Jens Konerow

Managed DirectX und C#

Jens Konerow

Managed DirectX und C#

Einstieg und professioneller Einsatz

entwickler.press

Jens Konerow
Managed DirectX und C# - Einstieg und professioneller Einsatz
ISBN: 978-3-86802-213-1

Ein Imprint der Software & Support Verlag GmbH

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.

Ihr Kontakt zum Verlag und Lektorat:
Software & Support Verlag GmbH
entwickler.press
Geleitsstraße 14
60599 Frankfurt am Main
Tel: +49(0) 69 63 00 89 - 0
Fax: +49(0) 69 63 00 89 - 89
lektorat@entwickler-press.de
http://www.entwickler-press.de

Inhaltsverzeichnis

Vorwort

Insofern Sie als Entwickler nicht auf andere Schnittstellen wie OpenGL zurückgreifen, stellt DirectX das Fundament eines jeden Spiels dar. Im Gegensatz zu OpenGL beschränkt sich Microsofts Multimediaschnittstelle nicht auf die Ausgabe dreidimensionaler Grafiken, sondern bietet zugleich die zweidimensionale- und dreidimensionale Wiedergabe von Audiodaten (wobei OpenAL als Ergänzung zu OpenGL als direkter Konkurrent zu DirectSound anzusehen ist). Auch das Ansprechen von Eingabegeräten, wobei sich dessen Effekte nutzen lassen (wie z.B. Force Feedback), oder die Implementierung einer protokollunabhängigen Netzwerkfähigkeit stellen kein Problem dar. In diesem Buch wird Ihnen die Programmierung mit DirectX Schritt für Schritt näher gebracht. Mit diesen Kenntnissen werden Sie anschließend in der Lage sein, erste kleinere Projekte umzusetzen. Bei diesem Buch handelt es sich keineswegs um ein all umfassendes Nachschlagewerk, hierfür wäre schon ein ganzer Buch-Band nötig. Dass jene neue Welt nicht nur interessant sondern zugleich auch anstrengend sein kann, werden Sie bereits im ersten Kapitel feststellen. Da grundlegende Mathematik-Kenntnisse unverzichtbar sind, wird sich dieses Kapitel mit den wichtigsten Elementen befassen. Doch keine Angst, allzu tief greifend wird es nicht, denn DirectX bietet zahlreiche Funktionen, die Ihnen den Umgang mit Vektoren und Matrizen enorm erleichtern. Sofern Sie noch in keiner Weise etwas mit Vektoren oder Matrizen zu tun hatten, ist ein Buch über die lineare Algebra empfehlenswert.

An wen richtet sich dieses Buch?

Kurz gesagt richtet sich dieses Buch an all diejenigen, die sich für die Multimediaprogrammierung interessieren. DirectX ebnet den Weg zur Programmierung von Spielen. Endlich können Sie Ihrer Kreativität freien Lauf lassen und eigene virtuelle Welten erstellen. Doch damit ist es noch nicht getan. CAD-Programme, Bildschirmschoner, die simple Ausgabe kleiner Sounds bei eingetretenen Ereignissen oder kleine Netzwerkapplikationen sind denkbare Einsatzgebiete in der Programmierung mit dieser Multimediaschnittstelle.

Solang es Ihnen nicht an Kreativität, Motivation und einem Mindestmaß an Programmierkenntnissen fehlt, halten Sie genau das richtige Buch in den Händen. Zwar sind alle Beispiele in C# geschrieben, weshalb Ihnen diese Programmiersprache zumindest grundlegend bekannt sein sollte, dennoch eignet sich dieses Buch, dank der gemeinsamen Grundlage – dem .NET Framework, ebenfalls für Visual Basic-Programmierer. Doch keine Sorge, sollte der Quellcode mal etwas komplizierter bzw. komplexer ausfallen, erhalten Sie eine genaue Beschreibung der vorliegenden Anweisungen.

Außerdem sei zu erwähnen, dass es sich bei diesem Buch keinesfalls um eine Anleitung zur Entwicklung einer Engine handelt. Im Vordergrund steht die Verwendung der DirectX-Schnittstellen, wobei hin und wieder Techniken eingestreut werden, die für die Spieleprogrammierung unabdingbar sind. Wo immer es möglich war, wurden jene Elemente in Klassen gekapselt, so dass Sie diese mit wenig Aufwand in Ihren eigenen Projekten einsetzen können.

Ansprüche an Ihren Computer

Generell gilt, vor allem in der Spiele-Entwicklung, je neuer die Hardware, desto geeigneter ist das System. Ein absolutes (theoretisches) Minimum definiert das .NET Framework 2.0 oder die Entwicklungsumgebung. In der kleinsten Ausführung der Entwicklungsumgebung (in den Visual Studio Express-Versionen) entspräche dies einem Prozessor mit mindestens 600 MHz, 192 MB Ram und einer VGA-Grafikkarte. Empfehlenswert sind diese Angaben allerdings nicht. Für die recht gierige Entwicklungsumgebung sollte ein Arbeitsspeichervolumen von 512 MB oder mehr angestrebt werden. Im Bereich der Grafikprogrammierung ist ein sinnvolles Arbeiten ausschließlich mit einer beschleunigten 3D-Grafikkarte möglich. CPU und Grafikkarte sollten also nicht unbedingt dem absoluten Minimum entsprechen, aber Sie müssen auch nicht gleich viel Geld für die High-End-Modelle investieren. Ein zwei bis drei Jahre alter Computer reicht für die meisten Beispiele in diesem Buch vollkommen aus.

Bei der Shader-Programmierung sind die Ansprüche an die Grafikkarte am größten. Shader sind kleine Programme, welche von der Grafik-Hardware ausgeführt werden. Sollte Ihre Karte Vertex- und Pixel-Shader in den Versionen 1.1 und 2.0 nicht unterstützen, bleibt die Option der Softwareemulation. Zwar resultieren daraus extrem niedrige Frame-Raten, aber zum Testen der Beispiele reicht es vollkommen aus.

Softwaretechnisch gilt Visual Studio 2005 bzw. alternativ dessen kleine Geschwister Visual C# 2005 Express oder Visual Basic 2005 Express als Voraussetzung. Microsoft bietet die Express-Versionen zum kostenlosen Download an. Sowohl eine Internetinstallation als auch ein Image sind unter der Internetadresse *http://msdn.microsoft.com/vstudio/express/* zu finden.

Website und Buch-CD

Auf der CD-Rom zum Buch ist das DirectX 9 SDK (August 2006) enthalten, das Sie benötigen, um die Beispielprojekte ausführen zu können. Alle Beispiele sind selbstverständlich ebenfalls auf der CD-Rom enthalten. Zwar wurden die Beispielprogramme auf diversen Systemkonfigurationen getestet, dennoch sind Fehler nicht immer ausgeschlossen. Sollten Sie einmal Probleme mit einem der Beispielprogramme haben, dann scheuen Sie nicht, eine E-Mail an die Adresse *webmaster@jenskonerow.de* zu schicken.

Der Autor ist bestrebt, einige Zeit nach Erscheinen dieses Buches Ergänzungen und zusätzliche Beispiele auf der Internetseite *www.jenskonerow.de* anzubieten. Eventuelle Korrekturen werden ebenfalls auf der genannten Seite bekannt gegeben.

Über den Autor

Jens Konerow, geboren am 23. März 1985 in Greifswald, studiert Informatik an der Universität zu Lübeck. Nebenbei ist er als Software-Entwickler bei der Hamburger Firma KEEP IT SIMPLE Organisation und Information GmbH tätig.

Seit der siebenten Version entwickelt Jens Konerow mit der Multimediaschnittstelle DirectX, die damals erstmals für Visual Basic-Programmierer verfügbar wurde. Nachdem Microsoft die ersten Beta-Versionen von .NET der Öffentlichkeit präsentierte, ist der Autor begeisterter C#-Anhänger. Kurze Zeit später begann Jens Konerow dann in Verbindung mit der verwalteten Version von DirectX zu arbeiten, die heute als Managed DirectX bekannt ist und veröffentlichte ein Teil seiner DirectX-Kenntnisse in einer Artikelserie des dot.net-magazins sowie schließlich in dem Buch, dass Sie nun in den Händen halten.

Immer gern gesehen vom Autor sind Lob, Anregungen oder Kritik zum Buch. Nicht weniger erwähnenswert sind Korrekturhinweise, damit eventuelle Fehler schnell aus der Welt geschafft werden können.

1 Einführung

Schwerpunkt dieses ersten Kapitels sind die mathematischen Grundlagen, die nötig sind, um mit DirectX, insbesondere mit Direct3D, vernünftig arbeiten zu können. Neben den mathematischen Grundkenntnissen werden Ihnen die Komponenten von DirectX vorgestellt, Ihnen bei der Installation unter die Arme gegriffen und erste Konzepte von Direct3D vermittelt. Wenn Sie die nötigen Grundkenntnisse der Mathematik aufweisen können oder derzeit keine Muße haben sich mit dem mehr oder weniger „lästigen" Thematik auseinanderzusetzen, dann können Sie gleich nach dem DirectX-Überblick mit Abschnitt 1.5 fortfahren. Auf die ersten Beispielanwendungen treffen Sie dann in Kapitel 2.

1.1 Managed DirectX

Neben der „klassischen" DirectX-Variante, auf welche vorwiegend die C++-Programmierer zurückgreifen, existiert nun eine zweite Version namens „Managed DirectX" (MDX). Wie es der Name vielleicht vermuten lässt, gilt jene Version für Anwendungen mit verwaltetem Code. MDX ist der konsequente Weg Microsofts in Richtung .NET. Die Vorteile von Managed DX liegen auf der Hand: Sämtliche Features des .NET Frameworks können genutzt werden. Es sei als Beispiel die Speicherverwaltung genannt.

1.1.1 Installation

In der Regel funktioniert die Installation der DirectX-Komponenten ohne Probleme, so dass Sie nur den Bildschirmanweisungen folgen müssen. Sofern das .NET Framework noch nicht installiert war, als Sie die aktuelle DirectX-Version aufgespielt haben, wird eine Nachbesserung nötig. Hintergrund ist der, dass das Installationsprogramm von DirectX .NET-Assemblies im GAC (Global Assembly Cache) hinterlegt. Wenn kein .NET Framework installiert ist, existiert auch der GAC nicht. Oftmals ändert eine erneute Installation nichts an den fehlenden Assemblies. Wenn dem so sein sollte, dann müssen Sie das Setup mit folgenden Befehlszeilenargumenten aufrufen.

```
Setup.exe /InstallManagedDX
```

1.1.2 Die DirectX-Komponenten

Die Multimediaschnittstelle besteht aus diversen Komponenten für das jeweilige Einsatzgebiet. Anders ausgedrückt besitzt DirectX viele weitere Schnittstellen. Folgende Auflistung zeigt alle Komponenten von Managed DirectX und gibt eine kurze Erklärung zu deren jeweiligen Einsatzgebiet.

- Direct3D
  Für die Ausgabe zwei- und dreidimensionaler Grafiken ist Direct3D zuständig. Geometrie-Daten durchlaufen dabei mehrere Stufen (Pipelining) wobei beispielsweise deren Position und Ausrichtung in der virtuellen Welt definiert werden und die Lichtberechnung stattfindet.
- DirectSound
  Jene Komponente beschränkt sich nicht nur auf die simple Wiedergabe von Audiodaten, sondern bietet zudem eine dreidimensionale Klangakustik sowie die Möglichkeit Sound-Effekte hinzuzufügen. Außerdem lassen sich eingehende Audiodaten aufnehmen und auf der Festplatte speichern.
- DirectInput
  Neben der Maus und Tastatur gehören Joysticks und Lenkräder zur Standardausrüstung, um Spiele noch realistischer erleben zu können. DirectInput bietet durch sein Interface den Zugriff auf solche Hardware und erlaubt die Nutzung vorhandener Force Feedback-Effekte.
- DirectPlay
  Heutzutage sind Computer-Spiele ohne Mehrspielermodus kaum noch denkbar. Meist ist jener Teil das Erfolgsgeheimnis eines Spiels. DirectPlay ermöglicht dem Programmierer die protokollunabhängige Netzwerkprogrammierung. Diese Komponente ist speziell für Netzwerk-Spiele ausgelegt. Auch Sprach-Übertragungen finden Unterstützung.
- AudioVideoPlayback
  Diese Komponente eignet sich zur Wiedergabe von Audio- und Videodaten. Erwähnenswert an dieser Stelle ist sicher die Möglichkeit zum Abspielen von MP3-Dateien.
- DirectDraw
  DirectDraw galt als Komponente zur Ausgabe von 2D-Grafiken. Ab DirectX 8.0 wurden DirectDraw und Direct3D vereint, d.h. DD als solches gibt es in DX 9.0 nicht mehr. Um den .NET-Programmierern dennoch eine Möglichkeit zu geben, um auf die alte Schnittstelle zuzugreifen, vermittelt ein Wrapper zwischen der .NET-Anwendung und der alten DirectX 7.0-Bibliothek. In diesem Buch wird nicht weiter auf DirectDraw eingegangen. Wünschen Sie dennoch mehr Informationen, konsultieren Sie bitte die Dokumentation des DirectX SDKs.

1.2 Kartesische Koordinatensysteme

Erinnern Sie sich noch an das Thema Analysis aus dem Mathematik-Unterricht? Dort war eine Funktion gegeben, dessen Extrempunkte oder Grenzwerte anschließend ermittelt werden sollten. Um die Ergebnisse zu prüfen, durfte der Graph zu guter Letzt in ein Koordinatensystem übertragen werden. Dabei handelte es sich um ein kartesisches Koordinatensystem. In der Ebene besitzt solch ein Koordinatensystem zwei Achsen: Die X- und Y-Achse. Letztere zeigt nach oben. Hingegen zeigt die X-Achse nach rechts (siehe Abbildung 1.1).

Zeigen die Achsen in andere Richtungen, lässt sich durch Rotieren wieder der oben genannte Zustand erreichen. Erweitern Sie das Koordinatensystem um eine weitere Achse, dann lässt sich durch Rotieren des Systems nicht immer der Ausgangszustand wiederherstellen. Zwei Koordinatensysteme sind üblich: Das linkshändige und das rechtshändige kartesische Koordinatensystem.

Mathematiker bezeichnen den Ursprung eines Koordinatensystems als Origo (in der englischen Literatur werden Sie häufig auf den Begriff „origin“ stoßen). Beschreiben lässt sich ein Ursprung durch die xy-Koordinaten (0, 0).

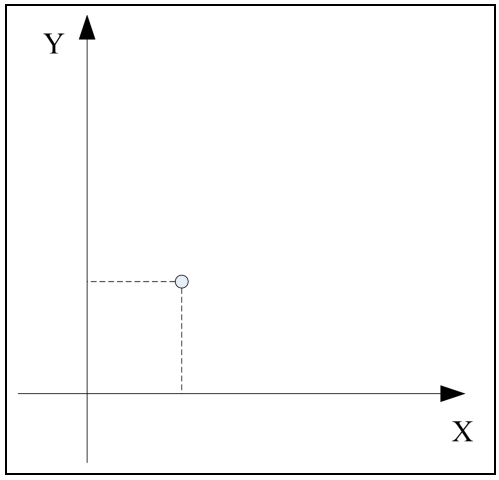

Abbildung 1.1: Ein kartesisches Koordinatensystem in der Ebene

1.2.1 Linkshändiges kartesisches Koordinatensystem

Standardmäßig verwendet Direct3D das linkshändige kartesische Koordinatensystem. Dabei sind die X- und Y-Achse genauso ausgerichtet wie im vorher beschriebenen Koordinatensystem. Die Z-Achse verläuft in den Monitor hinein, d.h. vom Betrachter weg (siehe Abbildung 1.2).

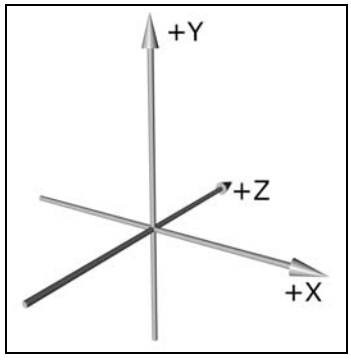

Abbildung 1.2: Linkshändiges kartesisches Koordinatensystem

1.2.2 Rechtshändiges kartesisches Koordinatensystem

Im Gegensatz zum linkshändigen kartesischen Koordinatensystem verläuft die Z-Achse zum Betrachter hin. Die Achse sticht quasi aus dem Monitor hinaus. Wollen Sie das rechtshändige kartesische Koordinatensystem einsetzen, dann sind Sie größtenteils auf sich allein gestellt, da nicht alle Hilfsfunktionen von DirectX sowohl für das linkshändige, als auch für das rechtshändige kartesische Koordinatensystem zur Verfügung stehen. Einige Methoden jedoch stellt DirectX in zwei Ausführungen bereit. In diesem Buch findet also ausschließlich die linkshändige Variante ihre Anwendung.

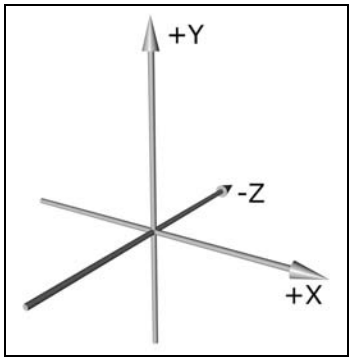

Abbildung 1.3: Rechtshändiges kartesisches Koordinatensystem

1.3 Vektoren

Vektoren gehören zur 3D-Programmierung wie der Deckel auf den Topf. Vielleicht erinnern Sie sich noch an den Physik-Unterricht, in dem Kräfte durch Vektoren repräsentiert wurden. Der folgende Abschnitt gibt Aufschluss darüber, worum es sich bei Vektoren genau handelt und wie Sie mit diesen rechnen können.

Wenn Ihnen jene Thematik noch ein Begriff ist, dann steht es Ihnen frei diesen Abschnitt zu überspringen.

1.3.1 Unterschied: Punkt – Vektor

Punkte, ob in der Ebene oder im Raum, werden immer durch zwei bzw. drei Koordinaten definiert. Abbildung 1.1 zeigt einen Punkt in der Ebene. Ermitteln lässt sich der Punkt, indem Sie drei Einheiten entlang der X-Achse nach rechts gehen und drei Einheiten entlang der Y-Achse nach oben. Im Raum müssten Sie zusätzlich die Z-Achse beachten.

Im Gegensatz zum Punkt, gibt ein Vektor keine absolute Position an, sondern eine Verschiebung. Ein Vektor besitzt immer eine Länge, eine Richtung sowie einen Richtungssinn.

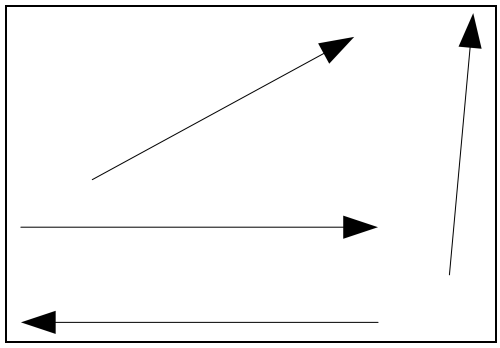

Abbildung 1.4: Vektoren

In welche Richtung der Vektor zeigt, wird durch dessen Orientierung bestimmt. Den Richtungssinn definiert der Pfeil des Vektors. Am unteren linken Rand sind zwei Vektoren mit derselben Länge und derselben Richtung aber einem unterschiedlichen Richtungssinn zu sehen.

Da DirectX Vektoren verwendet, um eine Position zu bestimmen, müssen diese irgendeinen Bezug haben. Ohne jeglichen Bezug würden Vektoren wahllos angeordnet werden können. Aus diesem Grund ist die Verschiebung immer vom Ursprung ausgehend. Die mathematisch korrekte Bezeichnung für solch einen Vektor lautet: Orts-Vektor. Abbildung 1.5 demonstriert eine Verschiebung vom Origo aus.

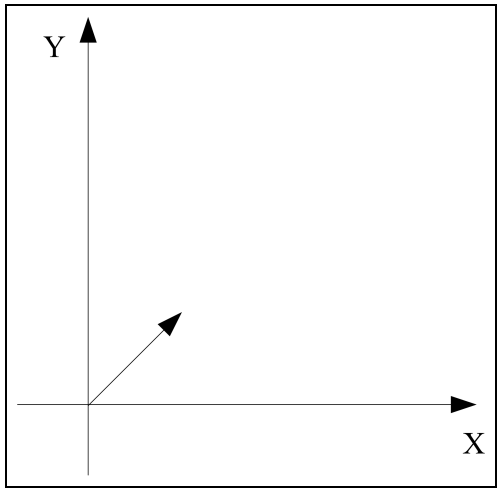

Abbildung 1.5: Ein Ortsvektor realisiert eine Verschiebung von einem Ursprung

1.3.2 Vektor-Darstellung

Zwei Darstellungen von Vektoren finden in der Mathematik häufigen Einsatz: Zum einen die Koordinatendarstellung und zum anderen die Komponentendarstellung. Letzteres basiert auf den Einheitsvektoren i, j und k. Dabei handelt es sich um Einheitsvektoren, die jeweils eine Einheit auf der entsprechenden Achse repräsentieren.

In der unteren Abbildung sehen Sie beide Darstellungsformen. Der untere dargestellte Vektor entspricht dem Einheitsvektor i.

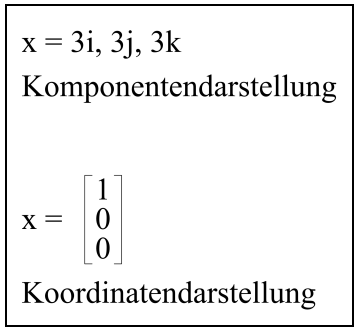

Abbildung 1.6: Darstellungsformen eines Vektors

Mathematiker kennzeichnen Kleinbuchstaben als Vektor, indem sie über diesem einen Pfeil zeichnen oder den Buchstaben unterstreichen. Eine einheitliche Form existiert nicht. Außerdem ist es üblicher runde Klammern zu verwenden. In diesem Buch werden Vektoren nach dem obigen Schema dargestellt.

1.3.3 Der Null-Vektor

Da Null-Vektoren keine Verschiebung angeben, ist dessen Name eigentlich unangebracht. Jener Vektor gibt keine Richtung an und besitzt auch keine Länge (diese würde 0 entsprechen, daher auch Null-Vektor).

1.3.4 Die Länge eines Vektors

Ausgenommen des Null-Vektors, besitzt jeder Vektor eine Länge. Der Satz des Pythagoras bewährt sich auch an dieser Stelle, um die Länge zu berechnen.

Warum kommt hier der Satz des Pythagoras in Frage, wenn es sich doch nicht um ein rechtwinkliges Dreieck handelt? Zwar existiert hier kein Dreieck, doch durch kleine, gedachte Erweiterungen ist ein Dreieck schnell konstruiert. Zum leichteren Verständnis zeigt Abbildung 1.7 ein Beispiel.

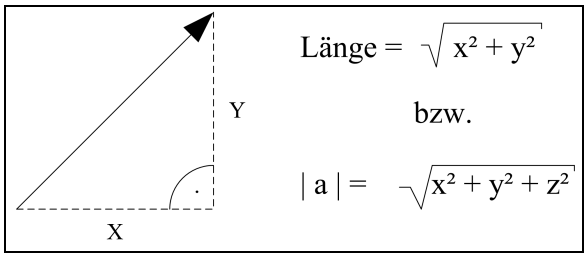

Abbildung 1.7: Berechnung der Länge eines Vektors

Der Begriff „magnitude“, häufig in der englischen Literatur anzutreffen, ist mit der Länge eines Vektors gleichbedeutend. Anders ausgedrückt handelt es sich um den Betrag von a.

Die Abstände zweier Vektoren lassen sich auf demselben Weg errechnen. Dazu subtrahieren Sie beispielsweise vom x-Wert des zweiten Vektors, den x-Wert des ersten. Das Ergebnis nehmen Sie zum Quadrat und bilden die Summe mit den anderen Komponenten, die gleichermaßen berechnet werden. Nachdem die Wurzel gezogen ist, liegt der Abstand beider Vektoren zueinander vor.

1.3.5 Einheitsvektoren

Manche Situationen fordern lediglich eine Richtungsangabe, wobei der Griff zu einem Einheitsvektor geradezu gleichbedeutend ist. Einheitsvektoren sind all jene Vektoren, dessen Länge einer Einheit entspricht. Wie Sie aus einem vorhandenen Vektor einen Einheitsvektor errechnen können, stellt die Abbildung 1.8 schematisch dar.

$$x_E = \frac{x}{|x|}$$

Abbildung 1.8: Errechnung des Einheitsvektors aus einem bestehenden Vektor

Die Berechnung des Einheitsvektors findet im Fachjargon den Begriff der Normalisierung.

1.3.6 Addition und Subtraktion

Vektoren lassen sich addieren und subtrahieren. Beide Methoden bewirken, dass ein neuer Vektor entsteht, wobei Orientierung und/ oder Länge möglicherweise variieren.

$$\begin{bmatrix} 4 \\ 1 \\ 6 \end{bmatrix} + \begin{bmatrix} 0 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 9 \end{bmatrix}$$

Abbildung 1.9: Addition zweier Vektoren

Sowohl Additionen als auch Subtraktionen werden auf Vektoren angewendet, indem die Operation auf dessen Komponenten („komponentenweise“) ausgeführt wird.

Interessant ist die geometrische Interpretation, wie sie Abbildung 1.10 aufzeigt.

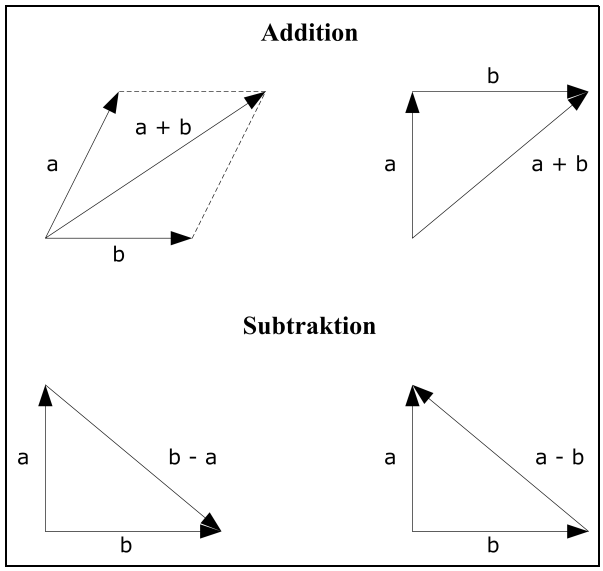

Abbildung 1.10: Auswirkungen der Addition und Subtraktion

1.3.7 Skalar-Multiplikation

Eine Multiplikation zweier Vektoren ergibt keinen neuen Vektor, sondern ist nicht definiert. Die Multiplikation mit einer reellen Zahl, einem Skalar, ist möglich. Ein Skalar beeinflusst die Länge und teilweise auch die Orientierung eines Vektors. Streng mathematisch genommen, hat ein Skalar lediglich Einfluss auf den Richtungssinn, nicht aber auf die eigentliche Richtung.

Bei den Schreibweisen gelten die normalen mathematischen Regeln, wie auch beim Ausklammern. Abbildung 1.11 zeigt ein Beispiel, welchen Einfluss der Skalar -½ auf den Vektor nimmt.

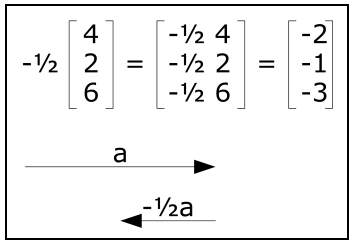

Abbildung 1.11: Multiplikation mit einer reellen Zahl

1.3.8 Skalar-Produkt (Punkt-Produkt)

Mit Hilfe des Skalar-Produkts lässt sich der Winkel ermitteln, der durch zwei Vektoren aufgespannt wird. Doch zunächst einmal soll das Augenmerk auf die Berechnung des Punkt-Produkts gerichtet werden.

Multiplizieren Sie die korrespondierenden Komponenten und bilden Sie im Anschluss daran die Summe aus allen Produkten. Abbildung 1.12 zeigt die daraus resultierende Formel.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \cdot \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = x_1 * x_2 + y_1 * y_2 + z_1 * z_2$$

Abbildung 1.12: Berechnung des Punkt-Produkts

Außerdem gilt die Gleichung:

```
a * b = |a| * |b| * cos(a, b)
```

Durch Umstellen dieser Gleichung lässt sich der aufgespannte Winkel errechnen (siehe Abbildung 1.13).

$$\sphericalangle(a, b) = \text{acos}\left(\frac{x_1 * x_2 + y_1 * y_2 + z_1 * z_2}{|a| * |b|}\right)$$

Abbildung 1.13: Berechnung des durch zwei Vektoren aufgespannten Winkels

Bilden Sie das Skalar-Produkt zweier senkrecht zueinander liegenden Vektoren, erhalten Sie als Ergebnis den Wert 0. Zusammenfassend lässt sich somit folgende Formel aufstellen unter der Bedingung, dass a senkrecht zu b ist.

```
a * b = 0
```

1.3.9 Kreuz-Produkt (Vektor-Produkt)

Ein Vektor der senkrecht auf anderen Vektoren steht oder orthogonal zu einer Ebene positioniert ist, wird häufig als Normale bezeichnet. Normalen dürften Ihnen noch aus dem Mathematik-Unterricht bekannt sein. Oftmals galt es eine Tangente zu ermitteln, die den Graph an der Stelle x berührt. Darüber hinaus sollte die Normale der Tangente berechnet werden. Selbstverständlich ging das bereits ohne Kenntnis von der Vektorrechnung.

Normalen spielen in der 3D-Programmierung vor allem für die Lichtberechnung eine bedeutende Rolle. Mittels eines Normal-Vektors ermittelt Direct3D den Winkel des einfallenden Lichts und der Farbwert wird entsprechend manipuliert. Insofern zwei Vektoren gegeben sind, lässt sich mit dem Kreuz-Produkt ein neuer Vektor errechnen, der zu den beiden anderen Vektoren senkrecht ist. Abbildung 1.14 demonstriert den Rechenweg.

$$a \times b = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix}$$

Abbildung 1.14: Berechnung des Vektor-Produkts

1.3.10 Der Vector3-Datentyp

DirectX enthält bereits eine Struktur für 3D-Vektoren: `Vector3` (zu finden im Namensraum `Microsoft.DirectX`). Neben den drei Komponenten `x`, `y` und `z` verfügt die Struktur über einige nützliche Methoden, mit dessen Hilfe Sie beispielsweise die Länge eines Vektors ermitteln können.

Methode	Beschreibung
Add()	Addiert zwei Vektoren und gibt einen 3D-Vektor zurück. Existiert als statische Methode und als Instanzmethode.
Cross()	Errechnet das Kreuz-Produkt zweier Vektoren und gibt den resultierenden Vektor zurück.
Dot()	Gibt das Skalar-Produkt (Punkt-Produkt) als Float-Wert zurück.
Length()	Liefert die Länge des Vektors als Float-Wert. Existiert als statische Methode und als Instanzmethode.
LengthSq()	Liefert das Quadrat der Länge eines Vektors als Float-Wert. Existiert als statische Methode und als Instanzmethode.
Multiply()	Führt eine Skalar-Multiplikation durch. Das Ergebnis ist vom Typ Vector3.
Normalize()	Normalisiert den gegebenen Vektor (Es wird ein Einheitsvektor erzeugt). Existiert als statische Methode und als Instanzmethode.
Subtract()	Subtrahiert zwei Vektoren voneinander. Liefert ein Ergebnis vom Typ Vector3. Existiert als statische Methode und als Instanzmethode.

Tabelle 1.1: Methoden des Vector3-Datentyps

1.4 Matrizen

Hinter dem Begriff Matrix (Plural: Matrizen) verbirgt sich ein rasterartiger Aufbau von Skalaren (reellen Zahlen). Folglich besitzt eine Matrix Spalten und Zeilen. Ein Vektor entspricht einem Array von Skalaren. Analog dazu entspricht eine Matrix einem Array von Vektoren. Matrizen dienen der Manipulation von Vektoren oder anderer Matrizen. Hinter diesem evtl. für Sie neuartigen Konstrukt steckt eine vereinfachte Schreibweise einer mathematischen Operation.

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

Abbildung 1.15: Eine 3x3 Matrix

Die Anzahl der Zeilen und Spalten bilden den Namen der Matrix und teilen somit dessen Größe mit. Die Matrix aus Abbildung 1.15 ist eine 3x3 Matrix, da sie drei Zeilen und drei Spalten enthält.

Der Index m_{ij} gibt das Element in der Matrix M an, dessen Zeilennummer i und dessen Spaltennummer j entspricht. Matrizen mit derselben Anzahl von Zeilen und Spalten werden als quadratische Matrizen bezeichnet.

1.4.1 Diagonal- und Identity-Matrix

Wenn alle Elemente mit denselben Zeilen- und Spaltennummern einen Wert ungleich 0 und alle restlichen Einträge den Wert 0 enthalten, wird von einer Diagonal-Matrix gesprochen. Eine besondere Diagonal-Matrix stellt jene aus Abbildung 1.16 dar. Alle Elemente der Diagonalen enthalten den Wert 1. Diese Matrix hört auf den Namen Einheitsmatrix (Identity-Matrix).

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Abbildung 1.16: Die Einheitsmatrix

1.4.2 Vektoren als Matrix dargestellt

Wenn Sie einen Blick auf eine der Vektoren aus Abschnitt 1.5 werfen, stellen Sie fest, dass eine gewisse Ähnlichkeit zwischen einem Vektor und einer Matrix besteht. Vektoren lassen sich ebenfalls als Matrix betrachten. Eine 1xn-Matrix entspräche einem sog. Zeilen-Vektor. Als Spalten-Vektor wird eine nx1-Matrix angesehen.

$$\begin{bmatrix} 4 & 1 & 6 \end{bmatrix}$$

Zeilen-Vektor

$$\begin{bmatrix} 4 \\ 1 \\ 6 \end{bmatrix}$$

Spalten-Vektor

Abbildung 1.17: Zeilen- und Spalten-Vektoren

1.4.3 Transposition

Transpositionen werden durch ein hochgestelltes T gekennzeichnet und bewirken, dass Zeilen und Spalten vertauscht werden. Anders ausgedrückt spiegelt eine Transposition die Zeilen und Spalten an dessen Hauptdiagonalen. Das Resultat einer Transposition hört auf den Namen transponierte Matrix.

Mit Hilfe dieses Konstrukts kann ein Spaltenvektor als Zeilenvektor ausgeschrieben werden. Dass es sich eigentlich um einen Spaltenvektor handelt, verdeutlicht das Transpositionszeichen.

$[2 \;\; 1 \;\; 3]^T$

Abbildung 1.18 zeigt eine etwas größere transponierte Matrix und dessen Original.

$$\begin{bmatrix} 4 & 3 & 9 \\ 1 & 5 & 2 \\ 2 & 7 & 5 \end{bmatrix}^T = \begin{bmatrix} 4 & 1 & 2 \\ 3 & 5 & 1 \\ 9 & 2 & 5 \end{bmatrix}$$

Abbildung 1.18: Transposition einer 3x3 Matrix

1.4.4 Multiplikation mit einem Skalar

Analog zu den Vektoren verhält sich eine Matrix bei einer Skalar-Multiplikation. Der Faktor wird mit jeder Komponente der Matrix multipliziert. Die Summe der Teilergebnisse bildet die neue Matrix.

$$M = k \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} = \begin{bmatrix} k\,m_{11} & k\,m_{12} & k\,m_{13} \\ k\,m_{21} & k\,m_{22} & k\,m_{23} \\ k\,m_{31} & k\,m_{32} & k\,m_{33} \end{bmatrix}$$

Abbildung 1.19: Skalar-Multiplikation mit einer Matrix

1.4.5 Matrix-Multiplikation

Im Hinblick zur Skalar-Multiplikation, gestaltet sich eine Multiplikation zweier Matrizen etwas schwieriger. Zudem unterliegt sie einer wichtigen Voraussetzung.

Hinweis

Eine Multiplikation zweier Matrizen ist dann undefiniert, wenn die Anzahl der Spalten, der Matrix A, nicht mit der Anzahl der Zeilen, der Matrix B, übereinstimmt.

Aus zwei Matrizen, `rxn` und `rxc`, ergibt sich nach deren Multiplikation ein Produkt mit den Dimensionen `rxc`.

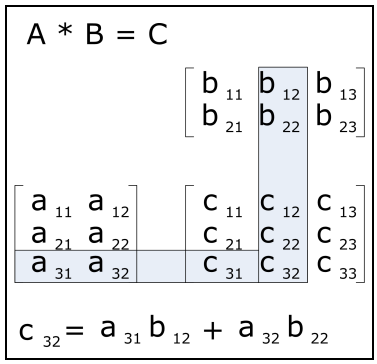

Abbildung 1.20: Matrizen miteinander multiplizieren

Die Vorgehensweise zum Ermitteln der einzelnen Elemente, einer Matrix `C`, demonstriert Abbildung 1.20. Der Index c_{ij} gibt an, dass die Elemente der Reihe `i` der Matrix `A` mit den Elementen der Spalte `j` der Matrix `B` multipliziert werden. Addieren Sie anschließend die Produkte.

Im Gegensatz zur Multiplikation einfacher reeller Zahlen, ist bei den Matrizen die Reihenfolge zu beachten. Das Produkt `AB` entspricht nicht dem Produkt `BA`. Im zweiten Kapitel wird dies an einem Beispielprogramm praktisch belegt.

1.4.6 Multiplikation von Matrizen und Vektoren

Da ein Vektor ebenfalls als eine Matrix, mit einer Zeile bzw. einer Spalte, angesehen werden kann, lässt sich ebenfalls eine Multiplikation durchführen. Es gelten die Regeln aus dem vorherigen Abschnitt. Abbildung 1.21 zeigt die vier Möglichkeiten einer Vektor-Multiplikation.

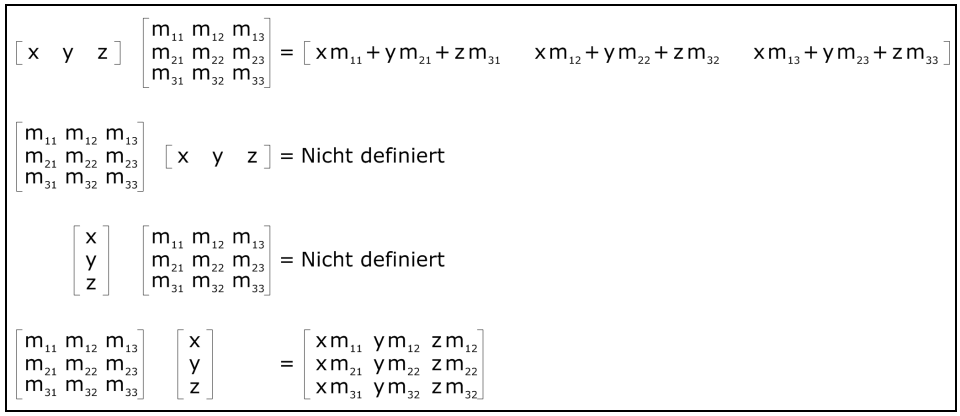

Abbildung 1.21: Multiplikation von Matrizen und Vektoren

1.5 Vertices und Primitive

Ein Vertex ist ein Punkt im Raum, der neben den reinen Koordinaten auch zusätzliche Informationen wie beispielsweise einen Farbwert beinhalten kann.

Virtuelle Welten bestehen aus einer Vielzahl von Vertices. Durch die Ansammlung vieler „Punkte“ entsteht jedoch noch keine Fläche. Erst durch das Verbinden der Vertices zu Linien oder Dreiecken entsteht ein sichtbares Objekt. Üblich ist die Zusammenfassung der Vertices zu einem Dreieck, welches eine glatte Oberfläche bildet. Eine Primitive ist eine Ansammlung vieler Vertices die entweder beziehungslos sind oder durch unterschiedliche Beziehungen Linien und Dreiecke ergeben können. Direct3D unterstützt fünf Arten von Primitiven, die Sie im zweiten Kapitel zu Gesicht bekommen, sobald mit der Programmierung begonnen wird.

Dreiecke dienen in der dreidimensionalen Grafikprogrammierung der Modellierung von Körpern jeglicher Art. Ob der geometrische Körper ein simpler Würfel oder ein komplexes Abbild eines Menschen ist, spielt keine Rolle. Allein entscheidend ist die Anzahl der verfügbaren Dreiecke. Je mehr Polygone (Dreiecke) Ihnen zur Verfügung stehen, umso detaillierter lässt sich ein Objekt darstellen.

1.6 Koordinatentypen

1.6.1 Objekt-Koordinaten

Alle Vertices eines 3D-Körpers liegen relativ zu dessen Ursprung vor. Nehmen Sie einmal an, dass Sie eine Stadt gegründet haben und von dieser nun ein Stadtplan erstellt werden soll. Sie definieren den Marktplatz als das Zentrum (den Ursprung). Alle Hauptstraßen verlaufen zum Markt und dessen Hausnummern richten sich nach der Entfernung zum Stadtmittelpunkt. So vergrößern sich die Hausnummern, je weiter man sich vom Markt entfernt. Zur Erinnerung, der Marktplatz (Ursprung) hatte die Koordinaten (0, 0, 0).

1.6.2 Welt-Koordinaten

Neben Ihnen existiert ein weiterer Gründer einer Stadt. Die Aufgabe eines Dritten besteht nun darin, eine Karte zu entwerfen auf der beide Städte und deren Lage zueinander erkennbar sind. Zur Lösung dieser Aufgabe muss als Erstes ein neuer Ursprung festgelegt werden. Der Einfachheit halber belassen wir den Origo bei Ihrem Marktplatz. Allerdings tritt dabei ein Problem auf: Sämtliche Koordinaten der zweiten Stadt müssen übersetzt werden, weil deren Ursprung in der neuen Karte keine Gültigkeit mehr besitzt. Durch Verschieben und Rotieren wird deren Stadtplan in der neuen Karte eingefügt. Dennoch sind alle Verhältnisse in dieser Stadt erhalten geblieben.