open source PRESS

Grundlagen der 3D-Programmierung

Mathematik und Praxis mit OpenGL

professional reference

Gerhard Virag


open
source
PRESS

Gerhard Virag

Grundlagen der 3D-Programmierung

Mathematik und Praxis mit OpenGL

Bibliografische Information Der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Gesamtlektorat: Dr. Markus Wirtz
Satz: Open Source Press (LaTeX)
Umschlaggestaltung: Olga Saborov, Open Source Press
ISBN der Print-Ausgabe: 978-3-941841-75-8

ISBN (E-Book, PDF) 978-3-941841-90-1 http://www.opensourcepress.de

Inhaltsverzeichnis

Über dieses Buch

Mit der 3D-Computergrafik zeigt sich die Informatik von ihrer schönsten Seite, besonders in den Königsklassen dieser Disziplin, bei 3D-Filmen und Games, denn dort spielen auch Aspekte des Erzählerischen und der darstellenden Kunst eine Rolle. All dies ist mehr als nur mathematisch-technische Problemlösung, es ist auch eine Annäherung an die Wirklichkeit. Objekte der virtuellen Welt erhalten ein Volumen und sind nicht mehr nur Kulisse. Mit CAD-Systemen am Computer erstellte Objekte – von Mikromaschinen bis zur Raumstation – sind in diesem Sinne sogar „vorweggenommene Wirklichkeit", die erst noch erschaffen werden muss.

Im technischen und wissenschaftlichen Bereich ordnet man den Volumina, abstrakt gesprochen, auch Bewertungen zu. Zum Beispiel repräsentiert ein mit komplexen Verfahren vermessenes Volumen eine bestimmte Gesteinsformation oder ein Gewebe, und die bei solchen Verfahren anfallenden Datenmengen erzwingen geradezu eine visuelle Auswertung mit den Mitteln der 3D-Computergrafik. 3D-Filme, Spiele und andere Produkte der Werbe- und Unterhaltungsindustrie sind also nur die Spitze des Anwendungseisberges, und mit Sicherheit nimmt die Bedeutung der 3D-Computergrafik weiter zu.

Warum dieses Buch?

Mein erster ernsthafter Kontakt mit der 3D-Computergrafik war Anfang der 90er Jahre, als ich auf das Lindenmayer-System aufmerksam wurde. Aristid Lindenmayer, ein ungarischer Biologe, nutzte eine einfache, aber rekursive Grammatik, um damit das Formenwachstum von Pflanzen zu beschreiben.[1] Hinzu kamen die sog. *Turtle Graphics*, ein Schildkröten-Cursor, der sich im 3D-Raum bewegt und dort Objekte (Blätter, Äste o. Ä.) platziert. Aus purer Neugier und Forscherdrang wollte ich herausfinden, welche Formen und Figuren sich mit solchen Produktionsregeln bilden lassen. Dazu musste ich einen Compiler für die Umwandlung eines Regelsat-

[1] Zum sog. „L-System" vgl. Przemyslaw Prusinkiewicz, Aristid Lindenmayer: „The Algorithmic Beauty of Plants", Springer, 1990.

zes und einen 3D-Softwarerenderer für die Turtle Graphics erstellen. Alles, was ich benötigte, waren ein C++-Compiler, ein Editor und ein Bildspeicher für die Ausgabe. 20 Jahre später befinden sich in der Werkzeugkiste erheblich mehr Software-Werkzeuge, einige erfordern sowohl für die Nutzung als auch für die Entwicklung ein hohes Maß an Know-how. Eines dieser Werkzeuge, OpenGL, stelle ich hier zur Demonstration und für die Experimente des Lesers in den Mittelpunkt. Ich werde diese „Black Box“ jedoch nur gelegentlich öffnen, hauptsächlich um die zugrundeliegende Architektur zu beschreiben, weniger der Implementierung wegen.

Der Kern der 3D-Programmierung ist nicht so sehr bei den einzelnen Werkzeugen zu suchen, vielmehr im Konzeptionellen und der Mathematik. Auch aus diesem Grunde blicke ich nach oben, hin zu höheren Software-Schichten, Middleware wie Game Engines und anderen Anwendungen, aber auch in die Geschichte, auf Personen, die die Entwicklung der Technologie maßgeblich beeinflusst haben.

Ein solches Buch ist immer ein Balanceakt, jedenfalls wenn beide Stars, Praxis *und* Theorie, zur Vorstellung kommen. Natürlich muss ein Softwareentwickler nicht in der Theorie des Compilerbaus bewandert sein. Wenn er aber die Meldungen seines Compilers nicht interpretieren kann, vermag er auch nicht mit den Konzepten der gewählten Sprache darauf zu reagieren. Das bedeutet in unserem Falle konkret: Man muss OpenGL verstehen, um mit „Sprachelementen“ wie dem dreidimensionalen Raum, Licht, Oberflächen und deren Farben zu gestalten.

Das vorliegende Buch, das sich in erster Linie an ambitionierte Einsteiger richtet, zeigt, wie man mit diesen Sprachelementen Programme schreibt, die virtuelle Welten am Bildschirm darstellen. Hier schließt sich der Kreis, denn es geht auch um die Freude des Entwicklers, aus dem Abstrakten konkrete, sichtbare Ergebnisse zu schaffen, die im Falle der 3D-Computergrafik auch eine ästhetische, ja spielerische Dimension haben.

Kapitelübersicht

In Kapitel 1, der *Einführung*, werfen wir einen genaueren Blick in unseren Werkzeugkasten und klären Grundbegriffe.

Das Kapitel 2, *Der dreidimensionale Raum* (Seite 37 ff.), stellt die in der 3D-Computergrafik so bedeutende 4×4-Transformationsmatrix in den Mittelpunkt. Diese Matrix ist ein universelles Werkzeug, und zwar für alle Arten von Transformationen, die entlang der Rendering Pipeline auftreten, fundamental für die Platzierung von Objekten in der virtuellen Welt und letztlich der Projektion auf den zweidimensionalen Bildschirm. Es werden einzelne Ausprägungen der Matrix präsentiert, wie etwa die Rotation um einen Drehvektor sowie die verschiedenen Arten der Projektionsmatrizen.

Das Kapitel befasst sich auch mit dem Darstellungsbereich, dem Tiefenpuffer und der Art und Weise, wie OpenGL eine virtuelle Welt abbildet.

Das Kapitel 3, *Der vernetzte Raum* (Seite 137 ff.), befasst sich mit weiteren Grundelementen der 3D-Computergrafik. Knoten und Kanten sind Begriffe der Graphentheorie, einem schon seit Jahrhunderten beackerten Feld der Mathematik. Für die Navigation bedeutend sind die Pfeile, die von einem Knoten zu einem anderen führen, im Web würde man von Links sprechen. Mathematisch gesehen handelt es sich um spezielle Relationen, ein Begriff der Mengenlehre, von denen die wichtigsten im Kapitel dargelegt werden. Die Verbindung zur Praxis liefert der Abschnitt 3.2 über *Meshes*, den Netzwerken der 3D-Computergrafik. Vertices, Edges, Faces und Normale sind damit verbundene Begriffe, die in dem Abschnitt beschrieben werden und die in der Software als Datenstrukturen auftreten. Immer mit der beschränkten Speicherkapazität kämpfend und um eine hohe Verarbeitungsgeschwindigkeit ringend, werden im Abschnitt 3.2.2 wichtige Datenstrukturen vorgestellt. Speziell für die Navigation erstellte Netzwerke sind die Halfedge-Netzwerke, die noch etwas näher betrachtet werden.

Im Kapitel 4, *Objekte* (Seite 169 ff.), stehen die Individuen einer Szene im Mittelpunkt. Dabei handelt es sich um Objekte, von den Grafikern auch Modelle genannt, die in einer Szene oder einer 3D-Zeichnung einzeln angesprochen werden. Diese müssen natürlich erst einmal in einer Szene bereitgestellt werden, und daher befasst sich das Kapitel zuerst mit der Übergabe von Objekten an OpenGL. Der Schwerpunkt des Abschnittes sind Vertex Buffer Objects (VBO), auch im Zusammenspiel mit einem Vertexshader. Der darauf folgende Abschnitt befasst sich mit der Erzeugung von Objekten in einer Applikation. Es werden die verschiedenen Aspekte eines Objektes näher betrachtet, hier steht insbesondere die Verbindung mit den VBOs sowie die korrekte Orientierung der Flächen und der Normalen im Mittelpunkt. Anhand des erstellten Wedge-Objektes können in einem Beispielprogramm diverse Effekte, die Flächenverhüllung und Beleuchtungseffekte betrachtet werden. Sind die Objekte platziert, kann eine Szene auch gerendert werden – für viele Anwendungen durchaus ausreichend. Mindestens genauso viele Anwendungen erfordern aber auch einen Eingriff des Nutzers, und sobald sich einzelne Objekte in einer Szene bewegen, besteht auch eine Kollisionsgefahr. Objekte nehmen einen Raum in der Szene ein, der nur bedingt mit anderen geteilt werden kann – ein möbiliertes Haus etwa –, und die einzelnen Teile oder auch Objekte als Ganze sollen für eine Identifizierung ausgewählt werden. Mit dieser Problematik befassen sich die Abschnitte 4.4, 4.5 und 4.6.

Mit der Beleuchtung und dem gezielten Einsatz von Farben erhalten virtuelle Welten eine ganz neue Qualität. Licht und Farbe sind ausgesprochen komplizierte Eigenschaften der realen Welt, und die Nachbildung in der virtuellen Welt kann nur unvollkommen und mit massivem Einsatz von Tricks sowie Vereinfachungen gelingen. Die genauesten Verfahren basie-

ren auf energetischen Betrachtungen, den Radiosity-Modellen und einer Strahlverfolgung. Beide Techniken waren aber in der Vergangenheit kaum für den Desktop zugänglich, und sie sind auch heute noch für das Echtzeit-Rendering nur mit hohen Anforderungen an die Hardware durchführbar. Daneben haben sich vereinfachte Modelle durchgesetzt, welche die Grundlage für das Rendern mit OpenGL bilden. Im Kapitel 5, *Licht und Farbe* (Seite 301 ff.), werden diese Modelle und ihr mathematischer Hintergrund vorgestellt. Ein mit Shadern erstelltes Programmbeispiel rundet die Einführung ab. Es ermöglicht die Änderung aller Licht- und Materialparameter, des Standardmodells von OpenGL und das Studium der Auswirkungen in einer virtuellen Szene.

Neben den Meshes, der Repräsentation der Objekte in der virtuellen Welt, sind die Texturen vielleicht das wichtigste Element für die 3D-Computergrafik-Software. Texturen müssen als Bilder zur Verfügung gestellt werden, und daher ist es logisch, dem Leser zuerst die Verwaltung und Darstellung von Bildern in OpenGL zu präsentieren, wie im Kapitel 6, *Images, Texturen und Text* (Seite 355 ff.). Das Ziel ist es, Bilder an OpenGL zu übergeben, aber dazu müssen diese erst einmal von einer Datei geladen und im Speicher mit den Mitteln der Applikationssprache abgelegt werden. Dieser Weg wird in dem Abschnitt ausgeleuchtet, ein Weg der zwangsläufig an der Klasse `BufferedImage` aus der Java2D-Bibliothek vorbeiführt. Während Bilder noch eigenständige Elemente sind, werden Texturen auf die Oberflächen der virtuellen Welt aufgebracht, es muss daher auch eine in der Software abgebildete Verbindung zwischen den Polygonen eines Mesh und den Bildern vorhanden sein. Diese Verbindung stellen die Texturkoordinaten her, deren Herkunft und Behandlung dargelegt wird. Auch das „Aufkleben“ selbst, im Entwickler-Jargon *Mapping* genannt, ist ein wichtiger Vorgang, der im Abschnitt 6.2 ausgeführt wird. Es sind einige Programmbeispiele zu finden, welche im Umfeld der Texturen angesiedelt sind. Für den Informationsarbeiter ist das geschriebene Wort von grundlegender Bedeutung. Nun mag es erstaunen, dass gerade Text nur mit aufwändigen Mitteln und mit wenig Hilfe des Renderers in einer virtuellen Welt aufscheinen kann. Aber Text bedeutet in der modernen Welt die Darstellung sehr großer Zeichensätze, die als 3D-Objekt, auch noch in verschiedenen Größen, nur sehr aufwändig gerendert werden können. Diese Verfahren und einfachere Alternativen dazu werden im Abschnitt 6.3 dargelegt.

Im Kapitel 7, *Mathematische Grundlagen* (Seite 431 ff.), findet sich der Lesestoff der einzelne Aspekte der 3D-Computergrafik aus rein mathematischer Sicht. Die Themenwahl konzentriert sich darauf, Leser ohne tiefen mathematischen Hintergrund an die Mathematik der 3D-Computergrafik heranzuführen. Der Abschnitt zur Vektorrechnung ist typisch für diese Themenwahl. Andererseits geht das Kapitel auch über eine Einführung hinaus. Es werden auch Themen und Problemstellungen angeschnitten, auf die der Leser erst später bei ausgedehnten Expeditionen in das Reich der 3D-

Computergrafik stößt. Die Matrixinversion und Quaternionen sind Beispiele für solch weiterführende Themen. Eine dritte Gruppe bilden ergänzende Abschnitte. Hierzu zählt etwa der Abschnitt über die Zahlen und deren Darstellung auf dem Rechner nebst den damit verbundenen Rundungsproblemen. Auch der Abschnitt über Drehungen ist als Ergänzung gedacht.

Als Buch im Buche kann das Kapitel fortlaufend gelesen oder bedarfsweise herangezogen werden, die Übergänge der oben genannten Abschnitte sind fließend. Wenn dem Leser ein Abschnitt zu schwierig oder weniger interessant für den Augenblick erscheint, möge er ruhig abbrechen und später einmal darauf zurückkommen. So ist es gedacht, und es sind keine Prüfungen abzulegen.

Das Kapitel 8, *Programmierung* (Seite 547 ff.), bietet zuerst eine Einführung in die Shaderprogrammierung. Shaderbeispiele finden sich im gesamten Text, aber in diesem Abschnitt wird speziell auf die Verwaltung und Erzeugung von Shadern eingegangen. Auch liefert der Abschnitt eine mehr formale Einführung in die Programmiersprache der Shader. Im Abschnitt 8.2 wird eine kleine Klassensammlung vorgestellt, die an die Belange der Java-Programmierung angepasst ist. Solche Klassen, wie etwa `VectorF3` und `TransformF4x4`, werden häufig benötigt. Im genannten Abschnitt wird auch auf die Problemstellungen bei der Erstellung eigener Klassen oder der Nutzung von Mathematikklassen von Drittanbietern eingegangen. Im Abschnitt 8.3 stehen die Einrichtung großer Datenpuffer und die Bindung von Java an OpenGL im Mittelpunkt.

Das Kapitel 9, *OpenGL-Funktionen* (Seite 637 ff.), liefert eine Übersicht der im Buch verwendeten OpenGL-Kommandos. Während im Hauptext die Verwendung dieser Funktionen in einem bestimmten Umfeld dargelegt wird, gibt die Übersicht weitergehende Informationen. Es werden die Parameter der Funktionen beschrieben und deren Auswirkung bzw. Bedeutung auf die OpenGL-Statusmaschine erklärt. Im Haupttext dient eine OpenGL-Funktion vorrangig als Beispiel dafür, wie mit einem Renderer ein bestimmter Effekt erzeugt oder ein Algorithmus realisiert werden kann. Im OpenGL-Kapitel steht die OpenGL-Funktion selbst im Mittelpunkt der Aufmerksamkeit. Bei einigen Funktionsbeschreibungen ist auch ein umgekehrter Weg möglich, denn es sind auch Verweise auf den Haupttext eingefügt. Sie erhalten in diesem Kapitel auch eine Übersicht über die Versionen von OpenGL. Der Abschnitt 9.4 befasst sich eingehender mit den Variablen und Funktionen der Shadersprache GLSL.

Einstieg in das Buch

Das Buch wendet sich an verschiedene Lesergruppen, vornehmlich spricht es jedoch Leser an, für die die 3D-Computergrafik noch Neuland ist. Es ist andererseits kein klassisches Anfängerbuch, denn einige Themen werden

recht umfassend behandelt. Anfängern empfehle ich, ohne Gewissensbisse ruhig aus einem Thema auszusteigen, wenn es zu schwierig oder für den Augenblick nicht interessant erscheint. Generell behandelt das Buch den mathematischen Hintergrund und die Programmierung zu etwa gleichen Teilen, je nach Schwerpunkt wird der Leser den ein oder anderen Abschnitt nur überfliegen. Viele Fragen, die in Foren auftauchen, besonders im Zusammenhang mit der Shaderprogrammierung, basieren auf einer unklaren Vorstellung über die Lage von Punkten und Vektoren relativ zu einem Koordinatensystem sowie dem Wechsel desselben, und daher habe ich das Kapitel über den dreidimensionalen Raum an den Anfang gestellt. Folgende Zielgruppen lassen sich für das Buch zusammenfassen:

Anfängern
: oder Lesern, die sich vornehmlich für die Programmierung interessieren und die schon etwas Erfahrung mit Java oder einer anderen Programmiersprache haben – für die die 3D-Computergrafik aber noch ein Buch mit sieben Siegeln ist – empfehle ich zuerst die Einführung sowie die Abschnitte 8.3.5 (Seite 614 ff.) und 8.3.6 (Seite 623 ff.). Danach sollten sie das Beispiel `SimpleZoom` (Abschnitt 8.3.8, Seite 626 f.) auf dem eigenen Rechner zum Laufen bringen. Im Anschluss empfehle ich die Abschnitte 7.4 (*Vektoren*, Seite 463 ff.) und 7.3 (*Matrizen*, Seite 452 ff.). Danach erst sollten Sie „linear“ lesen, beginnend mit dem Kapitel 2.

Einsteiger in die 3D-Computergrafik
: mit mathematischem Hintergrund beginnen am besten mit der Einführung und dem Kapitel 2 und ziehen gegebenfalls noch den Abschnitt 7.4 (*Vetoren*, Seite 463 ff.) hinzu. Ich empfehle ebenfalls das bereits erwähnte Programmbeispiel `SimpleZoom`, um so eine Programmbasis für eigene Experimente zu gewinnen.

Leser mit DirectX-Erfahrungen
: sollten auf jeden Fall mit der Einführung beginnen und dann mit Kapitel 2 fortfahren. Es wird für Sie wichtig sein, da OpenGL mit einer anderen Orientierung der Koordinatensysteme agiert. Ich empfehle zusätzlich auch das Überfliegen der OpenGL-Referenz – so erhalten Sie auch eine grobe Vorstellung von der Begriffswelt und den Funktionen von OpenGL.

Die 3D-Computergrafik ist ein weites Feld, und es gibt zahlreiche Middleware für bestimmte Anforderungen; so ist es schwierig, sich am Anfang zu orientieren – da geht es Ihnen nicht anders als dem Autor, wenngleich dieser einen zeitlichen Vorsprung hat. Die Mathematik hinter der 3D-Computergrafik wird häufig unterschätzt, sie ist aber letztlich der Orientierungsrahmen für bestimmte Problemlösungen. Darum ist es für alle Lesergruppen sinnvoll, mit dem Abschnitt 7.4 über Vektoren einzusteigen und so zumindest ihr Schulwissen wieder aufzufrischen.

Auch wenn es nicht direkt auf der Linie des Buches liegt, empfehle ich Ihnen, die freie Software Blender herunterzuladen[2] und zu installieren. Probieren Sie das ein oder andere Tutorial,[3] denn damit erhalten Sie einen guten Überblick über Anforderungen und Aufgaben der 3D-Computergrafik, sozusagen ein Gefühl dafür, worum es dabei eigentlich geht und welche Gestaltungselemente es gibt.

Viele Beispiele sind im Buch aus Platzgründen nicht vollständig abgedruckt; Programmquellen und auch Datenfiles finden Sie auf den Verlagsseiten zum Download:

`https://www.opensourcepress.de/fileadmin/osp/download/gBook.zip`

Eine weitergehende Beschreibung für den Download und die Einrichtung einer Programmierumgebung auf Ihrem Rechner liefert der Abschnitt *Programmierumgebung der Beispiele* (Seite 624).

Als Autor lebt man für eine gewisse Zeit in seiner eigenen Welt, getrieben von der enormen Fülle des Materials und der Notwendigkeit, daraus eine sinnvolle Auswahl zu treffen und diese in geeigneter Weise aufzubereiten. Nach einigen Jahren folgt dann der Zeitpunkt, an dem man sein „Baby“ dem Verlag als Rohtext übergibt – keine einfache Lektüre, durchsetzt mit Verweisen auf Bilder, Formeln und Quellen. Die Überführung in das Format des Verlages nebst den Korrekturen bei der „ersten Lesung“, in meinem Falle durch Herrn Franz Mayer, war keine leichte Aufgabe: Respekt! Bei der Durchsicht der ersten Fahnen wird das Ergebnis erkennbar und wie der Verlag Probleme löst – in Layout und Text, aber auch in der Kommunikation mit dem Autor. All das gefiel und gefällt mir sehr gut, und dafür danke ich dem Verleger, Herrn Dr. Markus Wirtz.

Es gab in meinem Freundeskreis Personen, die mir durch Tipps und dezente „No-Go“-Hinweise geholfen haben, einigen Fallgruben auszuweichen – auch dafür mein herzlicher Dank.

Gerhard Virag — München, Juli 2012

[2] `http://www.blender.org`
[3] Zum Beispiel von `http://www.agenzasbrothers.de`.

Kapitel

1

Einführung

Eine Einführung in die 3D-Computergrafik ohne Formeln und die Darlegung des mathematischen Hintergrundes gleicht einem Kochbuch, das ausschließlich Rezepte enthält. Sicher lassen sich damit Speisen bereiten, aber es weist nicht den Weg zu echtem Kochen – hier sind Kenntnis von und Verständnis für die Zutaten ebenso gefragt wie Kreativität. Zwar lassen sich auch in der Computergrafik allein mit „Rezepten“ wie Schnittstellenbeschreibungen und Code-Beispielen ohne großen Aufwand ansehnliche Ergebnisse erzielen; wirklich gute Lösungen schafft man aber nur mit Verständnis für die theoretischen Zusammenhänge.

Mit der vorliegenden Einführung in die 3D-Computergrafik wird der Leser, um im Bild zu bleiben, in die Lage versetzt, eigene Rezepte zu kreieren und die Zubereitung der Speisen – einschließlich der Beurteilung der Ausgangsprodukte wie des Ergebnisses – anderen zu erklären und vorzuführen; kurz gesagt: Sie lernen das Kochen. Java und OpenGL sind dabei lediglich das Kochgeschirr, das Sie für Ihre Arbeit benötigen.

3D-Computergrafik umfasst zwei Tätigkeitsfelder: die Gestaltung virtueller Welten und die Softwareentwicklung. In diesem Buch ist der Softwareentwickler der Koch, der vorbereitete Nahrungsmittel auf dem Markt einkauft und daraus seine Speisen zubereitet. Konkret bedeutet diese Analogie, dass Sie 3D-Modelle und Texturen in der Regel nicht selbst erstellen, diese aber in Ihre Programme importieren und gegebenenfalls anpassen. Wenn Sie auch noch selbst die Zutaten herstellen wollen – oder wenigstens einen Eindruck davon bekommen möchten, was das bedeutet –, dann empfehle ich die freie 3D-Grafiksoftware *Blender*.[1]

Um den Programmbeispielen zu folgen, diese auszubauen oder eigene Programme zu schreiben, sollte der Leser die Grundlagen der Programmiersprache Java kennen und schon das ein oder andere Progamm erstellt und zum Laufen gebracht haben. Leser, die aus anderen Programmierwelten stammen, können ebenfalls von diesem Buch profitieren, denn hier wird nur einfaches und klar verständliches Java zu finden sein, so dass der Inhalt gedanklich leicht in andere Sprachen zu übertragen ist.

Sie werden in diesem Buch keine Übungsaufgaben finden – diese zu stellen, sei der Schule oder den Professoren überlassen. Dafür gibt es eine Fülle an Programmbeispielen und Anregungen sowie einen sorgfältig aufbereiteten Grundstoff, gedacht für den spielerischen Umgang mit der Materie. Stellen Sie sich Fragen zum Text und verwenden Sie den beigelegten Programmcode für eigene Lösungen im Vertrauen darauf, dass Sie so „nebenbei“ zu einem gestandenen 3D-Computergrafikentwickler werden.

1.1 Programmbeispiele

Programmbeispiele sind im Buch ohne Dokumentation der Klassen und Methoden wiedergegeben. In den originalen Quellen ist jedoch diese Dokumentation zu finden, die auch eine automatische Erzeugung (JavaDoc) erlaubt. Auf die detaillierte Dokumentation der Methoden in den Buchbeispielen wurde verzichtet. Grundsätzlich aber steht der Autor zu dem Satz: „Nur dokumentierte Software ist nützliche Software.“

Im Text gibt es drei Arten von Codebeispielen: *Programmfragmente* verdeutlichen einen bestimmten Aspekt, etwa die Einstellung einer Seitenansicht. Oder sie zeigen auf, wie man mit OpenGL ein Konzept, ein Algorithmus oder ein Effekt umsetzt. Auf die Position solcher Programmabschnitte findet sich meist ein Hinweis, etwa `initialize`, wenn der Abschnitt einmalig in einem Initialisierungsteil erscheinen sollte, oder `display`, wenn es sich um ein Progammfragment handelt, das typisch ist für das Rendern eines Frames. Ich empfehle, die OpenGL-Funktionen in Kapitel 9 nachzuschlagen, das weitere Aspekte der Funktion oder ihrer Argumente erläutert.

[1] `http://www.blender.org`

Gegen Ende eines Kapitels finden sich *Projekte*, das sind vollständige Programme, die eine ganze Reihe von Konzepten verdeutlichen. Sie sind als Grundlage für eigene Experimente gedacht. Da der Quellcode der meisten Programme viel zu groß ist, um im Text abgedruckt zu werden, beschränke ich mich auf die Deklaration von Methoden und Klassen und präsentiere Auszüge, welche im unmittelbaren Zusammenhang mit dem Text stehen. Im Abschnitt 8.3.8 ab Seite 626 finden Sie eine Übersicht der vollständigen und ausführbaren Programmbeispiele.

Die dritte Art der Programmbeispiele enthalten *Bibliotheksklassen*, die als Werkzeug benutzt werden, als Basisklassen für Spezialisierungen dienen oder als Getriebe verschiedene Programmteile miteinander verbinden. Zwischen den Programmbeispielen im Buch und dem Quellcode können Abweichungen auftreten.

Die Programmbeispiele sind keine Verträge; zwar kommt es auf die genaue Formulierung an, aber Sie müssen sich nicht an die Verträge halten. Experimentieren Sie mit und ändern Sie den Programmcode ganz nach Ihren Vorstellungen.

1.2 Warum Java und OpenGL?

Hier die Argumente für diese Wahl in einer Übersicht:

- Java ist auch eine Einsteigersprache, und das gewonnene Know-how können Sie für große, ja sogar sehr große Projekte nutzen. So sind zum Beispiel große Teile der SAP-Software in Java erstellt. Auch ist Java eine klare und einfache Sprache, die jederzeit in andere Sprachen (insbesondere C++) übersetzt werden kann. Die Verbindung zwischen OpenGL und Java ist über die Bindungssoftware JOGL realisiert; moderne Betriebsysteme wie Android enthalten bereits einen angepassten Java-Interpreter.

- OpenGL mag für einen Spiele-Interessierten auf den ersten Blick nicht die erste Wahl sein, aber es ist eine professionelle Software, und zwar für überaus vielfältige Anforderungen. Als ein Beispiel sei das „World Wind“-Projekt[2] der NASA genannt, das auf die Erdkugel Themen projiziert, etwa die Landesgrenzen und andere Pläne (beeindruckend: die Zahl der aktiven Feuer auf diesem Globus in Echtzeit). „World Wind“ ist Open Source, in Java geschrieben und rendert mit OpenGL; als Bindung kommt JOGL zum Einsatz (siehe dazu auch Abschnitt 8.3.5 ab Seite 614). Zudem besteht die Welt nicht nur aus Windows, es gibt auch andere Betriebssysteme, wie Linux oder Handy-Betriebsysteme, auf denen OpenGL die erste Adresse für 3D-Computergrafik ist. OpenGL konzentriert sich darauf, ei-

[2] http://goworldwind.org/demos/

ne ausgezeichnete Renderer-Software zu spezifizieren, nicht mehr und nicht weniger.

- Wichtig ist es festzuhalten, dass sich OpenGL nicht stark von DirectX unterscheidet; beide habe die gleichen Eltern (SPHIGS und PHIGS), wie im Buch von James D. Foley und Andries van Dam dargelegt.[3] Konzepte können in OpenGL sogar klarer präsentiert werden, denn es ist weniger Overhead zu kommentieren, da OpenGL dicht an der Grafikpipeline residiert. In diesem Buch geht es um Konzepte und um Lösungsansätze im 3D-Grafikbereich – gleichzeitig um tragfähige Lösungen, auf die Leser aufbauen können. In diesem Bereich ist die Spieleprogrammierung aufgrund der sehr vielfältigen Anforderungen so etwas wie die Meister- oder gar Großmeisterklasse. Nun, wer Großmeister werden will, muss zunächst Lehrlings-, Gesellen- und Meisterstufe bewältigen. Mit diesem Buch bewegt sich der Lehrling, aber auch der Geselle, auf der sicheren Seite. Sicherheit heißt hier, dass die Grundlagen verstanden sind und die Beispiele selbst nachvollzogen wurden, ohne Overhead, der durch die Komplexität einzelner Produkte entsteht. Darum sind hier Java statt C++ und OpenGL statt DirectX die erste Wahl.

- Open Source ist das Gegenteil von „Information Hiding“. Zwar ist die Implementierung der Software für OpenGL nicht Open Source, aber die Schnittstelle ist herstellerunabhängig und plattformübergreifend; zudem erfreut sich OpenGL einer großen Nutzergemeinde. Je mehr sich die Anwendungen auf unterschiedliche Plattformen und in die Browser verlagern, desto mehr wird auch OpenGL an Boden gewinnen.

1.3 Warum nicht Java 3D?

Java 3D ist eine professionelle Software die, mit dem nötigen Hintergrundwissen eingesetzt, das Rendern animierter Szenen erlaubt, einschließlich der Interaktion mit dem Benutzer.

Ich habe aus mehreren Gründen Java 3D nicht als Grundlage gewählt: Java 3D ist vor allem ein Szenengraph mit einer besonderen Stufe der Abstraktion, sehr weit weg von der Rendering Pipeline und den mathematischen Konzepten. In einem Buch über Java 3D lernen Sie, wie mit *dieser* Software programmiert wird; hier geht es aber um die Grundlagen, die Sie für *jede* Software im 3D-Bereich benötigen. Auch für Spieleprogrammierung wäre Java 3D nicht meine erste Wahl, ich würde eher zu der Java Monkey Engine (jME) oder einer Portierung aus der C++-Welt greifen. Damit Sie mich nicht missverstehen: Ich halte Java 3D für eine vorbildliche Architektur, gut als theoretische Grundlage für eigene Projekte. Die jME wird weiterentwickelt,

[3] James D. Foley, Andries van Dam u.a.: „Computer Graphics: Principles and Practice“, Addison Wesley, 1995.

bei Java 3D ist die zukünftige Entwicklung unklar (Stand Anfang 2012), auch besteht ein wesentlicher Unterschied zwischen einem Szenengraphen und einer ausgewachsenen Game-Engine. Sehr problematisch ist die fehlende Shader-Integration.

1.4 Was macht OpenGL eigentlich?

Diese Frage ist etwas unscharf, denn OpenGL ist ein Standard, der eine bestimmte Rendering Pipeline definiert; als solcher „macht“ OpenGL eigentlich gar nichts, außer Dokumente erzeugen und die Einhaltung des Standards überwachen.

Hinter OpenGL steht eine Firmengruppe,[4] welche die Entwicklung und Pflege des Standards finanziert, Personal und Ressourcen stellt sowie Impulse für die Weiterentwicklung liefert. In der Firmengruppe stark vertreten sind große amerikanische Softwarefirmen, wie Apple, IBM, Intel und Hersteller von Grafikkarten.

Die Zielgruppe von OpenGL sind in erster Linie Firmen und deren Entwickler, die den Standard auf der Grundlage ihrer Hardware implementieren. Die Spezifikation ist frei zugänglich,[5] doch ist erheblicher Sachverstand erforderlich, um auf der Basis dieses Papiers Software zu erstellen. Der Fokus der Spezifikation liegt bei der Schnittstelle zu Applikationen, die die Rendering Pipeline nutzen wollen. Es werden aber auch Festlegungen über den inneren Aufbau, insbesondere über die Ausführung bestimmter Bestandteile, getroffen. Zum Beispiel wird die Funktion `glOrtho` nebst Parametern definiert (Schnittstelle), aber auch klar bestimmt, welchen Aufbau die damit vorgegebene Projektionsmatrix haben muss oder dass intern im `float`-Format zu rechnen ist.

Blickt man von der Anwenderseite auf die Spezifikation, bleiben die Funktionen, über die OpenGL ansprechbar ist. Präzise muss man bei diesem Blickwinkel von dem „OpenGL-Framework“ sprechen, denn die einzelnen Funktionen üben intern Einfluss aufeinander aus. Auch ist es möglich, innere Bestandteile des Frameworks zu verändern. Ein Beispiel dafür ist die `glLoadMatrix`-Funktion, mit der eine vorgegebene Matrix gegen eine applikationsspezifische Matrix ausgetauscht werden kann, und natürlich die Shader, mit denen ganze Funktionsblöcke ausgetauscht werden.

Die Implementation, also die in Software gegossene Spezifikation, ist ein Treiber, der zwischen dem Betriebssystem und den Applikationen auf der einen Seite und der Grafikhardware auf der anderen Seite vermittelt. Implementiert wird nicht durch das OpenGL-Konsortium, sondern durch die Hersteller der Hardware, die dann zusammen mit ihren Produkten oder

[4] http://www.khronos.org
[5] http://www.opengl.org/registry/doc/glspec41.core.20100725.pdf

als Bestandteil des Betriebssystems den Treiber ausliefern. Hersteller, die OpenGL-Software mit ihren Produkten ausliefern wollen, benötigen eine Lizenz von der Khronos-Gruppe, und es ist ein Verfahren eingebaut, das es den Herstellern ermöglicht, OpenGL zu erweitern. Dafür ist das ARB (*OpenGL Architecture Review Board*) zuständig, das auch die Konformitätstests mit der produzierten Software durchführt. OpenGL ist somit nur für den Anwender freie Software.

Meist ist der Treiber gemeint, wenn wir sagen: „OpenGL tut dies oder das." Von der Spezifikation oder dem Standard sprechen wir in Formulierungen wie: „OpenGL stellt dafür die Funktion XY zur Verfügung." Vom Konsortium ist die Rede in Feststellungen wie: „OpenGL befasst sich mit dem Thema XY." Die Eingangsfrage wäre damit beantwortet – oder doch nicht?

1.4.1 Die Rendering Pipeline

Das erste Computerspiel ist wahrscheinlich „Tennis for Two", das William Higinbotham 1958 für einen Analogrechner entwickelte, um für seine Abteilung beim „Brookhaven National Laboratory" Öffentlichkeitsarbeit zu betreiben. Das „Rendern" bestand in der Beinflussung eines Strahls durch die Spieler auf einem modifizierten Oszilloskop, mit einer Darstellung der Flugbahn des Tennisballes abhängig vom Abschlagswinkel. Hier wurde die Umsetzung einer Spielidee in drei Teile aufgespalten: Eingabegerät (Kasten mit Knopf und „Joystick"), Darstellung der Objekte (Tennisball, Flugbahn und Netz) auf einem Bildschirm sowie das eigentliche Spielkonzept.

Das erste programmierte Computerspiel war wohl „Spacewar" von Stephen Russell, entwickelt 1962 auf einer PDP-1, einer damals extrem teuren Maschine. Hier waren schon Sprites als Objekte in einer primitiv dargestellten Weltraumschlacht zu sehen. Die nächsten Schritte führten über Heimkonsolen und Spielautomaten ab ca. 1971. Apple wurde 1976 gegründet, und eines der ersten Spiele auf dem Apple-I war „Zork", das zunächst noch ohne jede Grafik auskam; „gerendert" wurde im Kopf des Spielers, der die durch Texte vermittelte Situation in seiner Phantasie ausmalte und mit einfachen Anweisungen („gehe nach links") den Spielablauf beeinflusste. Bis etwa 1978 bestand die grafische Ausgabe der Heimkonsolen und Spieleautomaten in einer Rastergrafik, das Bild wurde immer zeilenweise aufgebaut und zum richtigen Zeitpunkt musste ein Pixel die passende Farbe erhalten. Larry Rosenthal baute 1978 für einen Spielhersteller eine Adaption von „Spacewar", „Spacewars" genannt, das auf einer Vektorgrafik basierte. Hier wurde die Erzeugung eines Rechtecks nicht mehr im Programm vollständig abgebildet, vielmehr genügte es, die Ecken und eine Füllfarbe anzugeben. Mitte 1981 änderte sich die Welt, denn die Firma IBM brachte ihren Personal Computer mit dem Betriebssystem MS-DOS auf den Markt, und damit waren auch einfache Animationen möglich.

Neben diesen, aus der Sicht einer breiten Öffentlichkeit spektakulären Entwicklungen entwickelten sich Hard- und Software für die grafische Ausgabe stetig weiter. Wichtig ist hier das Jahr 1977, in dem die Spezifikation „3D Core Graphics System“ von ACM SIGGRAPH vorgestellt wurde, aus dem dann 1988 das GKS-3D hervorging, ein durch das ANSI zum weltweiten Standard erhobenes „Graphical Kernel System“, erweitert auf 3D. Weitere Meilensteine waren dann PHIGS+ (Programmers Hierarchical Interactive Graphics System) und das Derivat SPHIGS (Simple PHIGS). Ein Merkmal von PHIGS ist die Transformation der Punkte, Linen, Kurven und Polygone, verbunden mit einer Hardwareunterstützung der Transformationen, des Clippings usw. Das „simple“ in SPHIGS sollte nicht falsch verstanden werden, denn SPHIGS fügte auch die Gleitkommarechnung (Float) und 3D-Koordinatensysteme hinzu. PHIGS und SPHIGS waren nicht die einzigen Grafik-Pipelines ihrer Zeit; auch „Renderman“ von Pixar aus dem Jahr 1986 ist hier zu nennen: „Luxo Jr.“, der Kurzfilm mit der kleinen Schreibtischlampe.

Der Begriff *Rendering Pipeline* ist zeitlich nicht genau zu fixieren, denn gemeint ist eigentlich jede Form der Umwandlung einer Szene mit 3D-Objekten in ein zweidimensionales Bild. „Pipeline“ ist Computerjargon aus der UNIX-Welt, wo die Ausgabe eines Prozesses oder Programmes als Eingabe an ein anderes Programm weitergereicht wird. Dieses Weiterleiten von Daten an Stationen, die eine Verarbeitung vornehmen, ist allen Rendering Pipelines gemeinsam. Es kann aber nicht von *der* Rendering Pipeline gesprochen werden, denn die Schnittstellen zwischen den Stationen sind je nach Renderer unterschiedlich gestaltet, auch können Teile je nach Bedarf in unterschiedlichen Feinheitsgraden untersucht werden.

OpenGL spricht von einer *Processing Pipeline*. Kommandos, die Funktionen von OpenGL, wirken sich auf diverse Stationen der Verarbeitung von Daten aus oder sie reichen diese intern an Folgestationen weiter. Bei OpenGL-Kommandos bzw. -Funktionen unterscheidet man drei Kategorien, wobei einige durchaus mehreren Kategorien angehören können. So gibt es Funktionen, die nur einen Zustand herstellen und kontrollieren, etwa `glEnableClientState`, die einen Modus, genauer: Kontext, für die nachfolgende Datenübergabe einstellt. `glBufferData` ist eine Funktion zur Datenübergabe, etwa der Vertices. Daneben gibt es Funktionen, die eine Verarbeitung abschließen und das Rendern auslösen. Hier ist, um beim obigen Beispiel zu bleiben, die Funktion `glDrawElements` anzuführen. Schaut man in die OpenGL-Spezifikation, stößt man in erster Linie auf die Festlegung der Kommandos und deren Auswirkungen.

OpenGL ist zustandsgetrieben, Sie können sich also auf einen eindeutig definierten Zustand verlassen und dieses Wissen für Ihre Eingriffe in die Pipeline nutzen. Schieben Sie – für einen Moment – die Vorstellung von sich bewegenden und animierten Objekten beiseite, das heißt von einer sich in der Zeit verändernden Szene. Jedes Kommando, das Sie an OpenGL

absetzen, verändert den Zustand von OpenGL, und die Änderungen können durchaus die gesamte Prozesskette durchlaufen. Wenn Sie aber aufhören, Kommandos zu senden, läuft die Prozesskette ab, und OpenGL geht in einen Wartezustand. Sie werden unter Umständen nicht einmal ein Bild am Monitor sehen, solange Sie keine Anweisung `flush` zur Übernahme des erzeugten Bildes in den Bildspeicher geben. Selbst wenn Sie eine solche Übernahme erzwungen haben, merkt sich OpenGL noch den aktuellen Zustand; daher müssen Sie vor der Erzeugung eines weiteren Bildes (*Frame*) auch Zustände explizit aufheben. Aus diesem Grunde wird am Anfang eines Frames zuerst der Puffer gelöscht, d. h. mit einer bestimmten Hintergrundfarbe überschrieben (`glClear`). Sie sind in der Situation eines Zeichentrickgrafikers, der jedes Bild malt und fotografiert (flush), um durch die schnelle Bildfolge die Illusion von Bewegung zu erzeugen. Im OpenGL-Konzept ist der Sender der Kommandos der Client und OpenGL der Server (Host); der Server wird seinerseits keine Anfragen beim Client durchführen, weder Callbacks noch andere Messages – auch keine Fehlermeldungen.

Eng verbunden mit der „Zustandsmaschine“ OpenGL ist deren Konsistenz: Alle Kommandos werden in der Abfolge ihres Empfangs abgearbeitet, und jedes Folgekommando kann die Verarbeitung des Vorgängers voraussetzen.

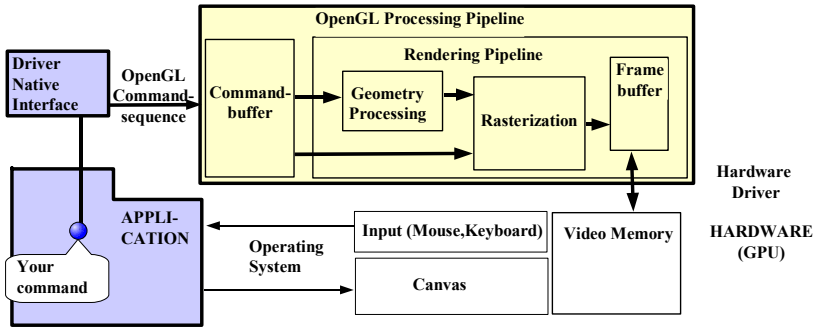

Abbildung 1.1 Einordnung des OpenGL-Treibers im Umfeld der Applikation und des Betriebssystems

Wenn Sie in Ihrer Applikation ein OpenGL-Kommando absenden, zum Beispiel `glClear`, dann muss dieses zuerst in einen Befehl umgewandelt werden, der Bestandteil des OpenGL-Treibers ist. In der Sprache Java bedeutet dies einen Java-Native-Zugriff auf eine externe Softwarebibliothek; dafür wird JOGL oder LWGL benötigt.

Ein Kommando landet erst einmal in einem Kommandopuffer, denn in manchen Fällen sammelt OpenGL Kommandos, bis ein bestimmter Zustand erreicht ist. Je nach Art des Befehls erfolgt die Weiterverarbeitung in der Geometry-Pipeline oder bereits eine Stufe weiter bei der Rasterstufe. Über den Hardware-Treiber, das ist hier der Treiber der Grafikkarte, gelan-