Jens Konerow

Spieleentwicklung mit dem Microsoft XNA Framework

Einstieg und professioneller Einsatz

entwickler.press

Jens Konerow
Spieleentwicklung mit dem Microsoft XNA Framework
ISBN: 978-3-86802-224-7

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

Kein anderer Bereich im IT-Umfeld hat eine derart große Anziehungskraft wie die Programmierung von Computerspielen. Nahezu jeder gestandene, aber auch viele angehende Programmierer träumen davon, selbst einmal ein Spiel nach den eigenen Vorstellungen zu kreieren. Zudem unterscheidet sich jener Markt grundlegend von allen anderen Bereichen der IT-Branche. Dort kommt in der Regel einer beliebigen Firma ein Auftrag zu, der an ein vorhandenes Budget geknüpft ist. Irgendwann erhält der Auftraggeber eine Testversion und schließlich das endgültige Produkt. Das Budget ist von vornherein vorhanden und die Softwareschmiede läuft nicht Gefahr, durch das Produkt in den Ruin getrieben zu werden.[1]

Im Game-Development-Gewerbe Fuß zu fassen ist weitaus schwieriger. Am Anfang eines jeden Projekts steht die Idee, die letztlich in ausführliche Konzepte mündet. Anschließend gilt es, einen lauffähigen Prototypen zu entwickeln und mit diesem gewappnet alle in Frage kommenden Publisher aufzusuchen, um deren Produktmanager von der Genialität der Spielidee zu überzeugen. Erst nachdem ein Publisher die Zusage erteilt hat, eröffnet sich der Softwareschmiede die Möglichkeit, zum Beispiel an eines der begehrten Developer Kits der Xbox 360 zu kommen. Ein Xbox 360 Developer Kit umfasst eine spezielle Konsole, auf der nicht signierte Anwendungen ausführbar sind sowie ein Software Developer Kit (SDK). Developer Kits benötigt man, um nicht zertifizierte Software auf der Konsole auszuführen. Vermarktete Spiele (für die Xbox 360) sind von Microsoft zertifiziert und erhalten dadurch Zugriff zum System. Jedes verkaufte Computerspiel finanziert zugleich die Hardware der Konsole, da der Kaufpreis der Xbox 360 weit unter dem Produktionskosten liegt. Der Besitz eines solchen Developer Kits ist also die Voraussetzung dafür, Computerspiele für Konsolen zu entwickeln. Hobbyprogrammierern bleibt dieses Tor verschlossen. Das liegt nicht nur am Unkostenfaktor, der sich im Rahmen eines Kleinwagens bewegt, sondern auch an der sorgfältigen Selektion der Empfänger durch Microsoft.

Mit dem XNA Framework schließt Microsoft nun diese Plattformlücke, sodass Hobbyprogrammierer in der Lage sind, verwalteten Code auf der Xbox 360 auszuführen, ohne dass das Spiel den Zertifizierungsprozess durchläuft. Es fällt lediglich eine jährliche Schutzgebühr an und die .NET-Anwendungen unterliegen einigen Restriktionen. Dazu später mehr.

Ende 2008 öffnete Microsoft dann das Tor für alle Entwickler, Computerspiele für die Xbox 360 zu entwickeln und diese über den Xbox-LIVE-Marktplatz (kostenpflichtig) zu vertreiben.

[1] Wenn man von Software für den Massenmarkt einmal absieht

V.1 An wen richtet sich dieses Buch?

Grundlegend richtet sich dieses Buch an all diejenigen, die davon träumen, ein eigenes Computerspiel zu designen und auf dem heimischen Fernseher auf der Xbox 360 oder aber auf dem PC zu genießen. Sowohl Anfänger als auch Umsteiger werden wertvolle Referenzen, Tipps und Techniken in diesem Buch finden, die dabei helfen, eigene Computerspiele erfolgreich umzusetzen. Als einzige Voraussetzung gelten grundlegende Kenntnisse in der Programmiersprache C# und ein Hauch von Ergeiz, denn die Entwicklung eines ansprechenden Computerspiels ist nicht in zwei bis drei Tagen erledigt.

Insbesondere grafische und physikalische Sachverhalte sind an mathematische Konstrukte gebunden. Ein gesundes Verständnis der Mathematik, speziell der linearen Algebra, ist erforderlich, damit Sie Ihre Ideen konsequent umsetzen können. Sollte die Schul- oder Studienzeit der Vergangenheit angehören, so empfiehlt es sich, sich ein Buch zum Thema der linearen Algebra zur Auffrischung der Materie zuzulegen.

V.2 Ansprüche an den Computer

Hinsichtlich der Hardware sollte Ihr Computer mit einer DirectX-9.0c-fähigen Grafikkarte ausgestattet sein, der mindestens das Shader Model 1.1 ein Begriff ist. Ansonsten definiert die Entwicklungsumgebung Visual C# 2008 Express bzw. Visual Studio 2008 das absolute (theoretische) Minimum. Das entspricht in etwa einem Prozessor mit mindestens 600 MHz, 192 MB RAM und ein wenig freie Festplattenkapazität. Aus Erfahrungswerten heraus reduziert ein Arbeitsspeichervolumen von 1024 MB bis 2048 MB die Wartezeiten enorm. Idealerweise liegt auch die Taktfrequenz der CPU in höheren Regionen. CPU, Arbeitsspeicher und Grafikkarte sollten also nicht unbedingt dem Minimum genügen, müssen aber nicht den High-End-Modellen entsprechen. Selbstverständlich benötigen Sie eine Xbox-360-Spielkonsole, wenn die Beispiele auch auf dieser ausgeführt werden sollen. Wobei erwähnt werden muss, dass das Core System nicht ausreicht, da diese Version nicht über eine Festplatte verfügt. XNA-Framework-Applikationen stellen den Anspruch, dass zum einen eine Festplatte in dem Gehäuse der Konsole residiert und zum anderen ein Xbox-Live-Account angelegt sowie eine XNA-Creators-Club-Mitgliedschaft abgeschlossen wurde. Doch dazu später mehr. Sofern Sie nicht im Besitz der Konsole sind, bleibt die Möglichkeit, die Beispiele auf dem PC auszuführen.

Insbesondere die Shader-Programmierung stellt hohe Anforderungen an Ihre Grafikkarte. Zwar lassen sich mit dem Shader Model 1.1 die grundlegenden Probleme lösen, aber um alle Beispiele in diesem Buch nachzuvollziehen, empfiehlt sich eine Grafikkarte mit Shader Model 2.0 bzw. Shader Model 3.0 (etwa eine Geforce 7600 GS).

Softwaretechnisch ist es erforderlich, dass Ihr System DirectX 9.0c, Visual C# 2008 Express bzw. Visual Studio 2008 und das XNA Game Studio vorweisen kann. XNA Game Studio ist eine Art Addon für Visual C# 2008 Express bzw. Visual Studio 2008. Microsoft bietet alle Komponenten zum kostenlosen Download auf deren Homepage an.

V.3 Website

Sämtliche Codebeispiele dieses Buchs stehen auf der Internetseite des Verlags (*www.entwickler-press.de/XNA*) und auf der Seite des Autors (*www.jenskonerow.de*) zum kostenlosen Download zur Verfügung. 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 sich nicht, eine E-Mail an die Adresse *webmaster@jenskonerow.de* zu schicken.

Eventuelle Korrekturen oder zusätzliche Informationen finden Sie auf der Website des Autors unter *www.jenskonerow.de*. Regelmäßige Informationen rund um das Thema Spieleentwicklung finden Sie in seinem Blog unter *http://blog.jenskonerow.de*.

V.4 Bildergalerie

Einige der im Buch gezeigten Grafiken machen eine farbige Darstellung nötig. Deshalb finden Sie unter *www.entwickler-press.de/xna-galerie* eine Auswahl der im Folgenden gezeigten Bilder in größerer Auflösung und in Farbe.

Jedes dort aufgeführte Bild kann auch direkt über einen shortlink gefunden werden. Der URL setzt sich zusammen aus *www.entwickler-press.de* und der Bildnummer, die in Klammern am Ende der jeweiligen Bildunterschrift steht.

Ein kleines Beispiel:

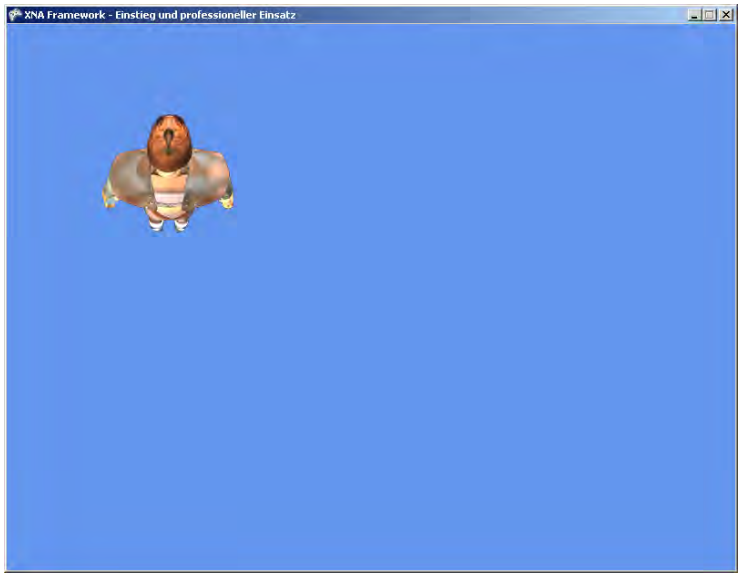

Abbildung 2.5: Die erste grafische Ausgabe: Ein Sprite *[xna001]*

shortlink: *http://entwickler-press.de/xna001*

V.5 Über den Autor

Jens Konerow, geboren am 23. März 1985 in Greifswald, studiert Informatik an der Universität zu Lübeck. Jens Konerow ist mehrfacher Buchautor, freier Softwareentwickler und Entwickler bei der Hamburger Firma KEEP IT SIMPLE Organisation und Information GmbH.

Seit der siebenten Version entwickelt Jens Konerow mit der Multimediaschnittstelle DirectX, die damals erstmals für Visual-Basic-Programmierer verfügbar wurde. Am 1. Juli 2008 wurde der Autor erstmals von Microsoft zum Most Valuable Professional im Bereich XNA/DirectX ernannt. 2009 erfolgte die Renominierung.

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

Mit dem XNA Framework schließt Microsoft eine weitere Plattformlücke, sodass Sie in der Lage sind, eigene Computerspiele auf der Konsole Xbox 360 und auf einem Windows-basierten PC auszuführen.[1] Zusammen mit dem XNA Game Studio 3.1, einem Addon für Visual C# 2008 Express und Visual Studio 2008 bietet Microsoft neue Methoden, die einen schnellen Einstieg garantieren. Zielsetzung des XNA Frameworks war es, neben der „Plattformunabhängigkeit" auch den Entwicklungsprozess zu vereinfachen, der an solch ein Vorhaben geknüpft ist. Beurteilen Sie am Ende des Buchs selbst, ob dem Softwaregiganten sein Vorhaben geglückt ist.

Dieses Kapitel beleuchtet das XNA Framework als „Big Picture", begleitet Sie bei der Installation und klärt die wichtigsten Begriffe, denen Sie zu Beginn begegnen werden.

1.1 Installation

Selbstverständlich stellt die Installation der Tools für einen Entwickler keine Herausforderung dar. Dennoch sei darauf hingewiesen, dass es ratsam ist, eine bestimmte Reihenfolge zu beachten. Installieren Sie die Entwicklungsumgebung Visual C# 2008 Express bzw. Visual Studio 2008 immer vor dem XNA Game Studio 3.1, da letztere Komponente eine Ergänzung für die jeweilige Entwicklungsumgebung darstellt. Um etwaigen Problemen aus dem Weg zu gehen, sollte ebenfalls das DirectX SDK oder alternativ die DirectX 9.0c Runtimes vor dem XNA Game Studio 3.1 installiert werden.

Nachdem das System um alle Komponenten ergänzt wurde und Sie XNA Game Studio 3.1 gestartet haben, liegt eine etwas spärliche Entwicklungsumgebung vor Ihnen, die sich im Groben nicht von Visual C# 2008 Express bzw. Visual Studio 2008 unterscheidet. Lediglich beim Anlegen eines neuen Projekts fällt auf, dass neue Projekttypen die standardmäßigen Typen abgelöst haben (Abschnitt 1.2).

[1] Ab dem XNA Game Studio 3.0 sind Sie zusätzlich in der Lage, für Zune Devices zu entwickeln.

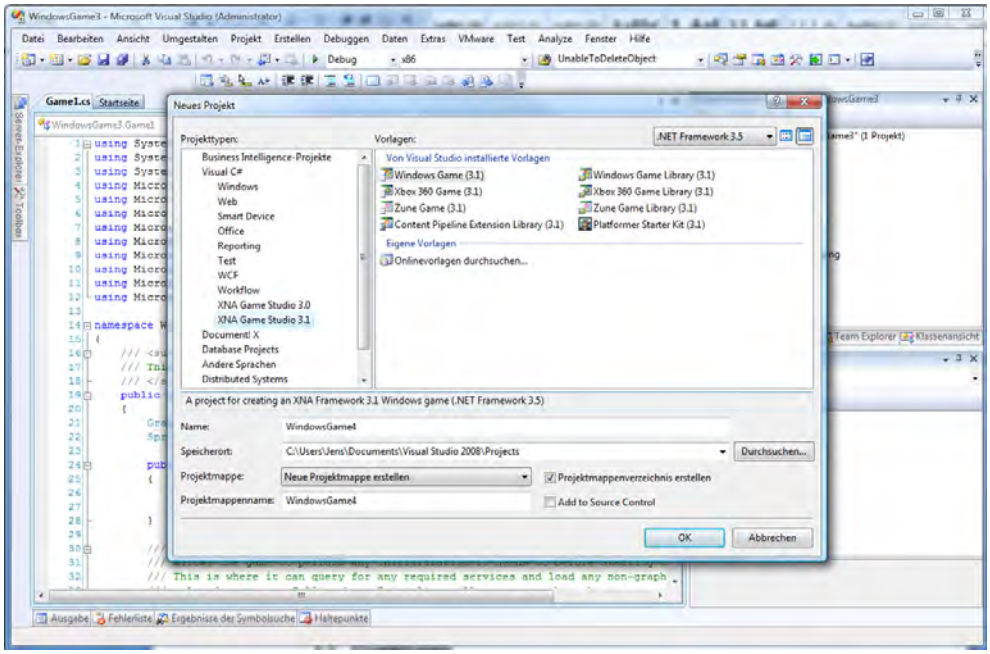

Abbildung 1.1: XNA Game Studio 3.1

1.2 Projekttypen

Insgesamt führt das XNA Game Studio acht neue Projekttypen ein, wobei zwei davon das Starter Kit *Platformer* repräsentieren. Starter Kits sind vollständig lauffähige Anwendungen, auf dessen Grundlage Sie eigene Computerspiele mit dem XNA Framework entwickeln können. Es existieren jeweils zwei Typen, die ein und demselben Zweck dienen. Während *Windows Game* eine Applikation für Windows-Plattformen darstellt, zielt das Pendant namens *Xbox 360 Game* auf die Konsole ab. Projekte, deren Zielsystem der Konsole entsprechen, werden beim Start aus Visual Studio heraus auf die Konsole übertragen, sofern auf der Xbox 360 das XNA Game Studio Connect läuft (Anhang A). *Windows Game Library* bzw. *Xbox 360 Library* sind Projekttypen, die in einer Assembly, also in einer Bibliothek, resultieren.

Neu gegenüber dem XNA Game Studio Express 1.0 (Refresh) ist der Projekttyp *Content Pipeline Extension Library*. Jener Typ stellt Ihnen einen Rahmen zur Verfügung, mit dem Sie Erweiterungen für die Content Pipeline entwickeln können. Was die Content Pipeline ist und welchen Zweck sie erfüllt, erfahren Sie zu einem späteren Zeitpunkt. Ab Version 3.0 stehen zudem die Projekttypen *Zune Game* und *Zune Game Library* zur Verfügung. Mittels jener Projekte sind Sie in der Lage, für die portablen Zune-Geräte zu entwickeln. Allerdings sind jene iPod-Konkurrenten derzeit noch nicht in Deutschland erhältlich, weshalb das Buch nicht auf deren Eigenheiten eingeht.

Vorausgesetzt, Sie beachten einige Regeln, die im Laufe dieses Buchs hervorgehoben werden, bedarf es keiner Änderung am Code, um das Computerspiel sowohl auf dem PC als auch auf der Xbox 360 auszuführen. Erzeugen Sie gleich zu Beginn zwei Projekte, je eins für die entsprechende Plattform, und verknüpfen Sie alle Dateien im Xbox-360-Game-Projekt. Dadurch bleibt Ihr Quelltext in beiden Projekten immer auf demselben Stand.

Abbildung 1.2: XNA Creators Club Website

1.3 XNA Creators Club

Der XNA Creators Club (*http://creators.xna.com*) ist der Dreh- und Angelpunkt der XNA-Community. Microsoft publiziert stetig neue Codebeispiele, Mini-Spiele und Starter Kits. Ferner finden Sie an jener Stelle zusätzliche Hilfe in Form eines Forums mit einer Menge kompetenter Leute. Mitglieder des Entwicklerteams des XNA Frameworks lesen ebenfalls im Forum mit und geben nicht selten wertvolle Tipps rund um die Spieleentwicklung.

Die Inhalte der Seite sind im Großen und Ganzen für jedermann zugänglich. Eine Ausnahme bilden Mini-Spiele und Starter Kits, die als *Premium* gekennzeichnet sind. Diese Daten können Sie nur herunterladen, wenn eine kostenpflichtige Mitgliedschaft im XNA Creators Club vorliegt. Die Premium-Mitgliedschaft ist außerdem Voraussetzung, um Anwendungen auf Basis des XNA Frameworks auf der Konsole auszuführen.

1.4 Das XNA Framework im Überblick

Vorweg sei gesagt, dass das XNA Framework keine Spiele-Engine darstellt, mit der sich im Handumdrehen der nächste Toptitel entwickeln lässt. Stattdessen kann das XNA Framework als eine Art Plattform angesehen werden, die die Kommunikation mit der Hardware dahingehend abstrahiert, dass der Programmierer keine Gedanken an die verwendete Hardware verschwenden muss.

Im Hintergrund existieren eine Handvoll nativer APIs (Application Programming Interface), auf denen ein Framework aufgesetzt ist, das dem Entwickler immer wiederkehrende Aufgaben abnimmt und ein objektorientiertes Modell zur Verfügung stellt. Dem C#-Programmierer stellt das XNA Framework somit ein gewohntes Umfeld bereit.

Abbildung 1.3 zeigt ein „Big Picture“ des XNA Frameworks und gliedert jede Komponente in die dazugehörige (logische) Schicht ein.

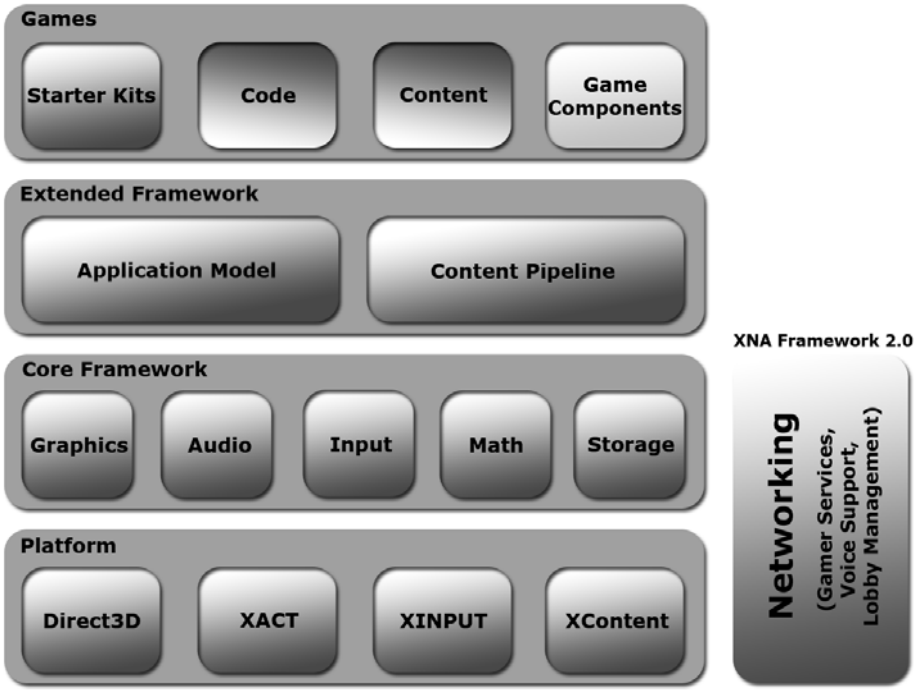

Abbildung 1.3: Schematische Darstellung des XNA Frameworks

1.4.1 Platform

Direct3D

Das Fundament (im Microsoft-Umfeld als „Platform“ bezeichnet) bilden die nativen APIs. Zur Ausgabe zwei- und dreidimensionaler Grafiken greift das XNA Framework im Hintergrund auf Direct3D auf dem PC zu bzw. bedient sich eines Pendants auf der Xbox 360. Jene Schnittstelle beruht auf der DirectX-Version 9.0c, weshalb zumindest die DirectX 9.0c Runtimes auf dem Windows-basierten Computer installiert sein müssen. Auf Seiten der Konsole sind selbstverständlich alle nötigen Komponenten gegeben.

XACT

Als XACT bezeichnet Microsoft eine plattformübergreifende Engine zur Wiedergabe von Audiodaten. Jenen Befehlssatz kann sowohl der PC als auch die Xbox 360 verarbeiten. Während im ersten Release des XNA Frameworks noch keine dreidimensionale Audiowiedergabe möglich war, kann die aktuelle Version auch mit dieser Funktionalität auftrumpfen.

Neben XACT hat Microsoft die Audioausgabe auch durch das SoundEffect-API und die Medienbibliothek ermöglicht. Jene Bibliotheken verzichten auf XACT und führen schneller zur Ausgabe erster Klänge.

XINPUT

Voraussetzung für ein interaktives System ist die Kommunikation mit den entsprechenden Peripheriegeräten. Dazu gehören die Tastatur oder die Maus seitens des Windows-Betriebssystems bzw. das Gamepad seitens der Xbox 360. Zwar können Sie auf der Homepage von Microsoft einen speziellen Treiber herunterladen, um das Gamepad der Konsole auch am PC zu betreiben, dennoch werden PC-Spieler bevorzugt auf Tastatur und Maus zurückgreifen und selten ein Gamepad am Computer angeschlossen haben.

Gleiches gilt bei der Konsole bezüglich einer Tastatur. Die Xbox 360 verarbeitet, sofern vorhanden, die Eingaben einer Tastatur, die via USB an der Konsole angeschlossen ist.

XContent

Anwendungen, die unter Windows laufen, genießen in der Regel nahezu grenzlose Freiheiten. Es ist Ihnen möglich, durch eine .NET-Applikation sämtliche Dateien aller Datenträger zu ermitteln und auszulesen. Auf der Konsole herrschen strengere Richtlinien, weshalb eine spezielle Schnittstelle namens *XContent* die nötige Funktionalität zur Verfügung stellt, mit der Sie Spielstände speichern und auslesen. Direkte Zugriffe auf die Festplatte sind nicht gestattet.

1.4.2 Core Framework

Analog zu Managed DirectX offeriert das sog. Core Framework alle in Abschnitt 1.4.1 genannten Schnittstellen in dessen verwalteter Variante. Allgemein gesagt bietet das Core Framework Klassen, Strukturen und Enumerationen, die erforderlich sind, um Grafik- sowie Audiodaten auszugeben, Peripheriegeräte anzusprechen oder Daten von

einem Speichermedium zu laden bzw. zu speichern. Zudem greift Ihnen diese Schicht bei mathematischen Operationen mit einer entsprechenden Bibliothek unter die Arme (beispielsweise, um Matrixoperationen durchzuführen).

Das Core Framework fungiert, salopp gesagt, als Schnittstelle zwischen der Hardware und einer .NET-Anwendung, die einige Erweiterungen, wie etwa den Support zusätzlicher mathematischer Operationen, in der Hinterhand hat.

1.4.3 Extended Framework

Mit dem XNA Framework wird der Projektstart eines Computerspiels beschleunigt und Sie können sich ab der erste Minute auf die Realisierung des eigentlichen Computerspiels konzentrieren. So die Aussage in sämtlichen Werbepublikationen und -veranstaltungen. Damit dies keine leeren Versprechungen sind, wurde das sog. Extended Framework ins Leben gerufen, das alle regelmäßig anfallenden Aufgaben, wie etwa die Initialisierung der Grafikkarte, übernimmt oder den Importprozess von Ressourcen genau spezifiziert.

Application Model

Das Application Model ist genau jener Teil des XNA Frameworks, der die Entwicklung von Computerspielen vereinfacht. Insbesondere der Start eines neuen Projekts ist einfacher als zu Zeiten von Managed DirectX. Früher waren mehrere Zeilen Code notwendig, um allein die Grafikkarte zu initialisieren. Beim XNA Framework nimmt Ihnen das Application Model im Hintergrund jene Aufgabe ab und prüft zudem, ob das Zielsystem auf den Namen Windows hört. Ist das der Fall, wird ein Fenster erzeugt, in dem das Rendering stattfindet. Liegt stattdessen ein Xbox-360-Projekt vor, so bedarf es keines Fensters, da der Konsole derartige Konstrukte nicht bekannt sind.

Weiterhin gehört zu jedem Spiel ein Game Loop, der die Aktualisierung der Logik und den Rendervorgang pro Frame anstößt. Auch hier traten in der Vergangenheit heftigste Diskussionen auf, wie ein Game Loop effizient implementiert wird.

Frame

Damit das menschliche Auge eine Bewegung als solche wahrnimmt, bedarf es mindestens 25 Bilder pro Sekunde. Ein Frame entspricht einem Bild aus einem Bewegungsablauf.

Content Pipeline

Computerspiele bestehen weniger aus Code als vielmehr aus diversen Ressourcen. Dazu zählen Texturen und 3D-Modelle, aber auch Audiodaten. Ein (zeitaufwändiges) Problem der Entwickler ist die Konvertierung diverser Formate, die vom 3D-Artist, den Grafikern und Sounddesignern zur Verfügung gestellt werden. Bisher oblag den Entwicklern die Aufgabe, die Daten in ein Format zu überführen, mit dem die Anwendung arbeiten kann.

Microsoft offeriert mit der Content Pipeline ein erweiterbares Framework, dessen Hauptkomponenten auf den Namen Content Importer und Content Processor hören.

Findet eine Ressource den Weg in die Content Pipeline, so steht am Ende ein Objekt im Sinne der objektorientierten Programmierung, sodass die Entwickler mit solchen Inhalten genauso umgehen können wie auch mit allen anderen Konstrukten des Programms.

Zudem ist die Funktionalität einer Konsole im Gegensatz zum Betriebssystem Windows sehr eingeschränkt. Die Xbox 360 fordert alle Daten in einem verständlichen Format, weshalb die Content Pipeline sämtliche Ressourcen in das entsprechende Format überführt, bevor die Applikation ihren Weg auf die Konsole findet.

1.4.4 Games

Die letzte Schicht stellen letztlich Ihre eigenen Anwendungen dar. In jene Schicht fließen sowohl Ihr eigener Code als auch andere Ressourcen wie Texturen oder 3D-Modelle mit ein. Zudem stellt Microsoft so genannte Starter Kits bereit. Das sind vollständige Computerspiele, die sich mit wenig Aufwand modifizieren und erweitern lassen (Abschnitt 1.2).

Game Components stellen ebenfalls Sie oder Programmierer aus der Community bereit, sie gewährleisten die schnelle Implementierung der Komponenten im eigenen Projekt. XNA Game Studio stellt ein Item-Template namens *GameComponent* bereit. Das Template erzeugt eine neue Klasse, die von *Microsoft.Xna.Framework.GameComponent* bzw. von *DrawableGameComponent* erbt.

1.5 Das Fundament eines Spiels: Die Game-Klasse

Grundlage eines jeden Spiels stellt eine Instanz der *Game*-Klasse dar. In der Regel existiert pro Anwendung genau eine Instanz jener Klasse, dessen Konstruktor parameterlos ist.

```
using(Game game1 = new Game())
{
   game1.Run();
}
```

Unter Verwendung des Project Templates *Windows Game* bzw. *Xbox 360 Game* liegt zu Projektbeginn eine Klasse vor, die von *Game* erbt und innerhalb der *Program.cs* instanziert wird. Weiterhin nennt jene neue Klasse bereits eine Variabel *graphics* ihr eigen. Die Variable *graphics* gewährt den Zugriff auf das Graphics Device und alle sonstigen Informationen bezüglich der Grafikkarte (Eigenschaft: *GraphicsDevice*). Ein Graphics Device fungiert als Vermittler zwischen Ihrer Anwendung und der Grafikhardware. Für sämtliche Rendering-Operationen benötigen Sie das Graphics Device.

Unter Verwendung des XNA Game Studio Express 1.0 (Refresh) finden Sie zudem eine Variable namens *content*. Hinter der Variablen verbirgt sich ein *ContentManager*-Objekt, über das Sie auf die Inhalte der Content Pipeline zugreifen.

Seit dem XNA Framework 2.0 verfügt die *Game*-Klasse des Weiteren über zwei Eigenschaften namens *GraphicsDevice* und *Content*, die entweder das *GraphicsDevice*-Objekt oder das *ContentManager*-Objekt zurückgeben. Dadurch vereinfacht sich der Zugriff auf jene Elemente aus einer Komponente heraus.

1.5.1 Das ereignisbasierte Modell

Die *Game*-Klasse verfügt über die Methoden *Update()* und *Draw()*, die der Game Loop stets aufruft. Dadurch entsteht der Eindruck eines ereignisbasierten Programmiermodells, das sich wie ein roter Faden durch das komplette XNA Framework zieht. Grund ist die konsequente Umsetzung der .NET Framework Guide Lines.

Nach Anlage eines neuen Projekts sind neben den Methoden *Update()* und *Draw()* per Default drei weitere Methoden überschrieben, deren Bedeutung zumindest in groben Zügen von vornherein klar sein sollte.

- Initialize()
- LoadContent()
- UnloadContent()

Für die *Initialize()*-Methode sind sämtliche Initialisierungsvorgänge angedacht, die nicht mit grafischen Elementen in Verbindung stehen. Zur Initialisierung der grafischen Elemente ist die *Game*-Klasse mit der *LoadContent()*-Methode ausgestattet. Jene Methode ersetzt die aus dem XNA Framework 1.0 (Refresh) bekannte *LoadGraphicsContent()*-Methode. Hintergrund ist, dass jede grafische Komponente mit dem aktuellen Zustand der Grafikkarte gekoppelt ist. Angenommen, Sie laden eine Grafikdatei als Textur, dann existieren jene Daten optimalerweise im Speicher der Grafikkarte. Diverse Faktoren, etwa die Variation der Fenstergröße, können dazu führen, dass das Graphics Device vom sog. Operational State in den Lost State übergeht. Etwas „fahrlässig" gesagt hat das Graphics Device die Verbindung zur Grafikkarte verloren und ist nicht mehr in der Lage, grafische Elemente auf den Monitor zu bringen. Stattdessen wird der Graphics Device neu initialisiert. Manche Elemente müssen deshalb ebenfalls neu initialisiert und an den aktuellen Graphics Device gebunden werden. Das Application Model signalisiert jene Anforderung durch den erneuten Aufruf der *LoadGraphicsContent()*-Methode. Ab dem XNA Framework 2.0 hat sich jener Umstand geändert. Das Graphics Device initialisiert das XNA Framework weiterhin neu, allerdings bleiben die Referenz auf das *GraphicsDevice*-Objekt sowie alle anderen Ressourcen gültig. Allerdings kann es vorkommen, dass der Inhalt mancher Ressourcen nicht wiederhergestellt werden kann, weshalb es Ihnen dann obliegt, die Ressourcen mit neuen Informationen zu füllen. *LoadGraphicsContent()* sollten Sie deshalb nicht mehr verwenden und stattdessen auf *LoadContent()* umsatteln.

Das Pendant hört auf den Namen *UnloadContent()*, dessen Zweck sich aus dem Methodennamen ableiten lässt. Platzieren Sie dort alle nötigen Anweisungen, um alle erdenklichen Aufräumarbeiten durchzuführen. Mehr Hintergrundinformationen zu dieser Thematik liefert das zweite Kapitel.

1.5.2 Der Game Loop

Sinn und Zweck des Game Loops ist es, die Logik sowie den Zeichenvorgang pro Frame erneut anzustoßen. Das Application Model nimmt Ihnen jene Last bereits zu Beginn eines Projekts von den Schultern. Dennoch ist es Ihnen gewährt, einen geringen Einfluss auf den Game Loop zu nehmen. Man unterscheidet zwischen zwei Modi, zwischen denen Sie als Programmierer wählen können.

- Statischer Game Loop
- Variabler Game Loop

Per Default nutzt eine XNA-Anwendung den statischen Game Loop, der die Anzahl der Frames pro Sekunde begrenzt. Genauer gesagt entspricht die maximale Anzahl der Frames der Bildwiederholrate des Monitors. Vorausgesetzt, das System verarbeitet die Anweisungen schnell genug, bleibt die Frame Rate zum Beispiel konstant bei 60 Bildern pro Sekunde. Auch dann, wenn das System über mehr Ressourcen verfügt. Wählen Sie stattdessen den variablen Game Loop, wird die Aktualisierung der Spiellogik direkt nach der Verarbeitung eines Frames erneut angeleiert.

Zuständig für den Modus ist die *IsFixedTimeStep*-Eigenschaft der *Game*-Klasse. Weisen Sie der Eigenschaft den Wert *false* zu, um zum variablen Game Loop überzugehen.

```
//Innerhalb der Ableitung von der Game-Klasse
this.IsFixedTimeStep = false; //Schaltet den variablen Game Loop
                              //aktiv
```

Zudem synchronisiert das XNA Framework im Hintergrund den Game Loop mit dem Monitor. Monitore bauen ein Bild zeilenweise von oben nach unten auf. Die vertikale Synchronisierung (VSync) verhindert, dass die Grafikkarte mit neuen Daten gefüllt werden kann, während der Bildschirm mit der Darstellung der Szene beschäftigt ist. Die Zeit, die erforderlich ist, um den Elektronenstrahl von unten rechts nach oben links zu verschieben, nennt man im Fachjargon Vertical Retrace. Solange eine vertikale Synchronisierung mit dem Monitor geschieht, begrenzt sich die maximale Anzahl von Frames auf die vertikale Bandbreite des Anzeigesystems. Das bedeutet, bei einem Monitor mit 60 Hz erreichen Sie maximal 60 FPS.[2]

Möchten Sie die maximale Anzahl von Bildern pro Sekunde erzielen, müssen Sie neben dem variablen Game Loop der Grafikkarte mitteilen, dass Sie die Synchronisierung unterdrücken möchten. Fügen Sie dazu folgende Zuweisung im Konstruktor der Game-Ableitung ein.

```
graphics.SynchronizeWithVerticalRetrace = false;
```

1.5.3 Die GameTime-Struktur

Von besonderer Bedeutung ist die *GameTime*-Struktur, die sowohl als Parametertyp der *Update()*-Methode als auch der *Draw()*-Methode fungiert. Jener Typ gibt Auskunft darüber, wie viel Zeit seit dem letzten Frame vergangen ist. Diese Daten sind zur Zeitsynchronisation unerlässlich. Animationen, d. h. Bewegungsabläufe in einem Spiel, müssen immer ins Verhältnis zur Zeit gesetzt werden. Andernfalls variiert die Geschwindigkeit, in der die Animationen ablaufen, auf unterschiedlichen Systemen. Starten Sie auf einem modernen Computer ein Computerspiel aus der guten alten Zeit, in der DOS als Betriebssystem vorherrschte. Sie werden feststellen, dass viele Spiele praktisch unspielbar sind, da die Bewegungen übermäßig schnell ausgeführt werden.

[2] LCD-Bildschirme besitzen in der Regel eine vertikale Bandbreite von 60 Hz.

Eine fehlende Zeitsynchronisation wäre in einem modernen Multiplayer-Spiel der Super-GAU. Spieler mit der neuesten Hardware wären klar im Vorteil. Sie könnten schneller rennen und folglich alle taktisch wichtigen Punkte früher erreichen als ihr Gegner.

Hier wiederum spielt das Application Model seine Vorzüge aus. Aufgrund des Game Loops werden pro Frame zunächst die *Update()*-Methode und anschließend die *Draw()*-Methode aufgerufen. Beiden Methoden übergibt der Game Loop eine *GameTime*-Struktur, in der die Zeit verzeichnet ist, die seit dem letzten Frame vergangen ist. Tabelle 1 listet alle essenziellen Eigenschaften der *GameTime*-Struktur. Jede Eigenschaft bedient sich des Rückgabetyps *TimeSpan*.

Eigenschaft	Beschreibung
ElapsedGameTime	Liefert die Zeit, die seit dem letzten Aktualisierungsvorgang verstrichen ist
ElapsedRealTime	Liefert die exakte Zeit, die zwischen dem aktuellen und dem letzten Frame liegt
TotalGameTime	Gibt die Zeit zurück, die seit dem Spielstart vergangen ist
TotalRealTime	Gibt die Zeit zurück, die seit dem Spielstart vergangen ist

Tabelle 1.1: Essenzielle Eigenschaften der GameTime-Struktur

ElapsedGameTime und *ElapsedRealTime* unterscheiden sich in der Regel dann, wenn ein statischer Game Loop im Einsatz ist, da dieser die Anzahl der Bilder pro Sekunde auf ein Maximum beschränkt.

1.5.4 Im Fokus: Die Draw()-Methode

Nachdem Sie ein neues Projekt mittels des Templates Windows Game erstellt haben, finden Sie bereits zwei Zeilen innerhalb der Methode *Draw()* vor (siehe nachfolgendes Listing).

```
protected override void Draw(GameTime gameTime)
{
  graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
  // TODO: Add your drawing code here

  base.Draw(gameTime);
}
```

Die Anweisung *Clear()* stellt sicher, dass sich keine Reste des vorherigen Frames im Back Buffer befinden. Genauer gesagt schreibt die Methode für jeden Pixel denselben Farbwert in den Back Buffer, der als Parameter übergeben wurde.

Beim Back Buffer handelt es sich um ein Offscreen-Surface. Hinter einem Surface verbirgt sich ein sequenziell aufgebauter Speicherbereich. In diesem werden Farbwerte der Pixel gespeichert, um ein Bild darzustellen. Nun kommt dem Back Buffer eine besondere Aufgabe zu:

Am Ende eines Rendervorgangs steht nicht etwa der Monitor als Ziel, sondern ein Surface, wobei meist der Back Buffer Verwendung findet. Erst nachdem die Szene fertig gerendert wurde, gelangt das Bild auf den Monitor (Front Buffer). Warum denn so umständlich? Zwar mag dieses Vorgehen auf den ersten Blick etwas unnötig erscheinen, würden Sie jedoch die grafischen Elemente direkt auf den Monitor zeichnen, entstünde ein lästiges Flackern, da der Benutzer den Aufbau der Szene mitverfolgen kann. Indem Direct3D den Umweg über einen Back Buffer nimmt, bleibt der Aufbau dem Anwender verborgen.

Oftmals liegen die Frame-Raten über den Wiederholungsraten (Refresh Rate) des Monitors. Ein Monitor mit 60 Hz braucht allerdings ca. 1/60 Sekunde, um ein Bild aufzubauen, d. h. theoretisch müssten wir nun abwarten, bis der Frame vollständig aufgebaut ist. Statt den Render-Vorgang zu unterbrechen, kommt uns eben der Back Buffer zugute. Während der Monitor mit dem Aufbau der aktuellen Szene beschäftigt ist, wird der nächste Frame bereits gerendert und in das Offscreen-Surface geschrieben. Anschließend wird der Back Buffer zum Front Buffer und umgekehrt. Jenes Verfahren erhält den Begriff „Page Flipping“.

1.6 Fullscreen

Logischerweise laufen Anwendungen auf der Xbox 360 ausschließlich im Vollbildmodus. Unter Windows ist sowohl ein Fenstermodus als auch ein Vollbildmodus verfügbar.

Prinzipiell genügt die folgend Zuweisung innerhalb des Konstruktors Ihrer *Game*-Klasse, um die Anwendung in den Vollbildmodus zu versetzen.

```
graphics.IsFullScreen = true;
```

Zur Laufzeit bemühen Sie stattdessen die *GraphicsDeviceManager.ToggleFullScreen()*-Methode, wenn Sie zwischen den beiden Modi wechseln möchten.

1.6.1 Das Dilemma mit den Auflösungen

XNA-Framework-Anwendungen, die unter Windows laufen, ist es freigestellt, eine gewünschte Auflösung festzulegen. Weisen Sie den Eigenschaften *PreferredBackBufferWidth* bzw. *PreferredBackBufferHeight* beliebige Werte zu, um die gewünschte Auflösung zu erzielen. Vorausgesetzt, die Grafikkarte und der Monitor unterstützen die Auflösung, kümmert sich das XNA Framework automatisch um alle nötigen Angelegenheiten. Entspricht das Format keiner unterstützten Variante, kommt die nächstgelegene Auflösung zum Einsatz.

```
//Verhältnis von 4:3
graphics.PreferredBackBufferWidth = 1280;
graphics.PreferredBackBufferHeight = 1024;

//Verhältnis von 16:9
graphics.PreferredBackBufferWidth = 1280;
graphics.PreferredBackBufferHeight = 720;
```