>java
  de.fhnon.as.behavior.KeyPressRotation
```



Legende:
Quellcode ↪ S. 151

Abbildung 5.3: Beispiel: `Navigation`

```
D:\bonin\artsprog\code>
```

5.4 `Navigation`

Die Klasse `Navigation` (↪ S. 151) ist ein Beispiel zum Modifizieren der *View Platform* per Tastendruck. Die Idee für dieses Beispiel wurde entnommen aus: *The Java 3D Tutorial — Chapter 4*[1]. Der Quellcode wurde wesentlich anders gestaltet. Grundlage ist die Klasse `com.`-

[1] ↪http://java.sun.com/products/java-media/3D/collateral/j3d_tutorial_ch4.pdf (online 04-May-2004)

Taste	Wirkung	mit Alt-Taste
←	Linksdrehung	Querverschiebung nach links
→	Rechtsdrehung	Querverschiebung nach rechts
↑	Verschiebung nach vorn	
↓	Verschiebung nachc hinten	
PgUp	Drehung nach oben	Verschiebung nach oben
PgDn	Drehung nach unten	Verschiebung nach unten

Tabelle 5.2: Klasse `KeyNavigatorBehavior` — Wirkungen der Tasten

`sun.j3d.utils.behaviors.keyboard.KeyNavigatorBehavior`. Ihr Konstruktor:

```
public KeyNavigatorBehavior(TransformGroup tg)
```

konstruiert ein Verhalten, bezogen auf einen Tastendruck für den Graphen, der an den Parameter `tg` gebunden ist. Die Tabelle 5.2 S. 151 zeigt die Wirkung von Tasten, wenn das Fenster vorab aktiviert wurde (per linken Mausklick).

Klasse `Navigation`

```
/**
 *  "Java 3D Example" Navigation mit Class
 *  com.sun.j3d.utils.behaviors.keyboard.KeyNavigatorBehavior;
 *
 *@author    Bonin
 *@version   1.0
 */

package de.fhnon.as.behavior;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.behaviors.keyboard.KeyNavigatorBehavior;
```

```
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Sphere;
import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.applet.MainFrame;

import javax.media.j3d.Appearance;
import javax.media.j3d.Background;

import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.GeometryArray;
import javax.media.j3d.IndexedTriangleArray;
import javax.media.j3d.LineArray;
import javax.media.j3d.Link;
import javax.media.j3d.Material;
import javax.media.j3d.Shape3D;
import javax.media.j3d.SharedGroup;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;

import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Point3f;
import javax.vecmath.Vector3f;

public class Navigation extends Applet
{
   private SimpleUniverse u = null;


   private Navigation()
   {
      super();
   }


   public BranchGroup createSceneGraph()
   {
      BranchGroup bg = new BranchGroup();
```

```
TransformGroup tg = null;
Vector3f vector = new Vector3f();
Transform3D t3d = new Transform3D();

TransformGroup tgLink = null;

bg.addChild(createLand());

SharedGroup share = new SharedGroup();
share.addChild(createPyramid());

float[][] position =
      {{0.0f, 0.0f, -3.0f},
      {6.0f, 0.0f, 0.0f},
      {6.0f, 0.0f, 6.0f},
      {3.0f, 0.0f, -10.0f},
      {13.0f, 0.0f, -30.0f},
      {-13.0f, 0.0f, 30.0f},
      {-13.0f, 0.0f, 23.0f},
      {13.0f, 0.0f, 3.0f}};

for (int i = 0; i < position.length; i++)
{
   vector.set(position[i]);
   t3d.setTranslation(vector);
   tgLink = new TransformGroup(t3d);
   tgLink.addChild(new Link(share));
   bg.addChild(tgLink);
}
tg = u.getViewingPlatform().
      getViewPlatformTransform();
vector.set(0.0f, 0.45f, 0.0f);
t3d.setTranslation(vector);
tg.setTransform(t3d);
KeyNavigatorBehavior keyNavigating =
      new KeyNavigatorBehavior(tg);
keyNavigating.setSchedulingBounds(
      this.getBoundingSphere());
bg.addChild(keyNavigating);

bg.compile();
```

```
      return bg;
   }


   public BranchGroup createBackground()
   {
      BranchGroup bg =
            new BranchGroup();
      Background back = new Background();
      back.setApplicationBounds(
            getBoundingSphere());
      BranchGroup bgGeometry =
            new BranchGroup();
      Appearance app = new Appearance();

      Sphere sphere = new Sphere(1.0f,
            Primitive.GENERATE_TEXTURE_COORDS |
            Primitive.GENERATE_NORMALS_INWARD,
            40, app);

      bgGeometry.addChild(sphere);
      back.setGeometry(bgGeometry);
      bg.addChild(back);
      return bg;
   }


   public void addLights(BranchGroup bg)
   {
      DirectionalLight light =
            new DirectionalLight(
            new Color3f(0.0f, 0.0f, 1.0f),
            new Vector3f(-1.0f, -1.0f, -1.0f));
      light.setInfluencingBounds(
            this.getBoundingSphere());
      bg.addChild(light);
   }


   public TransformGroup createBehaviors(
         BranchGroup bg)
   {
```

```
      TransformGroup objTrans =
            new TransformGroup();
      bg.addChild(objTrans);
      return objTrans;
   }


   BoundingSphere getBoundingSphere()
   {
      return new BoundingSphere(
            new Point3d(0.0, 0.0, 0.0), 10000.0);
   }


   Shape3D createPyramid()
   {
      IndexedTriangleArray pyramid =
            new IndexedTriangleArray(
            5, GeometryArray.COORDINATES
             | GeometryArray.COLOR_3
            , 12);

      pyramid.setCoordinate(
            0, new Point3f(0.0f, 0.9f, 0.0f));
      pyramid.setCoordinate(
            1, new Point3f(-0.5f, 0.0f, -0.5f));
      pyramid.setCoordinate(
            2, new Point3f(-0.5f, 0.0f, 0.5f));
      pyramid.setCoordinate(
            3, new Point3f(0.5f, 0.0f, 0.5f));
      pyramid.setCoordinate(
            4, new Point3f(0.5f, 0.0f, -0.5f));

      pyramid.setCoordinateIndex(0, 0);
      pyramid.setCoordinateIndex(1, 1);
      pyramid.setCoordinateIndex(2, 2);

      pyramid.setCoordinateIndex(3, 0);
      pyramid.setCoordinateIndex(4, 2);
      pyramid.setCoordinateIndex(5, 3);

      pyramid.setCoordinateIndex(6, 0);
```

```
    pyramid.setCoordinateIndex(7, 3);
    pyramid.setCoordinateIndex(8, 4);

    pyramid.setCoordinateIndex(9, 0);
    pyramid.setCoordinateIndex(10, 4);
    pyramid.setCoordinateIndex(11, 1);

    Color3f color = new Color3f(0.8f, 0.0f, 0.2f);
    pyramid.setColor(0, color);
    pyramid.setColor(1, color);
    pyramid.setColor(2, color);
    pyramid.setColor(3, color);
    pyramid.setColor(4, color);

    return new Shape3D(pyramid);
}


Shape3D createLand()
{
    LineArray landGeom = new LineArray(
          44, GeometryArray.COORDINATES
           | GeometryArray.COLOR_3);
    float l = -50.0f;
    for (int c = 0; c < 44; c += 4)
    {

       landGeom.setCoordinate(
             c + 0, new Point3f(-50.0f, 0.0f, l));
       landGeom.setCoordinate(
             c + 1, new Point3f(50.0f, 0.0f, l));
       landGeom.setCoordinate(
             c + 2, new Point3f(l, 0.0f, -50.0f));
       landGeom.setCoordinate(
             c + 3, new Point3f(l, 0.0f, 50.0f));
       l += 10.0f;
    }

    Color3f c = new Color3f(1.0f, 0.8f, 0.1f);
    for (int i = 0; i < 44; i++)
    {
       landGeom.setColor(i, c);
```

```
      }

      return new Shape3D(landGeom);
   }


   public void init()
   {
      this.setLayout(new BorderLayout());
      GraphicsConfiguration config =
            SimpleUniverse.getPreferredConfiguration();
      Canvas3D c = new Canvas3D(config);
      this.add("Center", c);
      u = new SimpleUniverse(c);
      u.getViewingPlatform().
            setNominalViewingTransform();
      u.addBranchGraph(this.createBackground());

      BranchGroup bgRoot = new BranchGroup();
      TransformGroup tg = this.createBehaviors(bgRoot);
      tg.addChild(this.createSceneGraph());
      this.addLights(bgRoot);
      u.addBranchGraph(bgRoot);

   }


   public void destroy()
   {
      u.cleanup();
   }


   public static void main(String[] args)
   {
      new MainFrame(new Navigation(), 700, 700);
   }

}
```

Protokoll `Navigation.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/behavior/Navigation.java

D:\bonin\artsprog\code>java
  de.fhnon.as.behavior.Navigation

D:\bonin\artsprog\code>
```

5.5 ObjectWithMouse

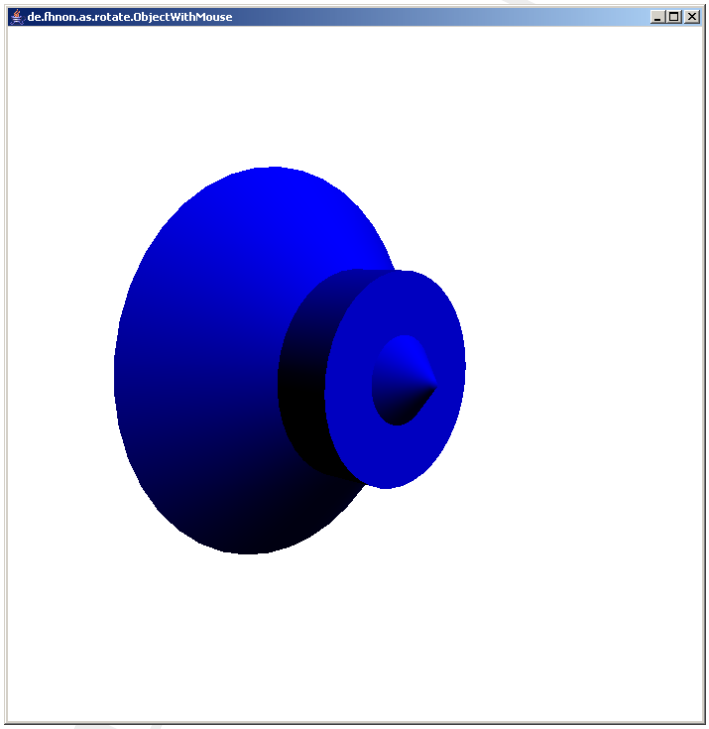
Die Klasse `ObjectWithMouse` (↪ S. 158) ist ein Beispiel zum Modifizieren der *View Platform* per Mausklick. Grundlage bildet die Klasse `com.sun.j3d.utils.behaviors.mouse.MouseRotate`. Sie wird folgendermaßen genutzt:

```
MouseRotate myMouseRotate = new MouseRotate();
myMouseRotate.setTransformGroup(tg);
myMouseRotate.setSchedulingBounds(new BoundingSphere());
bg.addChild(myMouseRotate);
```

Wird die linke Maustaste gedrückt gehalten und die Maus im Fenster bewegt, dann dreht sich das Objekt, hier ein Kegel (Instanz von `com.sun.j3d.utils.geometry.Cone`) kombiniert mit einem Zylinder (Instanz von `com.sun.j3d.utils.geometry.Cylinder`), siehe ↪ Abbildung 5.4 S. 159.

Klasse `ObjectWithMouse`

```
/**
 *  "Java 3D Example" Rotation mit Class
 *  com.sun.j3d.utils.behaviors.mouse.MouseRotate
 *
 *@author     Bonin
 *@version    1.0
 */
```



Legende:
Quellcode ↪ S. 158

Abbildung 5.4: Beispiel: `ObjectWithMouse`

```
package de.fhnon.as.rotate;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.geometry.Cone;
import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Sphere;
import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import javax.media.j3d.Appearance;
import javax.media.j3d.Background;

import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.GeometryArray;
import javax.media.j3d.Material;
import javax.media.j3d.Shape3D;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;

import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

public class ObjectWithMouse extends Applet
{
   private SimpleUniverse u = null;


   private ObjectWithMouse()
   {
      super();
   }


   public BranchGroup createSceneGraph()
```

```
{
   BranchGroup bg = new BranchGroup();

   TransformGroup tg = new TransformGroup();
   tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
   tg.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

   bg.addChild(tg);

   Appearance app = new Appearance();
   Color3f ambientC =
         new Color3f(0.9f, 0.2f, 1.0f);
   Color3f emissiveC =
         new Color3f(0.0f, 0.0f, 0.0f);
   Color3f diffuseC =
         new Color3f(0.9f, 0.2f, 1.0f);
   Color3f specularC =
         new Color3f(0.0f, 0.0f, 0.0f);
   float shininess = 80.0f;

   app.setMaterial(
         new Material(
         ambientC,
         emissiveC,
         diffuseC,
         specularC,
         shininess));

   tg.addChild(new Cone(0.6f, 0.6f,
         Primitive.GENERATE_NORMALS,
         40, 40,
         app));

   tg.addChild(new Cylinder(0.3f, 0.35f,
         Primitive.GENERATE_NORMALS,
         40, 40,
         app));


   MouseRotate myMouseRotate = new MouseRotate();
   myMouseRotate.setTransformGroup(tg);
   myMouseRotate.setSchedulingBounds(
```

```
            new BoundingSphere());
      bg.addChild(myMouseRotate);

      bg.compile();

      return bg;
   }


   public BranchGroup createBackground()
   {
      BranchGroup bg =
            new BranchGroup();
      Background back = new Background();
      back.setApplicationBounds(
            getBoundingSphere());
      BranchGroup bgGeometry =
            new BranchGroup();
      Appearance app = new Appearance();

      Sphere sphere = new Sphere(1.0f,
            Primitive.GENERATE_TEXTURE_COORDS |
            Primitive.GENERATE_NORMALS_INWARD,
            40, app);

      bgGeometry.addChild(sphere);
      back.setGeometry(bgGeometry);
      bg.addChild(back);
      return bg;
   }


   public void addLights(BranchGroup bg)
   {
      DirectionalLight light =
            new DirectionalLight(
            new Color3f(0.0f, 0.0f, 1.0f),
            new Vector3f(-1.0f, -1.0f, -1.0f));
      light.setInfluencingBounds(
            this.getBoundingSphere());
      bg.addChild(light);
   }
```

```
public TransformGroup createBehaviors(
      BranchGroup bg)
{
   TransformGroup objTrans =
         new TransformGroup();
   bg.addChild(objTrans);
   return objTrans;
}


BoundingSphere getBoundingSphere()
{
   return new BoundingSphere(
         new Point3d(0.0, 0.0, 0.0), 10000.0);
}


public void init()
{
   this.setLayout(new BorderLayout());
   GraphicsConfiguration config =
         SimpleUniverse.getPreferredConfiguration();
   Canvas3D c = new Canvas3D(config);
   this.add("Center", c);
   u = new SimpleUniverse(c);
   u.getViewingPlatform().
         setNominalViewingTransform();
   u.addBranchGraph(this.createBackground());

   BranchGroup bgRoot = new BranchGroup();
   TransformGroup tg = this.createBehaviors(bgRoot);
   tg.addChild(this.createSceneGraph());
   this.addLights(bgRoot);
   u.addBranchGraph(bgRoot);

}


public void destroy()
{
```

```
      u.cleanup();
   }


   public static void main(String[] args)
   {
      new MainFrame(new ObjectWithMouse(), 700, 700);
   }

}
```

Protokoll `ObjectWithMouse.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/rotate/ObjectWithMouse.java

D:\bonin\artsprog\code>java
  de.fhnon.as.rotate.ObjectWithMouse

D:\bonin\artsprog\code>
```

Kapitel 6

Beispielprojekt Jagdhund

Um einen Jagdhund, beispielsweise der Rasse „Deutscher Wachtelhund“ (DW), darzustellen, ist dieser in Java3D vollständig aus Dreiecken (*triangles*) und/oder Vierecken (*quads*) zu konstruieren.

Die Erfassung der großen Menge solcher Konstrukte ist sehr aufwendig.

Sie wird in Java3D etwas erleichtert, weil Polygone mit Hilfe der Klasse `com.sun.j3d.utils.geometry.GeometryInfo` automatisch in solche Konstrukte konvertiert werden.

Trainingsplan

Das Kapitel „Beispielprojekt Jagdhund“ zeigt:

- eine Jagdhundskizze konstruiert aus Polygonen mit 30 Punkten und
↪ Seite 166 . . .

- die entsprechende Java-Klasse mit ihren genutzen Daten.
  ↪ Seite 170 . . .

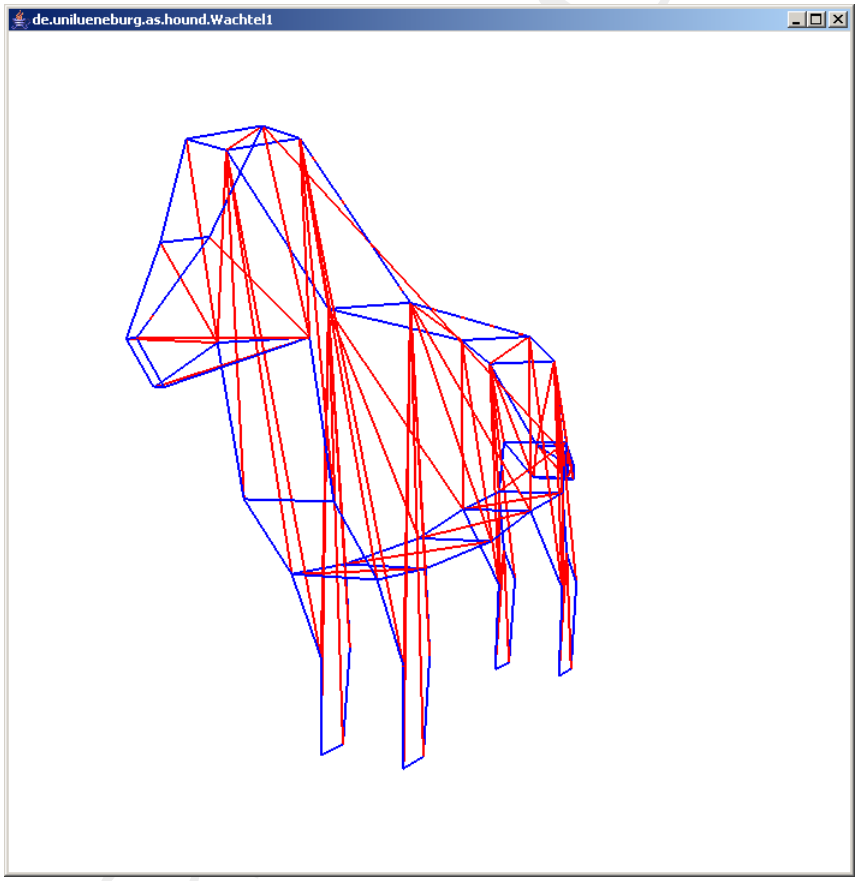
6.1 1. Ansatz mit `GeometryInfo` — `Wachtel1`

Im 1. Ansatz des Projektes „Jagdhund“ konstruieren wir einen Wachtel aus zwei quasi parallel verlaufenden Polygonen mit 30 Punkten (von P_0 bis P_{28}), wobei der erste Punkt dieselben Koordinatenwerte hat wie der dreißigste Punkt. So entsteht ein „geschlossenes Polygon“. Die beiden Polygone unterscheiden sich nur durch die z-Werte ihrer Punkte. Die Dicke des Hundes ist am Rutenende und am Fang wesentlich kleiner als am Rücken. Das wird durch kleinere z-Werte erreicht, weil die x-Achse als Längsachse des Hundes dient. Der Nullpunkt des Koordinatensystems liegt ungefähr im Hundemagen. Zusätzlich spezifizieren wir Polygone, jeweils aus fünf Koordinatenangaben (Anfangspunkt gleich Endpunkt), die die beiden Hundeseiten miteinander verknüpfen. Das Ergebnis zeigt Abbildung 6.1 S. 167.

Die automatische Berechnung der Dreiecke aus den Polygonen verläuft nach folgendem Verfahren:

Detail *Triangles*-Ermittlung

```
float[] coordinateData = createCoordinateData();

// für jedes Polygon die Anzahl seiner Punkte
int[] stripCount = {30, 30, 5, 5, ...};

GeometryInfo gi = new GeometryInfo(
  GeometryInfo.POLYGON_ARRAY);
gi.setCoordinates(coordinateData);
gi.setStripCounts(stripCount);

Triangulator tr = new Triangulator();

tr.triangulate(gi);
```



Legende:
Quellcode ↪ S. 170

Abbildung 6.1: Beispiel: `Wachtel1` — Dreiecke

```
NormalGenerator ng = new NormalGenerator();
ng.generateNormals(gi);

Stripifier st = new Stripifier();
st.stripify(gi);

Shape3D figure = new Shape3D();

figure.setAppearance(...);

figure.setGeometry(gi.getGeometryArray());
```

Mit dem Parameterwert `GeometryInfo.POLYGON_ARRAY` zeigen wir der `GeometryInfo`-Instanz `gi` an, dass unsere Koordinaten auch nichtplanare Polygone spezifizieren. Mit der Klasse `com.sun.j3d.utils.geometry.Triangulator` erzeugen wir daraus die Dreiecke. Die zugehörende *Normals* berechnet die Instanzmethode `generateNormals(gi)`, der Klasse `com.sun.j3d.utils.geometry.NormalGenerator`. Anschließend ändern wir die Primitiven (Dreiecke) in unserem `GeometryInfo`-Objekt `gi` in *Triangle Strips*. Den Wachtel, das eigenliche `Shape3D`-Objekt `figure`, erzeugen wir dann mit der Geometrie, die wir von `gi` durch Anwendung der Methode `getGeometryArray()` erhalten.

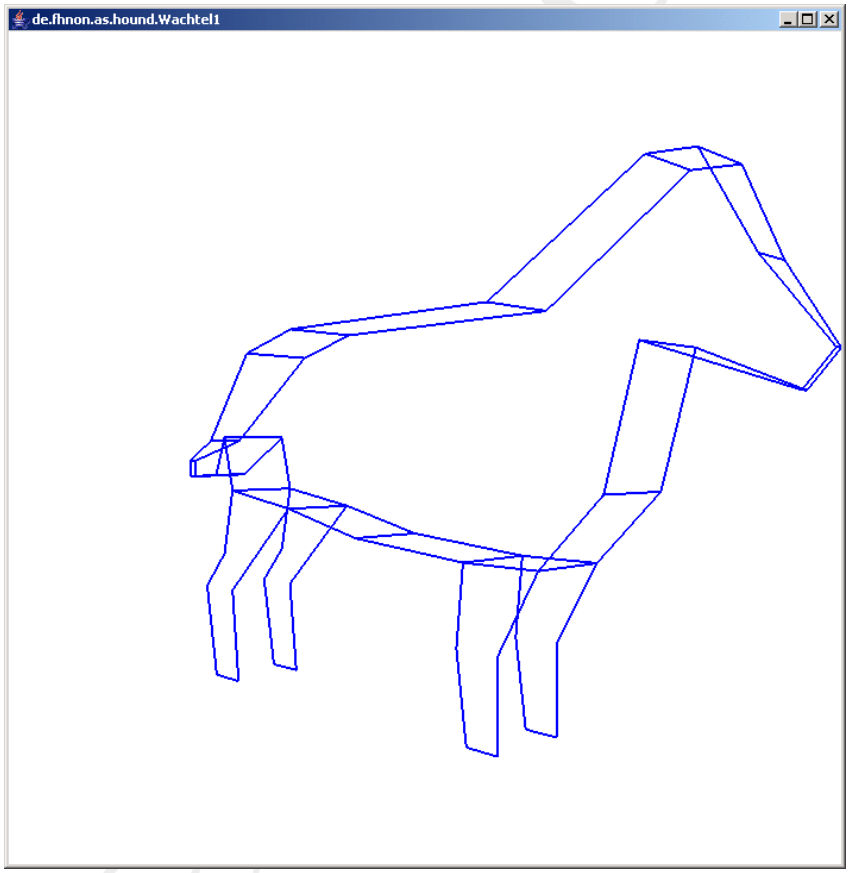
Mit der Klasse `javax.media.j3d.LineStripArray` stellen wir unsere Ausgangspolygone dar (↪ Abbildung 6.2 S. 169). Diese Klasse hat folgenden Konstruktor:

```
LineStripArray(
  int vertexCount,  // hier 160
  int vertexFormat, // hier LineArray.COORDINATES
  int[] stripVertexCounts) // hier stripCount
```

Damit erzeugen wir das leere `LineStripArray`-Objekt `lineArray`. Die Koordinaten verknüpfen wir mit der Methode:

```
setCoordinate(
  int index, // hier 0
  float[] coordinate) // hier coordinateData
```

Ihr erster Parameter `index` gibt die Einstiegsecke in der Geometry an. Der zweite Parameter `coordinate` ist die Referenz zu den einzelnen



Legende:
Quellcode ↪ S. 170

Abbildung 6.2: Beispiel: `Wachtel1` — Ausgangspolygone

Polygonpunkten, strukturiert als 3 Werte für einen Punkt.

Um die erzeugten Dreiecke besser sehen zu können, haben wir ein langsames, permanentes Drehen des Hundes eingebaut und zwar folgendermaßen:

Detail *Rotation*

```
Alpha rotationAlpha = new Alpha(-1, 144000);

RotationInterpolator rotator =
            new RotationInterpolator(
            rotationAlpha, objSpin);

rotator.setSchedulingBounds(getBoundingSphere());

objSpin.addChild(rotator);
```

Mit dem Wert `-1` für den ersten Parameter beim Konstruktor der Klasse `javax.media.j3d.Alpha` wird eine permanente Rotation erreicht. Der zweite Wert, hier 144000, legt die Periodendauer fest, die *Alpha* benötigt um von null bis eins zu gehen.

6.2 Klasse `Wachtel1`

Quellcode `Wachtel1.java`

```
/**
 *  Projekt Jagdhund 1. Ansatz
 *
 *@author     Bonin
 *@version    1.0
 */

package de.unilueneburg.as.hound;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.geometry.GeometryInfo;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.NormalGenerator;
import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.geometry.Stripifier;
import com.sun.j3d.utils.geometry.Triangulator;
```

```
import javax.media.j3d.Alpha;
import javax.media.j3d.AmbientLight;
import javax.media.j3d.Appearance;
import javax.media.j3d.Background;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.ColoringAttributes;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.LineArray;
import javax.media.j3d.LineAttributes;
import javax.media.j3d.LineStripArray;
import javax.media.j3d.Material;
import javax.media.j3d.RotationInterpolator;
import javax.media.j3d.Shape3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.Transform3D;
import javax.media.j3d.PolygonAttributes;

import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;
import javax.vecmath.Vector3f;

public class Wachtel1 extends Applet
{

   private SimpleUniverse u = null;


   private Wachtel1()
   {
      super();
   }



   BranchGroup createSceneGraph()
   {

      BranchGroup bg = new BranchGroup();

      float[] coordinateData =
           createCoordinateData();

      int[] stripCount = {30, 30, 5, 5, 5, 5, 5, 5, 5,
           5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5};

      GeometryInfo gi = new GeometryInfo(
           GeometryInfo.POLYGON_ARRAY);
      gi.setCoordinates(coordinateData);
      gi.setStripCounts(stripCount);

      Triangulator tr = new Triangulator();
```

```
tr.triangulate(gi);

/*
 *  to guarantee connection information
 */
gi.recomputeIndices();

NormalGenerator ng = new NormalGenerator();
ng.generateNormals(gi);
gi.recomputeIndices();

Stripifier st = new Stripifier();
st.stripify(gi);
gi.recomputeIndices();

Shape3D figure = new Shape3D();

figure.setAppearance(
      createWireFrameAppearance());

figure.setGeometry(gi.getGeometryArray());

TransformGroup objSpin = new TransformGroup();
objSpin.setCapability(
      TransformGroup.ALLOW_TRANSFORM_WRITE);
bg.addChild(objSpin);

objSpin.addChild(figure);

LineStripArray lineArray = new LineStripArray(
      160,
      LineArray.COORDINATES,
      stripCount);

lineArray.setCoordinates(0, coordinateData);

objSpin.addChild(
      new Shape3D(
      lineArray, createMyColorAppearance()));

Alpha rotationAlpha = new Alpha(-1, 144000);
RotationInterpolator rotator =
      new RotationInterpolator(
      rotationAlpha, objSpin);

rotator.setSchedulingBounds(getBoundingSphere());
objSpin.addChild(rotator);

addLights(bg);

bg.compile();
```

```
   return bg;
}


Appearance createMyColorAppearance()
{
   Appearance app =
         new Appearance();
   ColoringAttributes coloringAtt =
         new ColoringAttributes();
   coloringAtt.setColor(0.0f, 0.0f, 1.0f);
   app.setColoringAttributes(
         coloringAtt);
   LineAttributes lineAttrib =
         new LineAttributes();
   lineAttrib.setLineWidth(1.5f);
   app.setLineAttributes(
         lineAttrib);
   return app;
}



Appearance createWireFrameAppearance()
{

   Appearance app =
         new Appearance();
   PolygonAttributes polyAtt =
         new PolygonAttributes();
   polyAtt.setPolygonMode(
         PolygonAttributes.POLYGON_LINE);
   app.setPolygonAttributes(polyAtt);
   ColoringAttributes coloringAtt =
         new ColoringAttributes();
   coloringAtt.setColor(1.0f, 0.0f, 0.0f);
   app.setColoringAttributes(coloringAtt);
   LineAttributes lineAttrib =
         new LineAttributes();
   lineAttrib.setLineWidth(1.5f);
   app.setLineAttributes(
         lineAttrib);
   return app;
}


BranchGroup createBackground()
{
   BranchGroup bg = new BranchGroup();
   Background background = new Background();
   background.setColor(1.0f, 1.0f, 1.0f);
   background.setApplicationBounds(
         getBoundingSphere());
```

```
      bg.addChild(background);
      return bg;
   }



   void addLights(BranchGroup bg)
   {
      DirectionalLight lightD = new DirectionalLight();
      lightD.setDirection(
            new Vector3f(0.0f, -0.7f, -0.7f));
      lightD.setInfluencingBounds(
            getBoundingSphere());
      bg.addChild(lightD);

      AmbientLight lightA = new AmbientLight();
      lightA.setInfluencingBounds(
            getBoundingSphere());
      bg.addChild(lightA);
   }


   BoundingSphere getBoundingSphere()
   {
      return new BoundingSphere();
   }


   public void init()
   {
      this.setLayout(new BorderLayout());
      GraphicsConfiguration config =
            SimpleUniverse.getPreferredConfiguration();
      Canvas3D c = new Canvas3D(config);
      this.add("Center", c);
      u = new SimpleUniverse(c);
      u.getViewingPlatform().
            setNominalViewingTransform();

      u.addBranchGraph(this.createBackground());

      u.addBranchGraph(
            this.createSceneGraph());

   }


   public void destroy()
   {
      u.cleanup();
   }
```

```
public static void main(String[] args)
{
   new MainFrame(
         new Wachtel1(), 700, 700);
}


/*
 *  only test data from an other example
 */
float[] createCoordinateData()
{

   float[] data = new float[160 * 3];

   int i = 0;
   /*
    *  Front side
    */
   data[i++] = 0.94f;
   data[i++] = 0.1f;
   data[i++] = 0.01f;
   // 0
   data[i++] = 0.67f;
   data[i++] = 0.2f;
   data[i++] = 0.1f;
   // 1
   data[i++] = 0.59f;
   data[i++] = -0.09f;
   data[i++] = 0.1f;
   // 2
   data[i++] = 0.43f;
   data[i++] = -0.25f;
   data[i++] = 0.1f;
   // 3
   data[i++] = 0.32f;
   data[i++] = -0.44f;
   data[i++] = 0.1f;
   // 4
   data[i++] = 0.32f;
   data[i++] = -0.65f;
   data[i++] = 0.1f;
   // 5
   data[i++] = 0.23f;
   data[i++] = -0.65f;
   data[i++] = 0.1f;
   // 6
   data[i++] = 0.2f;
   data[i++] = -0.44f;
   data[i++] = 0.1f;
   // 7
   data[i++] = 0.22f;
   data[i++] = -0.25f;
```

```
data[i++] = 0.1f;
// 8
data[i++] = -0.14f;
data[i++] = -0.22f;
data[i++] = 0.1f;
// 9
data[i++] = -0.41f;
data[i++] = -0.16f;
data[i++] = 0.1f;
// 10
data[i++] = -0.67f;
data[i++] = -0.4f;
data[i++] = 0.1f;
// 11
data[i++] = -0.64f;
data[i++] = -0.65f;
data[i++] = 0.1f;
// 12
data[i++] = -0.75f;
data[i++] = -0.65f;
data[i++] = 0.1f;
// 13
data[i++] = -0.8f;
data[i++] = -0.4f;
data[i++] = 0.1f;
// 14
data[i++] = -0.71f;
data[i++] = -0.3f;
data[i++] = 0.1f;
// 15
data[i++] = -0.67f;
data[i++] = -0.12f;
data[i++] = 0.1f;
// 16
data[i++] = -0.71f;
data[i++] = 0.03f;
data[i++] = 0.1f;
// 17
data[i++] = -0.83f;
data[i++] = -0.08f;
data[i++] = 0.05f;
// 18
data[i++] = -1.04f;
data[i++] = -0.09f;
data[i++] = 0.01f;
// 19
data[i++] = -1.04f;
data[i++] = -0.04f;
data[i++] = 0.01f;
// 20
data[i++] = -0.86f;
data[i++] = 0.02f;
data[i++] = 0.05f;
```

```
//  21
data[i++] = -0.6f;
data[i++] = 0.26f;
data[i++] = 0.1f;
//  22
data[i++] = -0.4f;
data[i++] = 0.31f;
data[i++] = 0.1f;
// 23
data[i++] = 0.29f;
data[i++] = 0.31f;
data[i++] = 0.1f;
//  24
data[i++] = 0.67f;
data[i++] = 0.55f;
data[i++] = 0.08f;
// 25
data[i++] = 0.78f;
data[i++] = 0.54f;
data[i++] = 0.08f;
// 26
data[i++] = 0.88f;
data[i++] = 0.34f;
data[i++] = 0.05f;
// 27
data[i++] = 1.0f;
data[i++] = 0.17f;
data[i++] = 0.01f;
// 28
data[i++] = 0.94f;
data[i++] = 0.1f;
data[i++] = 0.01f;
//0


/*
 *  Back side
 */
data[i++] = 0.94f;
data[i++] = 0.1f;
data[i++] = -0.01f;
// 0
data[i++] = 0.67f;
data[i++] = 0.2f;
data[i++] = -0.1f;
// 1
data[i++] = 0.59f;
data[i++] = -0.09f;
data[i++] = -0.1f;
// 2
data[i++] = 0.43f;
data[i++] = -0.25f;
data[i++] = -0.1f;
```

```
// 3
data[i++] = 0.32f;
data[i++] = -0.44f;
data[i++] = -0.1f;
// 4
data[i++] = 0.32f;
data[i++] = -0.65f;
data[i++] = -0.1f;
// 5
data[i++] = 0.23f;
data[i++] = -0.65f;
data[i++] = -0.1f;
// 6
data[i++] = 0.2f;
data[i++] = -0.44f;
data[i++] = -0.1f;
// 7
data[i++] = 0.22f;
data[i++] = -0.25f;
data[i++] = -0.1f;
// 8
data[i++] = -0.14f;
data[i++] = -0.22f;
data[i++] = -0.1f;
// 9
data[i++] = -0.41f;
data[i++] = -0.16f;
data[i++] = -0.1f;
// 10
data[i++] = -0.67f;
data[i++] = -0.4f;
data[i++] = -0.1f;
// 11
data[i++] = -0.64f;
data[i++] = -0.65f;
data[i++] = -0.1f;
// 12
data[i++] = -0.75f;
data[i++] = -0.65f;
data[i++] = -0.1f;
// 13
data[i++] = -0.8f;
data[i++] = -0.4f;
data[i++] = -0.1f;
// 14
data[i++] = -0.71f;
data[i++] = -0.3f;
data[i++] = -0.1f;
// 15
data[i++] = -0.67f;
data[i++] = -0.12f;
data[i++] = -0.1f;
// 16
```

```
data[i++] = -0.71f;
data[i++] = 0.03f;
data[i++] = -0.1f;
// 17
data[i++] = -0.83f;
data[i++] = -0.08f;
data[i++] = -0.05f;
// 18
data[i++] = -1.04f;
data[i++] = -0.09f;
data[i++] = -0.01f;
// 19
data[i++] = -1.04f;
data[i++] = -0.04f;
data[i++] = -0.01f;
// 20
data[i++] = -0.86f;
data[i++] = 0.02f;
data[i++] = -0.05f;
//  21
data[i++] = -0.6f;
data[i++] = 0.26f;
data[i++] = -0.1f;
//  22
data[i++] = -0.4f;
data[i++] = 0.31f;
data[i++] = -0.1f;
// 23
data[i++] = 0.29f;
data[i++] = 0.31f;
data[i++] = -0.1f;
//  24
data[i++] = 0.67f;
data[i++] = 0.55f;
data[i++] = -0.08f;
// 25
data[i++] = 0.78f;
data[i++] = 0.54f;
data[i++] = -0.08f;
// 26
data[i++] = 0.88f;
data[i++] = 0.34f;
data[i++] = -0.05f;
// 27
data[i++] = 1.0f;
data[i++] = 0.17f;
data[i++] = -0.01f;
// 28
data[i++] = 0.94f;
data[i++] = 0.1f;
data[i++] = -0.01f;
//0
```

```
/*
 *  Ruecken 23--24
 */
data[i++] = -0.4f;
data[i++] = 0.31f;
data[i++] = +0.1f;
// 23 Front
data[i++] = -0.4f;
data[i++] = 0.31f;
data[i++] = -0.1f;
// 23 Back
data[i++] = 0.29f;
data[i++] = 0.31f;
data[i++] = -0.1f;
//  24 Back
data[i++] = 0.29f;
data[i++] = 0.31f;
data[i++] = 0.1f;
//  24 Front
data[i++] = -0.4f;
data[i++] = 0.31f;
data[i++] = +0.1f;
// 23 Front

/*
 *  Hals oben 24--25
 */
data[i++] = 0.29f;
data[i++] = 0.31f;
data[i++] = 0.1f;
//  24 Front
data[i++] = 0.29f;
data[i++] = 0.31f;
data[i++] = -0.1f;
//  24 Back
data[i++] = 0.67f;
data[i++] = 0.55f;
data[i++] = -0.08f;
// 25 Back
data[i++] = 0.67f;
data[i++] = 0.55f;
data[i++] = 0.08f;
// 25 Front
data[i++] = 0.29f;
data[i++] = 0.31f;
data[i++] = 0.1f;
//  24 Front

/*
 *  Kopf oben 25--26
 */
data[i++] = 0.67f;
data[i++] = 0.55f;
```

```
data[i++] = 0.08f;
// 25 Front
data[i++] = 0.67f;
data[i++] = 0.55f;
data[i++] = -0.08f;
// 25 Back
data[i++] = 0.78f;
data[i++] = 0.54f;
data[i++] = -0.08f;
// 26 Back
data[i++] = 0.78f;
data[i++] = 0.54f;
data[i++] = 0.08f;
// 26 Front
data[i++] = 0.67f;
data[i++] = 0.55f;
data[i++] = 0.08f;
// 25 Front

/*
 *  Stoss 26--27
 */
data[i++] = 0.78f;
data[i++] = 0.54f;
data[i++] = 0.08f;
// 26 Front
data[i++] = 0.78f;
data[i++] = 0.54f;
data[i++] = -0.08f;
// 26 Back
data[i++] = 0.88f;
data[i++] = 0.34f;
data[i++] = -0.05f;
// 27 Back
data[i++] = 0.88f;
data[i++] = 0.34f;
data[i++] = 0.05f;
// 27 Front
data[i++] = 0.78f;
data[i++] = 0.54f;
data[i++] = 0.08f;
// 26 Front

/*
 *  Schnauze oben 27--28
 */
data[i++] = 0.88f;
data[i++] = 0.34f;
data[i++] = 0.05f;
// 27 Front
data[i++] = 0.88f;
data[i++] = 0.34f;
data[i++] = -0.05f;
```

```
// 27 Back
data[i++] = 1.0f;
data[i++] = 0.17f;
data[i++] = -0.01f;
// 28 Back
data[i++] = 1.0f;
data[i++] = 0.17f;
data[i++] = 0.01f;
// 28 Front
data[i++] = 0.88f;
data[i++] = 0.34f;
data[i++] = 0.05f;
// 27 Front

/*
 *  Schnauze vorn 28--0
 */
data[i++] = 1.0f;
data[i++] = 0.17f;
data[i++] = 0.01f;
// 28 Front
data[i++] = 1.0f;
data[i++] = 0.17f;
data[i++] = -0.01f;
// 28 Back
data[i++] = 0.94f;
data[i++] = 0.1f;
data[i++] = -0.01f;
// 0 Back
data[i++] = 0.94f;
data[i++] = 0.1f;
data[i++] = 0.01f;
// 0 Front
data[i++] = 1.0f;
data[i++] = 0.17f;
data[i++] = 0.01f;
// 28 Front

/*
 *  Schnauze unten 0--1
 */
data[i++] = 0.94f;
data[i++] = 0.1f;
data[i++] = 0.01f;
// 0 Front
data[i++] = 0.94f;
data[i++] = 0.1f;
data[i++] = -0.01f;
// 0 Back
data[i++] = 0.67f;
data[i++] = 0.2f;
data[i++] = -0.1f;
// 1 Back
```

```
data[i++] = 0.67f;
data[i++] = 0.2f;
data[i++] = 0.1f;
// 1 Front
data[i++] = 0.94f;
data[i++] = 0.1f;
data[i++] = 0.01f;
// 0 Front

/*
 *  Hals unten 1--2
 */
data[i++] = 0.67f;
data[i++] = 0.2f;
data[i++] = 0.1f;
// 1 Front
data[i++] = 0.67f;
data[i++] = 0.2f;
data[i++] = -0.1f;
// 1 Back
data[i++] = 0.59f;
data[i++] = -0.09f;
data[i++] = -0.1f;
// 2 Back
data[i++] = 0.59f;
data[i++] = -0.09f;
data[i++] = 0.1f;
// 2 Front
data[i++] = 0.67f;
data[i++] = 0.2f;
data[i++] = 0.1f;
// 1 Front

/*
 *  Brust 2--3
 */
data[i++] = 0.59f;
data[i++] = -0.09f;
data[i++] = 0.1f;
// 2 Front
data[i++] = 0.59f;
data[i++] = -0.09f;
data[i++] = -0.1f;
// 2 Back
data[i++] = 0.43f;
data[i++] = -0.25f;
data[i++] = -0.1f;
// 3 Back
data[i++] = 0.43f;
data[i++] = -0.25f;
data[i++] = 0.1f;
// 3 Front
data[i++] = 0.59f;
```

```
data[i++] = -0.09f;
data[i++] = 0.1f;
// 2 Front

/*
 *  Brust unten 3--8
 */
data[i++] = 0.43f;
data[i++] = -0.25f;
data[i++] = 0.1f;
// 3 Front
data[i++] = 0.43f;
data[i++] = -0.25f;
data[i++] = -0.1f;
// 3 Back
data[i++] = 0.22f;
data[i++] = -0.25f;
data[i++] = -0.1f;
// 8 Back
data[i++] = 0.22f;
data[i++] = -0.25f;
data[i++] = 0.1f;
// 8 Front
data[i++] = 0.43f;
data[i++] = -0.25f;
data[i++] = 0.1f;
// 3 Front

/*
 *  Bauch vorn 8--9
 */
data[i++] = 0.22f;
data[i++] = -0.25f;
data[i++] = 0.1f;
// 8 Front
data[i++] = 0.22f;
data[i++] = -0.25f;
data[i++] = -0.1f;
// 8 Back
data[i++] = -0.14f;
data[i++] = -0.22f;
data[i++] = -0.1f;
// 9 Back
data[i++] = -0.14f;
data[i++] = -0.22f;
data[i++] = 0.1f;
// 9 Front
data[i++] = 0.22f;
data[i++] = -0.25f;
data[i++] = 0.1f;
// 8 Front

/*
```

```
 *  Bauch hinten 9--10
 */
data[i++] = -0.14f;
data[i++] = -0.22f;
data[i++] = 0.1f;
// 9 Front
data[i++] = -0.14f;
data[i++] = -0.22f;
data[i++] = -0.1f;
// 9 Back
data[i++] = -0.41f;
data[i++] = -0.16f;
data[i++] = -0.1f;
// 10 Back
data[i++] = -0.41f;
data[i++] = -0.16f;
data[i++] = 0.1f;
// 10 Front
data[i++] = -0.14f;
data[i++] = -0.22f;
data[i++] = 0.1f;
// 9 Front

/*
 *  Bauch ganz hinten 10--16
 */
data[i++] = -0.41f;
data[i++] = -0.16f;
data[i++] = 0.1f;
// 10 Front
data[i++] = -0.41f;
data[i++] = -0.16f;
data[i++] = -0.1f;
// 10 Back
data[i++] = -0.67f;
data[i++] = -0.12f;
data[i++] = -0.1f;
// 16 Back
data[i++] = -0.67f;
data[i++] = -0.12f;
data[i++] = 0.1f;
// 16 Front
data[i++] = -0.41f;
data[i++] = -0.16f;
data[i++] = 0.1f;
// 10 Front

/*
 *  Schnalle 16--17
 */
data[i++] = -0.67f;
data[i++] = -0.12f;
data[i++] = 0.1f;
```

```
// 16 Front
data[i++] = -0.67f;
data[i++] = -0.12f;
data[i++] = -0.1f;
// 16 Back
data[i++] = -0.71f;
data[i++] = 0.03f;
data[i++] = -0.1f;
// 17 Back
data[i++] = -0.71f;
data[i++] = 0.03f;
data[i++] = 0.1f;
// 17 Front
data[i++] = -0.67f;
data[i++] = -0.12f;
data[i++] = 0.1f;
// 16 Front

/*
 *  Rute unten 17--18
 */
data[i++] = -0.71f;
data[i++] = 0.03f;
data[i++] = 0.1f;
// 17 Front
data[i++] = -0.71f;
data[i++] = 0.03f;
data[i++] = -0.1f;
// 17 Back
data[i++] = -0.83f;
data[i++] = -0.08f;
data[i++] = -0.05f;
// 18 Back
data[i++] = -0.83f;
data[i++] = -0.08f;
data[i++] = 0.05f;
// 18 Front
data[i++] = -0.71f;
data[i++] = 0.03f;
data[i++] = 0.1f;
// 17 Front

/*
 *  Rute unten hinten 18--19
 */
data[i++] = -0.83f;
data[i++] = -0.08f;
data[i++] = 0.05f;
// 18 Front
data[i++] = -0.83f;
data[i++] = -0.08f;
data[i++] = -0.05f;
// 18 Back
```

```
data[i++] = -1.04f;
data[i++] = -0.09f;
data[i++] = -0.01f;
// 19 Back
data[i++] = -1.04f;
data[i++] = -0.09f;
data[i++] = 0.01f;
// 19 Front
data[i++] = -0.83f;
data[i++] = -0.08f;
data[i++] = 0.05f;
// 18 Front

/*
 *  Rute hinten 19--20
 */
data[i++] = -1.04f;
data[i++] = -0.09f;
data[i++] = 0.01f;
// 19 Front
data[i++] = -1.04f;
data[i++] = -0.09f;
data[i++] = -0.01f;
// 19 Back
data[i++] = -1.04f;
data[i++] = -0.04f;
data[i++] = -0.01f;
// 20 Back
data[i++] = -1.04f;
data[i++] = -0.04f;
data[i++] = 0.01f;
// 20 Front
data[i++] = -1.04f;
data[i++] = -0.09f;
data[i++] = 0.01f;
// 19 Front

/*
 *  Rute hinten oben 20--21
 */
data[i++] = -1.04f;
data[i++] = -0.04f;
data[i++] = 0.01f;
// 20 Front
data[i++] = -1.04f;
data[i++] = -0.04f;
data[i++] = -0.01f;
// 20 Back
data[i++] = -0.86f;
data[i++] = 0.02f;
data[i++] = -0.05f;
//  21 Back
data[i++] = -0.86f;
```

```
data[i++] = 0.02f;
data[i++] = 0.05f;
//  21 Front
data[i++] = -1.04f;
data[i++] = -0.04f;
data[i++] = 0.01f;
// 20 Front

/*
 *  Rute oben 21--22
 */
data[i++] = -0.86f;
data[i++] = 0.02f;
data[i++] = 0.05f;
//  21 Front
data[i++] = -0.86f;
data[i++] = 0.02f;
data[i++] = -0.05f;
//  21 Back
data[i++] = -0.6f;
data[i++] = 0.26f;
data[i++] = -0.1f;
//  22 Back
data[i++] = -0.6f;
data[i++] = 0.26f;
data[i++] = 0.1f;
//  22 Front
data[i++] = -0.86f;
data[i++] = 0.02f;
data[i++] = 0.05f;
//  21 Front

/*
 *  Kruppe 22--23
 */
data[i++] = -0.6f;
data[i++] = 0.26f;
data[i++] = 0.1f;
//  22 Front
data[i++] = -0.6f;
data[i++] = 0.26f;
data[i++] = -0.1f;
//  22 Back
data[i++] = -0.4f;
data[i++] = 0.31f;
data[i++] = -0.1f;
// 23 Back
data[i++] = -0.4f;
data[i++] = 0.31f;
data[i++] = 0.1f;
// 23 Front
data[i++] = -0.6f;
data[i++] = 0.26f;
```

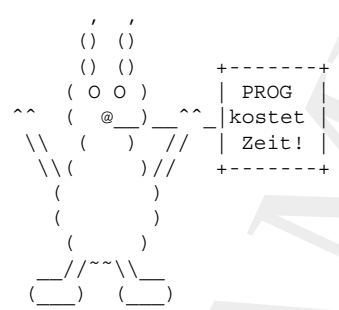
```
      data[i++] = 0.1f;
      //  22 Front

      return data;
   }

}
```

Protokoll `Wachtel1.log`

```
D:\bonin\prog\code>java -version
java version "1.5.0_04"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_04-b05)
Java HotSpot(TM) Client VM
  (build 1.5.0_04-b05, mixed mode, sharing)

D:\bonin\prog\code>javac -deprecation
  de/unilueneburg/as/hound/Wachtel1.java
de/unilueneburg/as/hound/Wachtel1.java:73:
warning: Triangulator() in
  com.sun.j3d.utils.geometry.Triangulator
  has been deprecated
  Triangulator tr = new Triangulator();
                        ^
1 warning

D:\bonin\prog\code>java de.unilueneburg.as.hound.Wachtel1
```

Kapitel 7

Ausblick



Jeder Text von derartiger Länge und „Tiefe“ verlangt ein abschließendes Wort für seinen getreuen Leser. Es wäre nicht fair, nach so vielen Seiten, die nächste aufzuschlagen und dann den Anhang zu finden. Daher zum Schluß ein kleiner Ausblick. Wenn man in Zukunft weit komplexere Aufgaben mit hinreichender Qualität programmieren will, dann bedarf es eines neuen, oder zumindest fortentwickelten Paradigmas.

Die Objekt-Orientierung in der klassischen Ausprägung von Java stößt schon heute oft an ihre Grenzen. Der hier gewählte Ansatz, die vielfältigen Konstrukte der Programmierung durch viele Beispiele aus dem Bereich Graphik zu verdeutlichen, möge Ihnen gefallen haben. Ich wünsche Ihnen, dass Sie beim Durcharbeiten dieses Buches die Faszination der Programmierung selbst ausgiebig erfahren haben. Zukünftigen Paradigmen der Programmierung werden Sie so motiviert sicherlich positiv gegenüberstehen.

Anhang A

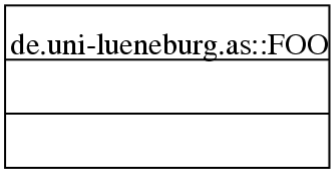
Übungen

Dieser Abschnitt enthält einige Aufgaben, die im Rahmen des Faches *Programmierung* im Studiengang *Wirtschaftsinformatik* der Fakultät III der Universität Lüneburg im Zeitraum von 2005–2006 gestellt wurden. Solche Aufgaben sind mit dem Zeichen † und anschließender Punktzahl nP gekennzeichnet.

Bei einer Klausur von 120 Minuten waren 100 Punkte zu erreichen. Zum Bestehen sind mindestens die Hälfte der erreichbaren Punkte zu erzielen. Die Punktergebnisse wurden wie folgt in die üblichen Noten umgerechnet:

- $\geq 95\% \equiv$ Note: $1,0$
- $\geq 92\% \equiv$ Note: $1,3$
- $\geq 89\% \equiv$ Note: $1,7$
- $\geq 80\% \equiv$ Note: $2,0$
- $\geq 77\% \equiv$ Note: $2,3$
- $\geq 74\% \equiv$ Note: $2,7$
- $\geq 65\% \equiv$ Note: $3,0$
- $\geq 62\% \equiv$ Note: $3,3$
- $\geq 59\% \equiv$ Note: $3,7$
- $\geq 50\% \equiv$ Note: $4,0$
- $< 50\% \equiv$ Note: nicht bestanden

Musterlösungen zu den Aufgaben finden Sie im Abschnitt B S. 207 ff. Eine dort notierte Lösung schließt andere, ebenfalls richtige Lösungen, natürlich nicht aus :-).



Legende:
UML ≡ Unified Modeling Language

Abbildung A.1: UML Klassensymbol

A.1 UML-Klassensymbol programmieren

Programmieren Sie in PostScript das Klassensymbol von UML (*Unified Modeling Language*) und zwar in der Form wie es die Abbildung A.1 S. 194 zeigt.

A.2 PostScript-Kontur erläutern

Zeichnen Sie die Graphik, die durch den folgenden PostScript-Quellcode beschrieben ist.

PostScript-Quellcode `halfcirclePS`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Halber Kreis
%%CreationDate: 09-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
/radius 4 cm def
%%EndProlog
5.0 cm 1.0 cm translate
0 cm  0 cm radius 0 180 arc
closepath
gsave
```

```
0.8 setgray
fill
grestore
0.6 setgray
1.4 cm setlinewidth
stroke
0 cm  0 cm radius 2 div 0 180 arc
closepath
0.1 setgray
fill
showpage
%%EOF
```

A.3 PostScript-Quellcode interpretieren

```
 1  %!PS-Adobe-3.0
 2  %%Creator: Emil Cody
 3  %%Title: Nameless
 4  %%CreationDate: 23-Jan-2006
 5  %%EndComments
 6  %%BeginProlog
 7  /cm { 28.35 mul } def
 8  %%EndProlog
 9  1 cm 3 cm moveto
10  7 cm 3 cm lineto
11  1 cm 2 cm moveto
12  7 cm 2 cm lineto
13  1 cm 1 cm moveto
14  1 cm 4 cm lineto
15  7 cm 4 cm lineto
16  7 cm 1 cm lineto
17  closepath
18  stroke
19  /Times-Roman findfont
20  14 scalefont
21  setfont
22  1.1 cm 1.1 cm moveto
23  (getSlot() : String) show
24  1.1 cm 2.1 cm moveto
25  (slot : String="OK!") show
26  1.1 cm 3.1 cm moveto
```

```
27  (de.unilueneburg.as::Person) show
28  showpage
29  %%EOF
30
```

A.3.1 Graphik zeichnen

Der obige Quellcode in PostScript stellt eine Graphik dar. Zeichnen Sie diese Graphik (†15P).

A.3.2 Graphik klassifizieren

Kommt Ihnen diese Graphik bekannt vor? Wenn ja, dann geben Sie an um welche Notation und um welches Symbol es sich handelt (†15P).

A.4 Java3D-Programm erläutern

Java3D-Quellcode `MyCylinder.java`

```
 1  /**
 2   *  "Java 3D Example"
 3   *
 4   *@author     Emil Cody
 5   *@version    1.0
 6   */
 7  package de.unilueneburg.as.figure3D;
 8
 9  import java.applet.Applet;
10  import java.awt.BorderLayout;
11  import java.awt.GraphicsConfiguration;
12  import com.sun.j3d.utils.applet.MainFrame;
13  import com.sun.j3d.utils.geometry.Primitive;
14  import com.sun.j3d.utils.geometry.Cylinder;
15  import com.sun.j3d.utils.universe.SimpleUniverse;
16  import javax.media.j3d.Appearance;
17  import javax.media.j3d.BranchGroup;
18  import javax.media.j3d.Canvas3D;
19  import javax.media.j3d.TransformGroup;
20  import javax.media.j3d.PolygonAttributes;
21
22
23  public class MyCylinder extends Applet
24  {
```

```
25      public SimpleUniverse u = null;
26
27      public BranchGroup createSceneGraph()
28      {
29          BranchGroup bg = new BranchGroup();
30
31          TransformGroup tg = new TransformGroup();
32
33          PolygonAttributes polyAtt =
34              new PolygonAttributes();
35          polyAtt.setPolygonMode(
36              PolygonAttributes.POLYGON_LINE);
37          polyAtt.setCullFace(
38              PolygonAttributes.CULL_NONE);
39
40          Appearance app = new Appearance();
41          app.setPolygonAttributes(polyAtt);
42
43          tg.addChild(new Cylinder(0.3f, 0.9f,
44                       Primitive.GENERATE_NORMALS,
45                       8,
46                       3,
47                       app));
48
49          bg.addChild(tg);
50          bg.compile();
51          return bg;
52      }
53
54      public void init()
55      {
56          this.setLayout(new BorderLayout());
57          GraphicsConfiguration config =
58              SimpleUniverse.getPreferredConfiguration();
59          Canvas3D c = new Canvas3D(config);
60          this.add("Center", c);
61          u = new SimpleUniverse(c);
62          u.getViewingPlatform().
63              setNominalViewingTransform();
64          u.addBranchGraph(
65                           this.createSceneGraph());
66      }
```

```
67
68      public void destroy()
69      {
70          u.cleanup();
71      }
72
73      public static void main(String[] args)
74      {
75          new MainFrame(
76                          new MyCylinder(),
77                          400, 400);
78      }
79  }
80
```

Log-Datei `MyCylinder.log`

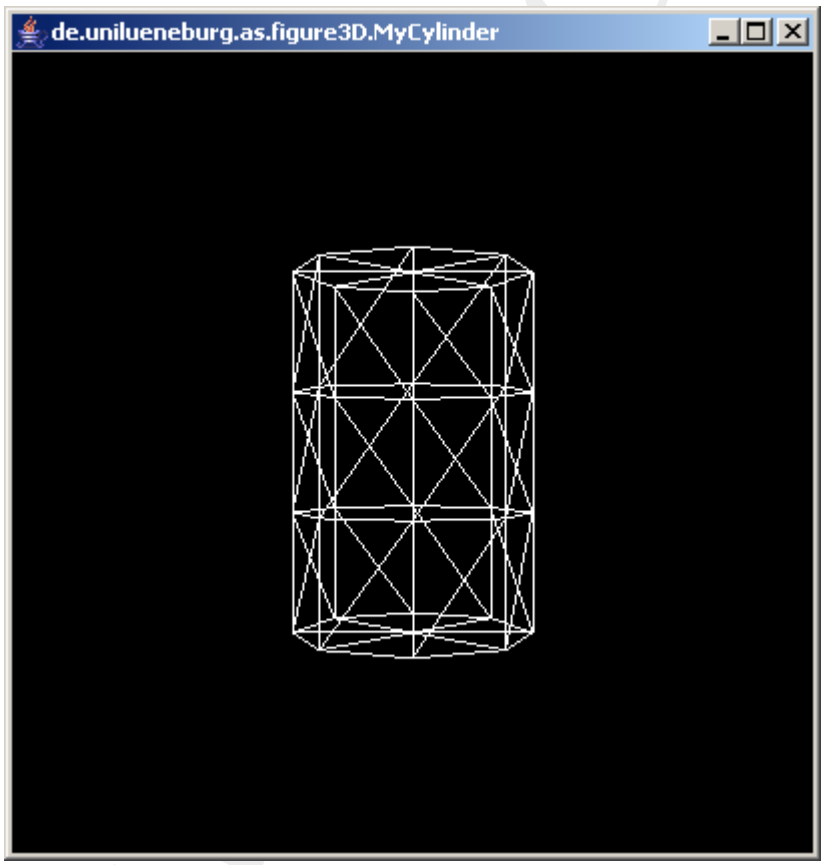
```
1  >java -version
2  java version "1.5.0_04"
3  Java(TM) 2 Runtime Environment, Standard Edition
4    (build 1.5.0_04-b05)
5  Java HotSpot(TM) Client VM
6    (build 1.5.0_04-b05, mixed mode)
7  >javac de/unilueneburg/as/figure3D/MyCylinder.java
8  >java de.unilueneburg.as.figure3D.MyCylinder
9
```

A.4.1 Konstruktor erläutern

In der Quellcodedatei `MyCylinder.java` (↪ Seite 196) wird der Konstruktor `MyCylinder()` appliziert. Ist dieser Konstruktor in der Quellcodedatei `MyCylinder.java` explizit deklariert? Wenn ja, geben Sie die Zeilennummer an. Wenn nein, dann erläutern Sie, warum das Programm trotzdem fehlerfrei (↪ Protokolldatei Seite 198) die Graphik (↪ Abbildung A.2 Seite 199) erzeugt (†10P).

A.4.2 Parameteränderung erläutern

Erläutern Sie die Änderung in der Graphik, wenn in Zeile 46 von `MyCylinder.java` (↪ Seite 196) der Wert von 3 auf 5 gesetzt wird (†10P). Hinweis: Die Compilation und die Applikation sind mit diesem Wert erfolgreich.



Legende:
Java3D-Quellcode `MyCylinder.java` ↪ Seite 196
Protokolldatei `MyCylinder.log` ↪ Seite 198

Abbildung A.2: Java3D-Graphik

A.5 Objekt & Referenz

Java-Quellcode `Customer.java`

```
 1  /**
 2   *  "Example Customer"
 3   *
 4   *@author     Emil Cody
 5   *@version    1.0
 6   */
 7
 8  public class Customer
 9  {
10      int id;
11      String name;
12
13      public String getName()
14      {
15          return name;
16      }
17
18      public Customer setName(String name)
19      {
20          this.name = name;
21          return this;
22      }
23
24      public Customer(int id, String name)
25      {
26          this.id = id;
27          this.setName(name);
28      }
29
30      public static void main(String[] args)
31      {
32          Customer c1 = new Customer(1, "Mustermann");
33          Customer c2 = c1.setName("Musterfrau");
34
35          System.out.println("c1: " + c1.getName());
36          System.out.println("c2: " + c2.getName());
37
38      }
39  }
```

```
40
```

Log-Datei `Customer.log`

```
 1  >java -version
 2  java version "1.5.0_04"
 3  Java(TM) 2 Runtime Environment, Standard Edition
 4    (build 1.5.0_04-b05)
 5  Java HotSpot(TM) Client VM
 6    (build 1.5.0_04-b05, mixed mode)
 7  >javac Customer.java
 8  >java Customer
 9  c1: Musterfrau
10  c2: Musterfrau
11  >
12
```

A.5.1 Kopieproblem erläutern

Der geniale (?) Programmierer *Emil Cody* hat in der `main`-Methode seiner Java-Applikation `Customer.java` (↪ Seite 200) erfolgreich ein Objekt der Klasse `Customer` erzeugt. Er ist der Ansicht, dass er von diesem Objekt `c1` mittels Gleichheitszeichen die Kopie `c2` erzeugen kann. Er wundert sich allerdings über das für ihn überraschende Ergebnis (↪ Protokolldatei Seite 201).

Erläutern Sie, warum in Zeile 33 keine Objektkopie entsteht. Geben Sie eine Korrektur für diese Zeile an, so dass ein zweites Objekt mit der Referenz `c2` entsteht (†10P).

A.5.2 Einhaltung von Notationsregeln prüfen

Üblicherweise haben sogenannte „*Setter*“ (also `set`-Methoden) eine Signatur ohne Rückgabewert, also mit der Angabe `void`. Prüfen Sie, ob in dieser Hinsicht die Java-Applikation `Customer.java` (↪ Seite 200) ordnungsgemäß codiert ist. Wenn nicht, formulieren Sie Ihre Korrektur unter Angabe der betroffenen Zeilennummern (†10P).

A.6 Interface & Inheritance

Java-Quellcode `Baz.java`

```
 1  /**
 2   *  "Example Inheritance"
 3   *
 4   *@author     Emil Cody
 5   *@version    1.0
 6   */
 7
 8  public interface Baz
 9  {
10      public String getSlot();
11      public void setSlot(String slot);
12  }
13
```

Java-Quellcode `Foo.java`

```
 1  /**
 2   *  "Example Inheritance"
 3   *
 4   *@author     Emil Cody
 5   *@version    1.0
 6   */
 7
 8  public class Foo
 9  {
10      protected String slot ="";
11      public static String global ="ENGLAND";
12
13  }
14
```

Java-Quellcode `Bar.java`

```
 1  /**
 2   *  "Example Inheritance"
 3   *
 4   *@author     Emil Cody
 5   *@version    1.0
 6   */
 7
 8  public class Bar extends Foo implements Baz
 9  {
10      private String id;
11
```

```
12      public String getId()
13      {
14          return id;
15      }
16
17      public void setId(String id)
18      {
19          this.id = id;
20      }
21
22      public String getSlot()
23      {
24          return slot;
25      }
26
27      public void setSlot(String slot)
28      {
29          this.slot = slot;
30      }
31
32      public static void main(String[] args)
33      {
34          Bar b = new Bar();
35          b.setId("007");
36          b.setSlot(global);
37          System.out.println("id:   " + b.getId());
38          System.out.println("slot: " + b.getSlot());
39
40      }
41  }
42
```

Log-Datei `Bar.log`

```
1  >java -version
2  java version "1.5.0_04"
3  Java(TM) 2 Runtime Environment, Standard Edition
4    (build 1.5.0_04-b05)
5  Java HotSpot(TM) Client VM
6    (build 1.5.0_04-b05, mixed mode)
7  >javac Baz.java
8  >javac Foo.java
9  >javac Bar.java
```

```
10  >java Bar
11  id:   007
12  slot: ENGLAND
13
```

A.6.1 Interface implementieren

Die Java-Quellcodedatei `Baz.java` (↪ Seite 201) beschreibt ein Interface. In der Java-Quellcodedatei `Bar.java` (↪ Seite 202) wird dieses implementiert. Erläutern Sie welche Teile in `Bar.java` konkret die Vorgaben des Interfaces implementieren. Geben Sie dabei die betroffenen Zeilennummern an (†10P).

A.6.2 Zugriff auf Klassenvariable

In der Klasse `Foo` ist eine Klassenvariable angegeben. Nennen Sie diese und erläutern Sie, warum ohne Angabe der Klasse `Foo` innerhalb der Klasse `Bar` darauf zugegriffen werden kann (†10P).

A.7 Lokale Klasse

Java-Quellcode `Think.java`

```
 1  /**
 2   *  "Example Think"
 3   *
 4   *@author     Emil Cody
 5   *@version    1.0
 6   */
 7
 8  public class Think
 9  {
10      private static final String slot = "outside";
11
12      String getSlot()
13      {
14          return slot;
15      }
16
```

```
17      String think()
18      {
19         class ThinkInside
20          {
21              String slot = "inside";
22
23              String getSlot()
24              {
25                  return slot;
26              }
27          }
28
29         return (new ThinkInside()).getSlot();
30      }
31
32      public static void main(String[] args)
33      {
34
35          System.out.println(slot +
36                             " > " +
37                             (new Think()).think() +
38                             " > " +
39                             (new Think()).getSlot());
40      }
41  }
42
```

A.7.1 Lokale Klasse erläutern

Die Java-Quellcodedatei `Think.java` (↪ Seite 204) enthält eine lokale Klasse bezogen auf eine Methode. Nennen Sie die Methode und geben sie die betroffenen Zeilennummern an (†5P).

A.7.2 Ergebnis der Java-Applikation `Think`

Der Java-Quellcode `Think.java` wurde erfolgreich compiliert. Geben Sie exakt die Ausgabe auf der Console an, wenn die Klasse `Think` ausgeführt wurde, also folgendes Kommando abgearbeitet wurde:

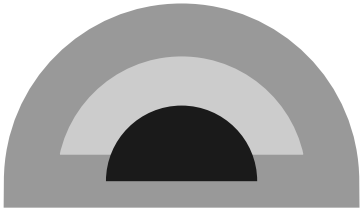
```
>java Think
```
(†5P).

Anhang B

Lösungen zu den Übungen

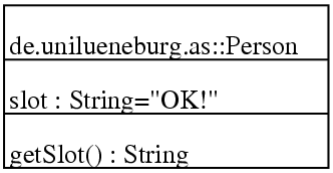
Lösung Aufgabe A.1 S. 194:

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: UML Class Symbol
%%CreationDate: 08-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
1 cm 3 cm moveto
7 cm 3 cm lineto
1 cm 2 cm moveto
7 cm 2 cm lineto
1 cm 1 cm moveto
1 cm 4 cm lineto
7 cm 4 cm lineto
7 cm 1 cm lineto
closepath
stroke
/Times-Roman findfont
16 scalefont
setfont
1.1 cm 3.1 cm moveto
(de.uni-lueneburg.as::FOO) show
showpage
%%EOF
```



Legende:
PostSript-Quellcode ↪ S. 194

Abbildung B.1: Zwei sich überdeckende Halbkreise



Legende:
PostSript-Quellcode ↪ S. 195

Abbildung B.2: UML Klassensymbol

Lösung Aufgabe A.2 S. 194:
Der PostScript-Quellcode stellt zwei sich überdeckende Halbkreise dar, wobei die Kontur des äußeren Kreises mit ein Strichstärke von $1,4cm$ erzeugt wird (↪ Abbildung B.1 S. 208).

Lösung Aufgabe A.3 S. 195:

Lösung Aufgabe A.3.1 S. 196:
Das Ergebnis des PostSript-Quellcodes (↪ S. 195) zeigt Abbildung B.2 S. 208.

Lösung Aufgabe A.3.2 S. 196:
Die Abbildung B.2 S. 208 zeigt ein Klassensymbol in der Notation der

Unified Modeling Language (UML).

Lösung Aufgabe A.4 S. 196:

Lösung Aufgabe A.4.1 S. 198:
In der Quellcodedatei `MyCylinder.java` (↪ Seite 196) ist kein Konstruktor `MyCylinder()` deklariert.

Das Programm erzeugt fehlerfrei (↪ Protokolldatei Seite 198) die Graphik (↪ Abbildung A.2 Seite 199) weil der Standardkonstruktor `MyCylinder()` in Java implizit deklariert ist.

Hinweis: Der implizit deklarierte Standardkonstruktur lässt sich explizit überschreiben:

```
public MyCylinder()
{
}
```

Lösung Aufgabe A.4.2 S. 198:
Der Zylinder wird in der vertikalen Einteilung in 5 Abschnitte geteilt, wenn in Zeile 46 von `MyCylinder.java` (↪ Seite 196) der Wert von 3 auf 5 gesetzt wird. Die Größe des Zylinders ändert sich nicht.

Lösung Aufgabe A.5 S. 200:

Lösung Aufgabe A.5.1 S. 201:
In der Zeile 33 „`c2 = c1.setName(...)`“ referenziert das Objekt `c2` das veränderte Objekt `c1`, da die Methode `setName(...)` als Rückgabewert dasjenige Objekt hat, auf das sie angewendet wurde, also hier `c1`.

Ein zweites Objekt mit der Referenz `c2` entsteht durch Applikation des Konstruktors der Klasse `Customer`, also durch die neue Zeile 33:

```
Customer c2 = new Customer(2, "Musterfrau");
```

Lösung Aufgabe A.5.2 S. 201:
Die Methode `setName(...)` von Zeile 18 bis Zeile 22 entspricht nicht dem Java-Standard einer *Setter*-Methode. Folgende Korrektur ist erforderlich:

```
public void setName(String name)
{
    this.name = name;
}
```

Lösung Aufgabe A.6 S. 201:

Lösung Aufgabe A.6.1 S. 204:
Die Java-Quellcodedatei `Baz.java` (↪ Seite 201) beschreibt ein Interface. In der Java-Quellcodedatei `Bar.java` (↪ Seite 202) wird dieses Interface mit den beiden Methoden `getSlot()` und `setSlot(...)` in den Zeilen 22 – 30 implementiert.

Lösung Aufgabe A.6.2 S. 204:
In der Klasse `Foo` ist die Klassenvariable `global` angegeben. Auf `global` kann ohne Angabe der Klasse `Foo` innerhalb der Klasse `Bar` zugegriffen werden, weil `Bar` eine Unterklasse von `Foo` ist.

Hinweis: Die Klassenvariable `global` muss für diesen Zugriff nicht den Modifier `public` aufweisen. Mit dem Modifier `protected` wäre dieser Zugriff ebenfalls möglich, nicht jedoch mit `private`.

Lösung Aufgabe A.7 S. 204:

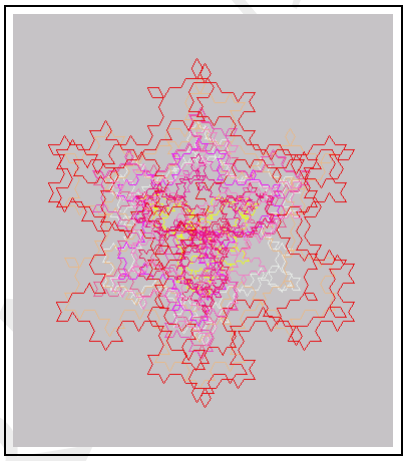
Lösung Aufgabe A.7.1 S. 205:
Die Java-Quellcodedatei `Think.java` (↪ Seite 204) enthält die lokale Klasse `ThinkInside` bezogen auf die Methode `think()`, notiert in den Zeilen 17–30.

Lösung Aufgabe A.7.2 S. 205:
Die Ausgabe auf der Console ist:

```
outside > inside > outside
```

Anhang C

Quellen



C.1 Literaturverzeichnis

Literaturverzeichnis

[Adobe99] Adobe Systems Incorporated; PostScript^{TM} — LANGUAGE REFERENCE,third edition, includes bibliographical references and index, ISBN 0-201-37922-8.

[Arnold/Gosling96] Ken Arnold / James Gosling; The Java Programming Language (Addison-Wesley) 1996.

[Alur/Crupi/Malks01] Deepak Alur / John Crupi / Dan Malks; Core J2EE Patterns, Sun Microsystems Press, Prentice Hall PTR, 2001, in deutsch von Frank Langenau; Core J2EE Patterns — Die besten Praxislösungen und Design-Strategien, 2002, ISBN 3-8272-6313-1.

[Bonin91b] Hinrich E. G. Bonin; Software-Konstruktion mit LISP, Berlin New York (Walter de Gruyter), 1991.

[Bonin92a] Hinrich E. G. Bonin; Arbeitstechniken für die Softwareentwicklung, (3. überarbeitete Auflage Februar 1994), FINAL, 2. Jahrgang Heft 2, 10. September 1992, [FINAL].

[Bonin04a] Hinrich E. G. Bonin; Aspect-oriented Softwaredevelopement — A Little Guidance to Better Java Applications —, aktuelle Fassung unter: `http://as.uni-lueneburg.de.publikation.aosdall.pdf` (online 21-Mar-2004), begonnen 26-Jan-2002. {Hinweis: Beispiele primär in `AspectJ`.“}

[Bonin04b] Hinrich E. G. Bonin; Der Java^{TM}-Coach — Modellieren mit UML, Programmieren mit Java^{TM} 2 Plattform (J2SE & J2EE), Dokumentieren mit XHTML —, aktuelle Fassung unter: `http://as.uni-lueneburg.de.publikation.anwdall.pdf` (online 21-Mar-2004), begonnen 5-Oct-1997. {Hinweis: Eine umfassende Einführung in die Objektorientierung auf der Basis von Java^{TM}.}

[Deussen03] Oliver Deussen; Computergenerierte Pflanzen — Technik und Design digitaler Pflanzenwelten, Berlin u. a. (Springer Verlag), ISBN 3-540-43606-5. {Hinweis: Schöne Bilder mit fundierter Analyse von vielfältigen Pflanzenstrukturen.}

[Eckel02] Bruce Eckel; Thinking in Java — The Definitive Introduction to Object-Oriented Programming in the Language of the World-Wide-Web, Upper Saddle River, NJ 07458 (Prentice Hall PTR), 3rd Edition, ISBN 0-13-100287-2.

[Flanagan97] David Flanagan; Java in a Nutshell, Second Edition, updated for Java 1.1, Köln (O´Reilly), May 1997.

[FINAL] Fachhochschule Nordostniedersachsen, Informatik, Arbeitsberichte, Lüneburg (FINAL) herausgegeben von Hinrich E. G. Bonin, ISSN 0939-8821, ab 7. Jahrgang (1997) auf CD-ROM, beziehbar: FH NON, Volgershall 1, D-21339 Lüneburg, Germany.

[Glover03] Dan Glover (Compiled by); Lila's Child — An Inquiry into Quality; with introduction and annotations by Robert M. Pirsig, (1stBooks Library) 2003, ISBN 1-4033-5620-3. {Remark: Lila's Child chronicles: Internet discussion group centered around Robert M. Pirsig's novel.}

[Horstmann05a] Cay S. Horstmann / Gary Cornell; Core Java^{TM} 2, Volume I — Fundamentals, seventh edition, Sun Microsystems Press, a Prentice Hall Title, 2005, ISBN 0-13-148202-5. {Remark: "*A non-nonsens tutorial and reliable reference, this book features thoroughly tested real-world examples.*"}

[Horstmann05b] Cay S. Horstmann / Gary Cornell; Core Java^{TM} 2, Volume II — Advanced Features, seventh edition, Sun Microsystems Press, a Prentice Hall Title, 2005, ISBN 0-13-111826-9. {Remark: "*Revised and updated coverage of multithreading, collections, database programming, distributed computing and XML.*"}

[JavaSpec] James Gosling / Bill Joy / Guy Steele; The Java Language Specification, (Addison-Wesley) 1996;
`http://www.javasoft.com/docs/books/jls/html/index.html`
Änderungen für Java 1.1;
`http://www.javasoft.com/docs/books/jls/html/1.1Update.html`
(Zugriff: 20-Sep-1997)

[Mäckler00] Andreas Mäckler (Hrsg.); 1460 Antworten auf die Frage: Was ist Kunst? — Neuausgabe — Köln (DuMont), 2000, ISBN 3-7701-5420-7. {Hinweis: „Künstler nutzen die Zitate für Aktionen."}

[McGilton/Campione92] Henry McGilton / Mary Campione; PostScript by Example, Reading Massachusetts u. a. (Addison-Wesley) 1992, ISBN 0-201-632286-4.

[Nake03] Frieder Nake, space.color — Raum. Algorithmus. Farbe, in: [Rödiger03] S. 135–140. {Hinweis: „sätze, vorgetragen zur eröffnung der ausstellung von manfred mohr im museum für konkrete kunst in ingolstadt am 14. oktober 2001". (Im Orginal in kleinen Buchstaben geschrieben)}

[Nadin03] Mihai Nadin; Das Interessante als computationale Zielsetzung, in: [Rödiger03] S. 99–133. {Hinweis: Überarbeiteter Vortrag vom 16. Dezember 1998.}

[Rödiger03] Karl-Heinz Rödiger (Hrsg.); Algorithmik — Kunst — Semiotik, Hommage für Frieder Nake, Heidelberg (Synchron Wissenschaftsverlag) 2003, ISBN 3-935025-60. {Hinweis: Festschrift für Frieder Nake, einer der großen Pioniere der Computergraphik.}

[Schader+03] Martin Schader / Lars Schmidt-Thieme; Java — Eine Einführung, Berlin Heidelberg (Springer), 4. Auflage 2003, ISBN 3-540-00663-X. {Hinweis: Das

Buch enthält gelungene Übungen mit Lösungen (auf der beigefügten CD-ROM).}

[Selman02] Daniel Selman; Java3D Programming, Greenwich CT 06830 (Manning Publications Co.), ISBN 1-930110-35-9. {Hinweis: "*Java 3D Programming* is a roadmap for application developers."}

[Shavor+03] Sherry Shavor / Jim D'Anjou / Scott Fairbrother / Dan Kehn / John Kellerman / Pat McCarty; The Java^{TM} Developer's Guide to Eclipse, Boston u. a. (Addison-Wesley), ISBN 0-321-15964-0. {Hinweis: "This Book does an excellent job of helping you learn Eclipse."}

[Ware04] Colin Ware; Information Visualization — Perception for Design, Amsterdam (Morgan Kaufmann / Elsevier) 2004, ISBN 1-55860-819-2. {Remark: "This book combines a strictly scientific approach to human perception with a practical concern for the rules governing the effective visual presentation of information."}

C.2 Web-Quellen

Java3D-Sun-Material
`http://java.sun.com/products/java-media/3D/`

Java3D-Material
`http://java3d.virtualworlds.de/`

J2SE-SDK-Dokumentation
`http://java.sun.com/docs/index.html`

Zur Geschichte von Java^{TM}
`http://java.sun.com/nav/whatis/storyofjava.html`

Anhang D

Hinweise zu Faszination Programmierung

D.1 Werkzeuge zum Manuskript

Mein Web-Server: `http://as.uni-lueneburg.de/`
Unter diesem Web-Server werden weitere Informationen zu diesem Buch angeboten.

Mit folgender Software wurde Faszination Programmierung erstellt:

Editor: GNU Emacs 21.2.1 (2002-03-19); jEdit 4.1 final

Layout: TeX, Version 3.14159 (Web2c 7.3.7x), LaTeX2e <2000/06/01>; Document Class: book 2001/04/21 v1.4e Standard LaTeX document class

Hardcopy: Corel CAPTURE 11; Corel PHOTO-PAINT 11 (version 10.427)

Figure: Microsoft Visio 2000 SR1 (6.0.2072)

Index: makeindex, version 2.13 [07-Mar-1997] (using kpathsea)

DVI→PS: LaTeX-File (Device Independent) to Postscript: dvips(k) 5.90a Copyright 2002 Radical Eye Software (www.radicaleye.com)

PS→PDF: Postscript file to PDF-File: Adobe Acrobat Distiller 7.0 Professional

Security: Adobe Acrobat 7.0 Professional (Version 5.0)

D.2 Liste der Fonts

PostScript-Quellcode `fontsPS`

```
Fa
FR
Fd
Ff
FJ
FW
CMMI9
FN
Fc
FE
FH
CMMI10
CMMI6
CMR12
CMR8
FG
FI
CMMI12
Times-Bold
Courier-Bold
Helvetica
Fe
CMMI8
Times-Roman
Fy
Fm
CMR10
Fr
Fb
CMR9
Fu
FM
FO
Fp
FB
FT
Fv
Fh
FK
FX
FL
Fq
FC
FV
Fw
Fi
Fo
FS
Ft
Courier
CMSY10
Fn
FQ
FD
FU
FF
Fg
Fj
Fz
Fl
FA
CMSY9
Times-Italic
Fs
Fx
Fk
FP
```

Legende:

DVIPS(k) 5.90a Copyright 2002 Radical Eye Software ↪ `http://www.radicaleye.com`

Die Idee für dieses Beispiel entnommen aus [McGilton/Campione92] p. 554; PostScript-Quellcode ↪ Seite 217.

Abbildung D.1: Liste der Fonts von `DVIPS`

Legende:

CorelDraw12 Version 12.0.0.458 Copyright 2003

Die Idee für dieses Beispiel entnommen aus [McGilton/Campione92] p. 554; PostScript-Quellcode ↪ Seite 217.

Abbildung D.2: Liste der Fonts von `CorelDraw12`

```
%!PS-Adobe-3.0
%%Creator: McGilton-Campione-1992, p.554 (Bonin)
%%Title: List of fonts
%%CreationDate: 07-Oct-2005
%%EndComments
%%BeginProlog
/inch { 72 mul } def
%%EndProlog
/PrintSize 12 def
/Leading 14 def
/TopMargin 10.5 inch def
/BottomTopMargin 0.5 inch def
/LeftMargin 0.5 inch def
/Courier-Bold findfont
PrintSize scalefont
setfont
/whichSide 0 def
/BaseLine TopMargin def
/JunkString 256 string def
/ShowFontName {
  JunkString cvs
  LeftMargin whichSide 4.25 inch mul add BaseLine moveto
  show
  /whichSide 1 whichSide sub def
  whichSide 0 eq {
      /BaseLine BaseLine Leading sub def
      BaseLine BottomTopMargin lt {
          showpage
          /BaseLine TopMargin def
      } if
  } if
} def
FontDirectory
{
    pop
    ShowFontName
} forall
showpage
%%EOF
```

Anhang E

Glossar

Alpha-Kanal Ein Bild wird üblicherweise über drei Farbwerte beschrieben: Rot, Grün und Blau (RGB) — beim Drucken : Gelb (*yellow*), Magenta (*magenta*) und Zyan (*cyan*). Zusätzlich wird noch eine Tranzparenzangabe benötigt. Sie gibt pro Bildpunkt den Grad der Durchsichtigkeit an. Diesen Tansparenzkanal bezeichnet man als *Alpha*-Kanal.

Appearance Aussehen, äußerer Schein

Bump-Mapping Mit einem Foto, gelegt auf die Objektoberfläche, erhöht man ohne großen Aufwand den Realismus des Objektes (↪ Textur 222. Mit einem Bump-Mapping verändert man die Richtung der Oberflächennormalen. Beispielsweise ist man damit in der Lage, eine gewellte Oberfläche vorzutäuschen.

Branch Kante, Abzweigung

Keyframing Eine Animation wird durch prägende Teilszenen spezifiziert. Die dazwischenliegenden Teile werden durch Interpolation aller Bildwerte berechnet.

Level-of-Detail (LOD) Die Objektdarstellung wird, je nach der visuellen Größe auf dem Bildschirm, in ihrer Komplexität verändert. Ein entfernt befindliches Objekt besteht aus wenigen Daten, ein nahes aus vielen.

inward inner, curve inward ≡ Kurve nach innen

Polygon beschreibt eine Fläche im Raum, zum Beispiel durch ein Dreieck. Ein allgemeines Polygon kann eine beliebige Eckenanzahl haben. Als Polyeder bezeichnet man das über die Eckpunkte und Seitenflächen beschriebene Volumenelement.

Polyeder Volumenelement ↪ Polygon 222

Rendering ≈ Umwandlung, Transformation, Wiedergabe, (künstlerische) Interpretation. Die Eingabe in Form von Geometrie- und Beleuchtungsdaten wird in ein betrachtbares Bild „umgewandelt".

Sphere Einflussbereich, Kugel

Textur ist ein Bild zur Projektion auf die Objektoberfläche, um das Objekt realistischer erscheinen zu lassen. Bei der Projektion ist es notwendig, jedem Bildpunkt der Textur einen Alpha-Wert (↪ Alpha-Kanal 221) mitzugeben. Er spezifiziert die Durchsichtigkeit des Bildpunktes.

Trigger auslösendes Ereignis, Auslöseimpuls

Vertex pl. *vertices* Scheitel(punkt), Spitze

Anhang F

Abkürzungen und Akronyme

API	Application programming interface
AWT	Abstract Windowing Toolkit
DAG	Directed acyclic graph
FHNON	Fachhochschule Nordost Niedersachsen
FOV	Fields of view
LOD	Level-of-Detail
UML	Unified Modeling Language

Anhang G

Index

Index

Index

Abstract Windowing Toolkit, 57
Acrobat, 217
    Distiller, 217
`add`, 217
`add()`, 66, 103
`addBranchGraph()`, 66, 75, 87
`addChild()`, 66, 75, 87, 103
Adobe
    Acrobat, 217
        Distiller, 217
Adobe Systems Incorporated, 213
Adobe-3.0, 29
Algorithmus, 21
`ALLOW_TRANSFORM_WRITE`, 87, 103
`aload`, 40
`Alpha`, 87, 103
Alpha-Kanal, 221
Alur, Deepak, 213
Ambient color, 65
Animation, 141
API, 47, 223
Appearance, 53, 55, 221
`Appearance`, 66, 75, 87, 103
Applet, 48
    appletviewer, 53
    Eingabefenster, 53
    Firefox, 54
`Applet`, 66, 75, 87, 103
Arc, 53
`arc`, 41, 42, 194
`args`, 75, 87
Aristoteles, 20
Array Object, 39
Aspect-oriented Softwaredevelopment, 213
AspectJ, 213
`AutomatonR110.eps`, 105
AWT, 57, 223
`AxisAngle4f`, 66, 142

Bach
    Fuge, 23
`Background`, 87, 103
Bauhaus, 19
Befehl
    Notation, 27
`BeginProlog`, 32, 33
`Behavior`, 142
BG, 55
Bill Joy, 214
Billboard, 141
Body
    Physical, 55
`BOLD`, 103
`Bonin`, 104
Bonin, Hinrich E.G., 213, 214
Boolean Value, 39
`BorderLayout`, 66, 103
`BoundingBox`, 31
`BoundingSphere`, 75, 87, 103
Branch, 221
Branch Group Node, 55
`BranchGroup`, 66, 75, 87, 103
`brickwork.eps`, 81
Bump-Mapping, 221

Campione, Mary, 214
Canvas, 57
Canvas3D, 55
`Canvas3D`, 66, 103
`catch()`, 130
`class`, 75, 87
`cleanup()`, 66, 103

clip, 42
closepath, 30–32, 194
cm, 29
/cm, 32, 33
code.eps, 87
Color
    ambient, 65
    diffuse, 65
    emissive, 65
    specular, 65
Color3f, 66, 75, 87, 103
com.sun.j3d.loaders.Incorrect-
        FormatException, 130
com.sun.j3d.loaders.object-
        file.ObjectFile, 130
com.sun.j3d.loaders.Parsing-
        ErrorException, 130
com.sun.j3d.loaders.Scene, 130
com.sun.j3d.utils.applet.-
        MainFrame, 66, 103, 130,
        151, 158
com.sun.j3d.utils.behaviors.-
        keyboard.KeyNaviga-
        torBehavior, 151
com.sun.j3d.utils.behaviors.-
        mouse.MouseRotate, 158
com.sun.j3d.utils.geometry.*,
        87
com.sun.j3d.utils.geometry.Cone,
        158
com.sun.j3d.utils.geometry.-
        Cylinder, 66
com.sun.j3d.utils.geometry.-
        Primitive, 66, 103, 151,
        158
com.sun.j3d.utils.geometry.-
        Primitive, 75
com.sun.j3d.utils.geometry.-
        Sphere, 75, 103, 151, 158
com.sun.j3d.utils.geometry.-
        Text2D, 103
com.sun.j3d.utils.image.Texture-
        Loader, 87, 103
com.sun.j3d.utils.universe.*,
        87
com.sun.j3d.utils.universe.-
        SimpleUniverse, 66, 75,
        103, 130, 151, 158
compile(), 66, 103
Core classes, 57
Corel
    CAPTURE, 217
    PHOTO-PAINT, 217
Cornell, Gary, 214
cos, 40
Courier, 42
createBackground(), 87
createBehaviors(), 75, 87
createSceneGraph(), 66, 75, 87
CreationDate, 30
Creator, 30
Crupi, John, 213
CULL_NONE, 66, 103
currentmatrix, 42
Cyan, 221
Cylinder, 66

DAG, 55, 223
D´Anjou, Jim, 215
DataSharing, 124
Datentyp, 39
de.uni-lueneburg.as.figure3D,
        75, 80, 87, 103
de.unilueneburg.as.figure3D,
        66
DECREASING_ENABLE, 87
def, 32, 33
destroy(), 66, 103
Deussen, Oliver, 213
Dexel, Walter, 19
Diffuse color, 65
DIN A4, 29
DirectionalLight, 75, 87
Distiller
    Acrobat, 217
draw(), 48
drawString(), 48
Drehener Text, 104
dvips, 217

Eckel, Bruce, 213

Emacs
GNU, 217
Emissive color, 65
`EndComments`, 30–33
`EndProlog`, 32, 33
`Enumeration`, 142
Environment
Physical, 55
`EOF`, 30–33
`eq`, 217
`extends`, 75, 87

Fairbrother, Scott, 215
FHNON, 223
`FileNotFoundException`, 130
`fill`, 32, 38, 194
`fill()`, 48
FINAL, 214
`findfont`, 33
Flanagan, David, 214
`float`, 103
`Font.BOLD`, 103
`Font.ITALIC`, 103
`FontDirectory`, 217
`forall`, 217
Forth, 27
FOV, 223
Fuge
Bach, 23

`GENERATE_NORMALS`, 87
`GENERATE_NORMALS`, 66, 75
`GENERATE_NORMALS_INWARD`, 87, 103
`GENERATE_TEXTURE_COORDS`, 87, 103
Geometrie
Konstruierte, 116
Geometry, 55
`get`, 40
`getAdvance()`, 48
`getAppearance()`, 103
`getAscent()`, 48
`getBoundingSphere()`, 75, 87, 103
`getDescent()`, 48
`getFontRenderContext()`, 48
`getParameter()`, 48
`getPolygonAttributes()`, 103
`getPreferredConfiguration()`, 66, 103
`getSceneGroup()`, 130
`getTexture()`, 87, 103
`getViewingPlatform()`, 66, 75, 87, 103
`getViewPlatformTransform()`, 103
Glover, Dan, 214
GNU
Emacs, 217
Gosling, James, 213, 214
`GraphicsConfiguration`, 66, 103
`grestore`, 38, 42, 194
Group, 56
`gsave`, 38, 42, 194

Heap Allocation, 137
JVM, 137
`HelloUniverse`, 59
`HelloWorld`, 71
Horstmann, Cay S., 214
HotSpot
Memory Options, 137

if, 217
`if`, 103
`import`, 75, 87
`inch`, 29
`IncorrectFormatException`, 130
`INCREASING_ENABLE`, 87, 103
Influencing Bounds, 75
`init()`, 66, 103
`initialize()`, 142
Instanz, 53
Integer Value, 39
Interaktion, 141
inward, 221
`ITALIC`, 103

J2SE, 215
Java
1.1
Spezifikation, 214
Historiebericht, 215
klassische Beschreibung, 213
Java-Coach, 213

java.applet.Applet, 66, 75, 87, 103, 151, 158
java.applet.Applet, 48
java.awt.BorderLayout, 66, 75, 103, 151, 158
java.awt.Color, 48
java.awt.event.KeyEvent, 142
java.awt.Font, 103
java.awt.Font, 48
java.awt.font.FontRenderContext, 48
java.awt.font.TextLayout, 48
java.awt.geom.Ellipse2D, 48
java.awt.geom.Rectangle2D, 48
java.awt.Graphics, 48
java.awt.Graphics2D, 48
java.awt.GraphicsConfiguration, 151, 158
java.awt.GraphicsConfiguration, 66, 103
java.util.Enumeration, 142
java.util.Random, 48
Java3D
    Sun-Dokumente, 215
    Web-Dokumente, 215
javax.media.j3d.*, 87, 103
javax.media.j3d.Alpha, 170
javax.media.j3d.Appearance, 66, 75, 130, 151, 158
javax.media.j3d.Background, 130, 151, 158
javax.media.j3d.Behavior, 142
javax.media.j3d.BoundingSphere, 130, 151, 158
javax.media.j3d.BoundingSphere, 75
javax.media.j3d.BranchGroup, 66, 75, 130, 151, 158
javax.media.j3d.Canvas3D, 66, 130, 151, 158
javax.media.j3d.DirectionalLight, 75, 130, 151, 158
javax.media.j3d.GeometryArray, 151, 158
javax.media.j3d.IndexedTriangleArray, 151
javax.media.j3d.LineArray, 151, 168
javax.media.j3d.LineAttributes, 168
javax.media.j3d.LineStripArray, 168
javax.media.j3d.Link, 151
javax.media.j3d.Material, 66, 75, 130, 151, 158
javax.media.j3d.PolygonAttributes, 66
javax.media.j3d.RotationInterpolator, 170
javax.media.j3d.Shape3D, 151, 158
javax.media.j3d.SharedGroup, 151
javax.media.j3d.Transform3D, 66, 130, 151, 158
javax.media.j3d.TransformGroup, 66, 75, 130, 151, 158
javax.media.j3d.WakeupOnAWTEvent, 142
javax.media.j3d.WakeupOnElapsedTime, 142
javax.swing.JOptionPane, 48
javax.vecmath.*, 87, 103
javax.vecmath.AxisAngle4f, 66
javax.vecmath.Color3f, 66, 75, 130, 151, 158
javax.vecmath.Point3d, 75, 130, 151, 158
javax.vecmath.Point3f, 151
javax.vecmath.Vector3f, 75, 130, 151, 158
Jawlensky von, Alexej, 19
jEdit, 217
JVM, 137
    Heap Allocation, 137

Kante, 53
Kehn, Dan, 215
Kellerman, John, 215
Ken, Arnold, 213
KeyEvent.KEY_PRESSED, 142
Keyframing, 221

`KeyNavigatorBehavior`, 151
Knoten, 53
Kommando
    Notation, 27
Kriegskunst, 20

LaTeX, 217
Leaf, 56
Level-of-detail, 221
Licht, 75
Lila's Child, 214
`lineto`, 29, 31, 32
LISP, 213
`load()`, 130
Locale, 55, 56
LOD, 141, 221, 223
`lt`, 217
Łukasiewicz, Jan, 28

Mäckler, Andreas, 214
Magenta, 221
`main()`, 75, 87
`MainFrame`, 66, 103
Malks, Crupi, 213
`Material`, 66, 75, 87
`Math.PI`, 103
`Math.toRadians()`, 142
`matrix`, 42
McCarty, Pat, 215
McGilton, Henry, 214
Microsoft
    Visio, 217
Mohr, Manfred, 214
Moore, Charles H., 27
`MouseRotate`, 158
`MoverBehavior`, 103
`moveto`, 29, 31–33
`mul`, 32, 33, 217
Musikwissenschaft, 23
Muster, 213

Nadin, Mihai, 23, 214
Nake, Frieder, 23, 214
Navigation, 141
`new`, 75, 87
`newpath`, 42
Node, 53
Node Component, 56
Notation, 17–18
    Postfix, 27
    Prefix, 27
`null`, 103

`<object>`, 51
`ObjectFile`, 130

`/Palatino-Roman`, 33
`<param>`, 51
Parent–Child Link, 56
`parseFloat()`, 48
`ParsingErrorException`, 130
Pattern, 213
Physical
    Body, 55
    Environment, 55
`PI`, 103
Pirsig, Robert M., 13, 214
`Point3d`, 75, 87, 103
Polyeder, 222
Polygon, 222
`PolygonAttributes`, 66, 103
`PolygonAttributes.CULL_NONE`, 66
`PolygonAttributes.POLYGON_LINE`, 66
`POLYGON_LINE`, 66
Pop, 36
`pop`, 40, 217
`PositionInterpolator`, 87
PostScript, 25–42
`Primitive`, 75, 87, 103
`Primitive.GENERATE_NORMALS`, 66
`processStimulus()`, 142
Programm
    Begriff, 21
    `PS`, 29
`PS-Adobe-3.0`, 31–33
`public`, 75, 87
Push, 36

Real Value, 39
Relationship, 53
rendering, 222

repeat, 42
return, 75, 87, 103
RGB, 221
Robustheit, 22
Rödiger, Karl-Heinz, 214
rotate, 33
RotationInterpolator, 103

scalefont, 33
Scene, 130
Scene Graph, 53
    Beispiel, 55
    Symbole, 56
Schader, Martin, 215
Scheme, 213
Schmidt-Thieme, Lars, 215
Schriftart
    Typewriter, 17
Screen3D, 55
Selman, Daniel, 215
Semiotik, 21
Serif, 103
setApplicationBounds(), 75, 87
setCapability(), 87, 103
setColor(), 48
setCullFace(), 66, 103
setFont(), 48
setfont, 33
setGeometry(), 103
setgray, 32, 194
setInfluencingBounds(), 75, 87
setLayout(), 66, 103
setLightingEnable(), 66
setlinejoin, 38
setlinewidth, 38, 194
setMaterial(), 66, 75, 87
setmatrix, 42
setNominalViewingTransform(), 66, 75, 87, 103
setPolygonAttributes(), 66
setPolygonMode(), 66
setRotation(), 66
setScale(), 103
setSchedulingBounds(), 87, 103
setTexture(), 87, 103
setTransform(), 66, 103
setTranslation(), 103
Shape3D, 103
Shape3D Node, 55
Shavor, Sherry, 215
Shininess, 65
show, 33
showInputDialog(), 48
showpage, 31–33, 194
SimpleFigure3Da, 76
SimpleFigure3Db, 82, 88
SimpleFigure3Dc, 96
SimpleUniverse, 57, 66, 75, 87, 103
sin, 40
Smiley, 18
Softwareentwicklung
    Arbeitstechniken, 213
Specular color, 65
Speicher
    Heap, 137
Sphere, 222
Sphere, 75, 87, 103
Stack, 36
static, 75, 87
Steele, Guy, 214
Stimulus, 141
String, 39
String, 75, 87
stroke, 30–32, 38, 194
sub, 217
System.err.println(), 130
System.exit(), 130

TEX, 217
Text.html, 51
Text, 48
Text2D, 103
Textur, 222
Texture, 87, 103
TextureLoader(), 87, 103
this, 75, 87
Title, 30
toRadians(), 66
Transform Group Node, 55
Transform3D, 66, 87, 103
TransformGroup, 66, 75, 87, 103
translate, 42, 194

`Transparency`, 98
Trigger, 141, 222
`try`, 130
Tschritter, Norbert, 15
tt EPSF, 31

Umgekehrten Polnischen Notation, 27
UML, 223
Utility classes, 57

`Vector3f`, 75, 87, 103
Vertex, 222
Vertices, 222
View, 55
View Platform, 55
Virtual Universe, 53, 55, 56
Visio
    Microsoft, 217
`void`, 75, 87
Volumenelement, 222

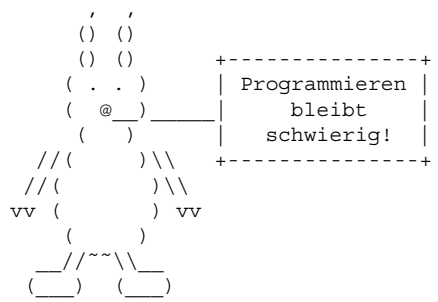
`Wachtel1`, 167
Wagner, Christian, 15
`wakeupOn()`, 142
`WakeupOnAWTEvent`, 142
`WakeupOnElapsedTime`, 142
Ware, Colin, 215

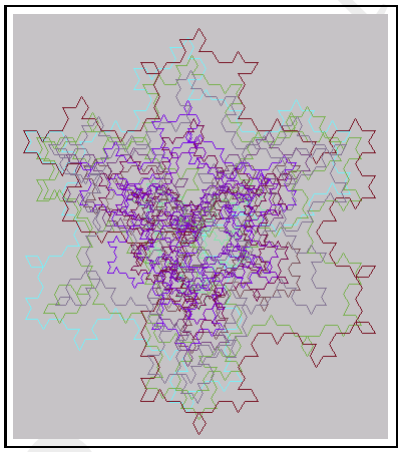
XHTML, 51

YMC, 221
`yPos`, 103

Zyan, 221

Alle Angaben in diesem Buch erfolgen nach bestem Wissen und Gewissen. Sorgfalt bei der Umsetzung ist indes dennoch geboten. Der Verlag, der Autor und die Herausgeber übernehmen keinerlei Haftung für Personen-, Sach- oder Vermögensschäden, die aus der Anwendung der vorgestellten Materialien und Methoden entstehen könnten.





* * *