Patrick Getzmann, Simon Hackfort, Peter Nowak

Windows Phone 7.5 Refresh

Zusatzkapitel zu Entwickeln für Windows Phone 7.5

Microsoft Press

Patrick Getzmann, Simon Hackfort, Peter Nowak:
Windows Phone 7.5 Refresh. Zusatzkapitel zu Entwickeln für Windows Phone 7.5

Kommentare und Fragen können Sie gerne an uns richten:

Microsoft Press Deutschland
Konrad-Zuse-Straße 1
85716 Unterschleißheim
E-Mail: *mspressde@oreilly.de*

Die Beispieldateien zu diesem Kapitel finden Sie unter http://www.microsoft-press.de/support.asp?s110=465

oder unter msp.oreilly.de/support/2234/723.

ISBN von Entwickeln für Windows Phone 7.5:
Druck-ISBN 978-3-86645-465-1, PDF-eBook-ISBN 978-3-86645-733-1

Satz: Silja Brands, ActiveDevelop, Lippstadt (www.ActiveDevelop.de)
Layout: Gerhard Alfes, mediaService, Siegen (www.media-service.tv)

Windows Phone 7.5 Refresh

In diesem Zusatzkapitel:

Nach der letzten großen Aktualisierung auf die Version 7.5 von Windows Phone (auch bekannt als *Mango*) im Jahr 2011, hat Microsoft diese Version nun erweitert: Windows Phone 7.5 Refresh (vielen auch als *Tango* bekannt).

Mit dieser Aktualisierung wurde der Speicherverbrauch minimiert, sodass das Betriebssystem nun auch auf Geräten ausgeführt werden kann, die nur über 256 MB RAM verfügen. Ein solches Gerät ist beispielsweise das Nokia Lumia 610. Alle vorherigen Geräte verfügten bis dahin über 512 MB RAM.

Diese »kleine« Änderung hat jedoch zur Folge, dass nicht nur der Speicherverbrauch des Betriebssystems, sondern auch der Apps minimiert werden muss, was dazu führt, dass einige Funktionen eingeschränkt oder teilweise gar nicht vorhanden sind.

Dieses Zusatzkapitel beschränkt sich auf Änderungen und Aktualisierungen der API des zugehörigen aktualisierten Windows Phone SDK 7.1.1, welche Sie unter *http://www.microsoft.com/download/en/details.aspx?id=29233 im Internet erhalten.*

Doch wozu wollte man den Speicherverbrauch minimieren?

Neben der Möglichkeit, Reserven zu schaffen, um zukünftig neue Funktionen einbauen zu können, gibt es noch einen weiteren ganz einfachen Aspekt: Stark reduzierte Hardwarekosten.

Zwar ist die Marge zwischen 512 MB und 256 MB Speicher nicht sehr groß, doch die Ersparnis kann in diesem Fall an den Kunden weitergegeben werden. Zusätzlich zu dem reduzierten Speicher können auch etwas leistungsschwächere Prozessoren verbaut werden. Diese bewirken nur eine minimale Verringerung der Leistung, die für den Anwender kaum sichtbar ist, aber in erheblichem Umfang Kosten spart. Somit wurde mit dem Windows Phone 7.5 Refresh-Update die Möglichkeit geschaffen, so genannte *Low-Entry*-Geräte als kostengünstige und massentaugliche Alternative zu den weit verbreiteten Feature-Phones auf den Markt zu bringen und damit eine weltweite Vermarktung anzugehen.

Der neue Emulator

Mit der Update-Installation des Windows Phone SDK 7.1.1 wird das bestehende Emulatorabbild *WM70C1.en-US.bin* bzw. *WM70C1.de.bin* in dem Ordner *%ProgramFiles%\Microsoft SDKs\Windows Phone\v7.1\Emulation\Images* für die Unterstützung von Geräten mit 256 MB Speicher verändert.

Im Auswahlfeld des Bereitstellungsziels innerhalb von *Visual Studio 2010* findet man jetzt insgesamt zwei Einträge zum Starten des Windows Phone Emulators für die jeweilige Speicherkapazität 256 MB und 512 MB. Die Emulator Tools werden von der neuen Emulation der 256-MB-Geräte voll unterstützt.

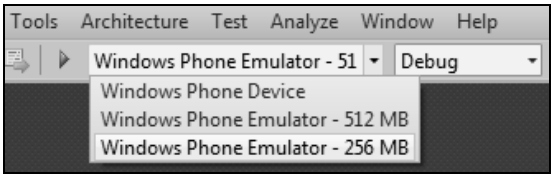

Abbildung Z.1 Auswahl des Bereitstellungsziels im Visual Studio 2010

Das im SDK enthaltene Tool *XapDeploy*, welches sich im Ordner *%ProgramFiles%\Microsoft SDKs\Windows Phone\v7.1\Tools\XAP Deployment* befindet, wurde durch die Update-Installation ebenfalls auf den neuesten Stand gebracht. Das Auswahlfeld für das Bereitstellungsziel wurde hier um die Option *Windows Phone Emulator - 256 MB* erweitert.

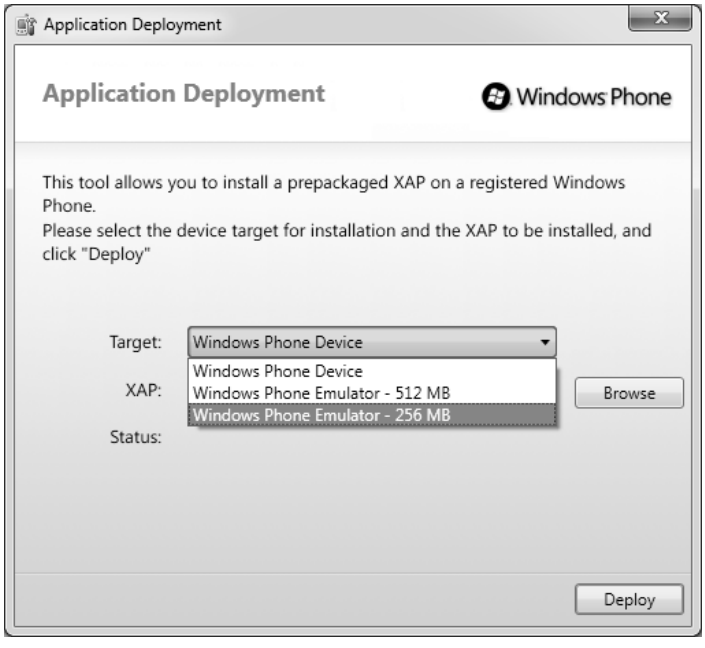

Abbildung Z.2 Auswahl des Bereitstellungsziels in der Anwendung *XapDeploy*

Änderungen beim manuellen Bereitstellen

Die Methode `GetDevices` der Klasse `Platform` aus der Assembly *Microsoft.Smartdevice.Connectivity.dll* liefert jetzt auch die Bezeichnung für den Emulator mit 256 bzw. 512 MB Speicher zurück:

- `Windows Phone Emulator - 256 MB`
- `Windows Phone Emulator - 512 MB`

Abhängig von der Sprache des installierten Windows Phone SDK werden die zuvor genannten Zeichenfolgen um den länderspezifischen Bezeichner wie zum Beispiel `DE` erweitert. Daher sollte bei der Initialisierung des `Device`-Objekts beim Vergleich der Eigenschaft `Name` die Methode `StartsWith` verwendet werden, um nur den Anfang der Zeichenkette zu prüfen.

```
…
// Auswählen des Geräts; hier der Emulator mit 256 MB Speicher
Device WP7Device = WP7SDK.GetDevices().Single(d => d.Name.StartsWith("Windows Phone Emulator - 256 MB"));

// für die Verbindung zum Emulator mit 512 MB Speicher
Device WP7Device = WP7SDK.GetDevices().Single(d => d.Name.StartsWith("Windows Phone Emulator - 512 MB"));
…
```

Listing Z.1 Angepasste Gerätewahl für die manuelle Bereitstellung

Die grafische Oberfläche des Beispiels zum Bereitstellen von Silverlight- und XNA-Anwendungen wurde ebenfalls hinsichtlich der neuen Emulatorkonfiguration mit 256 MB Speicher angepasst. Der komplette Quellcode sowie eine bereits vorkompilierte Version können über die im Anhang A des Buchs angegebene Internetadresse kostenlos bezogen werden.

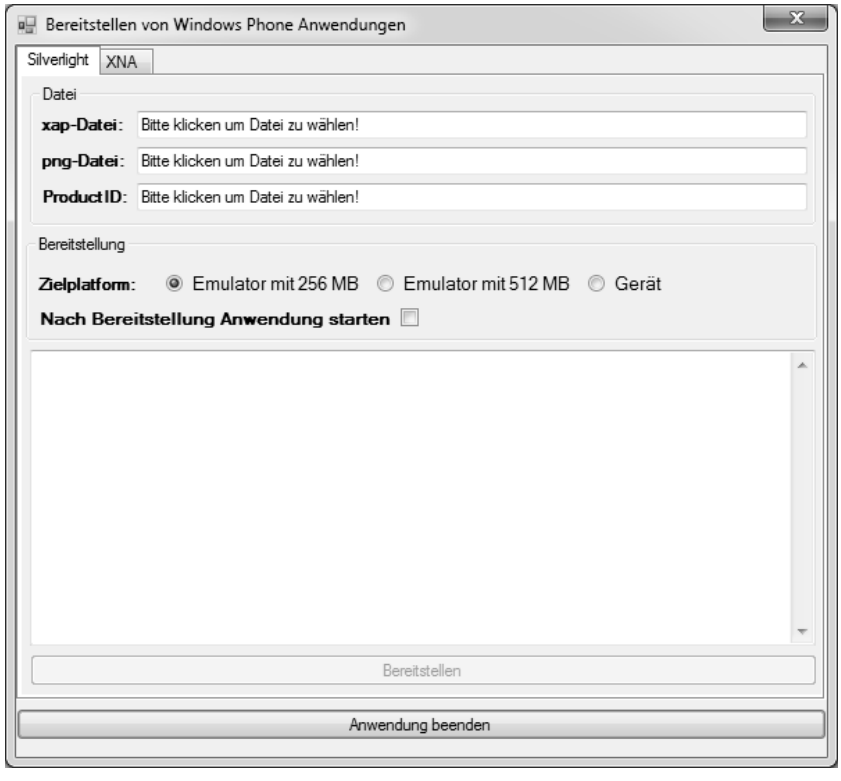

Abbildung Z.3 Anwendung zum manuellen Bereitstellen mit angepasster Emulatorauswahl

Weitere Information zum manuellen Bereitstellen finden Sie in unserem Buch »Entwickeln für Windows Phone 7.5«, Kapitel 4 ab Seite 247.

TIPP Sofern man eine Applikation ohne Einschränkungen für beide Gerätevarianten entwickelt, empfiehl es sich, den Emulator mit 256 MB standardmäßig als Bereitstellungsziel zu wählen oder ein entsprechendes Gerät zu verwenden.

Änderungen und Neuheiten im Windows Phone SDK

Identifizierung des vorhanden Speichers zur Laufzeit

Möchte man zur Laufzeit seiner App feststellen, ob es sich um ein Gerät mit 256 MB handelt, kann man dies über die `GetValue`-Methode der Klasse `DeviceExtendedProperties` abfragen. Dazu wird von *Windows Phone 7.5 Refresh*-Geräten die neue Eigenschaft `ApplicationWorkingSetLimit` bereitgestellt. Über diese Eigenschaft erhält

man das Speicherlimit der Applikation in Bytes. Liegt dieser Rückgabewert vom Typ `long integer` unter `94371840` (90 MB), handelt es sich um ein 256-MB-Gerät mit dem Windows Phone 7.5 Refresh-Update. Liegt der Wert über dem Limit von 90 MB handelt es sich sehr wahrscheinlich um ein 512-MB-Gerät mit Windows Phone 7.5 Refresh-Update. Da diese neue Eigenschaft nur auf Geräten mit dem Windows Phone 7.5 Refresh-Update abgefragt werden kann, muss diese Abfrage in einen `try-catch`-Block eingebettet werden, da es ansonsten zu einer Ausnahme bei einem Windows Phone ohne Refresh-Update kommt. Wenn es bei der Abfrage zu einer Ausnahme kommt, kann man von einem 512-MB-Gerät ausgehen, da, wie bereits in der Einleitung des Kapitels erwähnt, vor dem Windows Phone 7.5 Refresh-Update keine Geräte mit weniger Speicher auf den Markt gekommen sind. Aufgrund dieser Ausnahme bietet es sich auch an, diese Überprüfung nur ein einziges Mal vorzunehmen, und das Ergebnis abzuspeichern. Der folgende Beispielcode zeigt, wie eine solche Überprüfung aussehen könnte:

```
public partial class App : Application
{
   //Zustand applikationsweit verfügbar machen
   public static bool IsLowMem = false;
...
```

Listing Z.2 Globale Variable zur Auswertung

Über die statische Variable `IsLowMem` kann der Zustand applikationsweit abgefragt werden. Gefüllt wird die Variable in der Ereignisbehandlung `Launching`:

```
private void Application_Launching(object sender, LaunchingEventArgs e)
{
   IsolatedStorageSettings settings = IsolatedStorageSettings.ApplicationSettings;
   if (settings.Contains("LowMemDevice"))
   {
      IsLowMem = (bool)settings["LowMemDevice"];
   }
   else
   {
      try
      {
         Int64 bytes= (Int64)DeviceExtendedProperties.GetValue("ApplicationWorkingSetLimit");
         if (bytes < 94371840L)
            IsLowMem = true;
         else
            IsLowMem = false;
      }
      catch (ArgumentOutOfRangeException)
      {
         IsLowMem = false;
      }
      settings["LowMemDevice"] = IsLowMem;
      settings.Save();
   }
}
```

Listing Z.3 Abfrage des Applikationsspeicherzustands

Die Variable kann dann an beliebiger Stelle ausgewertet werden. Im folgenden Beispielcode wird je nach Zustand ein anderer Text in einem `TextBlock`-Steuerelement dargestellt:

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
   if (App.IsLowMem)
      textBlock1.Text = "256 MB";
   else
      textBlock1.Text = "512 MB";
}
```

Listing Z.4 Auswertung des Zustands

Festlegen der minimalen Speicheranforderungen für den Marketplace

Es gibt Situationen, in denen man mit seiner App bewusst keine 256-MB-Geräte unterstützen möchte. Zwar ist es immer ratsam, beide Gerätetypen zu unterstützen. Manchmal jedoch benötigt die App viele Ressourcen, oder sie ist auf Funktionen wie `PeriodicTask` angewiesen und würde auf einem 256-MB-Gerät nicht funktionieren.

In diesem Fall ist es möglich, diese App als eine solche über das Manifest auszuweisen. Dies bewirkt einerseits, dass die zugehörigen Testfälle im Zertifizierungsprozess nicht durchgeführt werden, und andererseits, dass die App nicht im Marketplace für 256-MB-Geräte dargestellt wird.

Um diese Anforderung für Geräte mit mindestens 512 MB einzurichten, öffnen Sie die *WMAppManifest.xml* des Projekts.

Fügen Sie nach der `Capabilities`-Sektion die Folgende ein.

```
<Requirements>
      <Requirement Name="ID_REQ_MEMORY_90" />
</Requirements>
```

Listing Z.5 Über dieses Fragment werden nur noch 512-MB-Geräte unterstützt

Was bedeutet in der Zeichenfolge jedoch die »90«?

Anstatt hier eine »512« anzugeben, die für den zur Verfügung stehenden RAM steht, besagt hier die »90«, dass die Anwendung mindestens 90 MB Applikationsspeicher bei der Ausführung benötigt. Typisch sind hier eher 60 MB für »normale« Apps.

Speichern Sie die XML-Datei und die Anforderung ist gesetzt. Weitere Änderungen sind nicht notwendig.

HINWEIS Wird diese Anforderung in der Manifestdatei hinzugefügt, so ist trotzdem ein Ausführen der App auf dem 256-MB-Emulator möglich. Hier wird diese Einschränkung nicht beachtet.

Optimierung Ihrer Applikationen für 256-MB-Geräte

Im folgenden Abschnitt sind Optimierungen für Ihre Windows Phone-Applikationen und Spiele beschrieben, damit Sie auch auf Geräten mit 256 MB optimale Ergebnisse erzielen. Detailliertere und weiterführende Informationen zu diesen Themen finden Sie auf den Webseiten, die im Abschnitt »Weiterführende Informationen« aufgelistet sind.

Optimierung Ihrer Windows Phone Applikationen

Mit einigen kleinen Änderungen können Sie den Speicherverbrauch reduzieren, beziehungsweise einige Engstellen teilweise auch vollständig umgehen und Ihre App somit für Geräte mit 256 MB optimieren.

Verwendung von Tasks

256-MB-Geräte bzw. der Emulator mit 256 MB unterstützen aufgrund der Speicherintensivität keine Verplanung von Tasks durch den `ScheduleTaskAgent`.

Sobald man ein Objekt vom Typ `PeriodicTask` oder `ResourceIntensiveTask` dem `ScheduledActionService` über die Methode `Add` hinzufügt, wird bei Geräten oder dem Emulator mit 256 MB Speicher zur Laufzeit eine Ausnahme in Form einer `InvalidOperationException` mit der folgenden Meldung erzeugt: *BNS Error: The maximum number of ScheduledActions of this type have already been added.*

Werden die Tasks in der Anwendung verwendet, sollte man vor der Initialisierung eine bedingte Abfrage hinzufügen, sodass der Programmteil mit der Verarbeitung der Tasks von 256-MB-Geräten komplett außer Acht gelassen wird.

Verwendung des WebBrowser-Steuerelements

Nicht für mobile Browser optimierte Webseiten verursachen häufig Speicher- und Performanceprobleme auf den mobilen Endgeräten. Bei Geräten mit wenig Applikationsspeicher treten diese Probleme wesentlich schneller auf, da der zugewiesene Applikationsspeicher nicht nur den generellen Speicher der App beinhaltet, sondern auch den zusätzlichen Speicherverbrauch durch die Webseite. Sollten Sie in Ihrer App Webseiten darstellen wollen, haben Sie zur Optimierung für 256-MB-Geräte zwei Möglichkeiten.

Zu einem können Sie die anzuzeigenden Webseiten vorab überprüfen, und nur die Webseiten zulassen, die Ihren Speicherverbrauch nicht zu sehr belastet.

TIPP Zur Messung des Speicherverbrauchs einer Webseite können Sie die Windows Phone Performance Analysis verwenden. Weiter Informationen zu diesem Thema finden Sie in Kapitel 10 unseres Buchs.

Dazu können Sie in der `Navigating`-Ereignisbehandlung der `WebBrowser`-Klasse den Aufruf von speicherkonsumierenden Webseiten für 256-MB-Geräte mithilfe einer Bedingung, die die aufzurufende Internetadresse zuvor prüft, unterbinden. Somit können Sie eine Liste von erlaubten Webseiten pflegen und alle anderen blockieren. Dies wird jedoch nur in speziellen Szenarien funktionieren, da üblicherweise von den meisten Webseiten auf andere Seiten verlinkt wird, und man oftmals nicht alle Seiten überprüfen kann. Ein Nachteil ist auch die Pflege der Liste mit erlaubten Webseiten, da diese sich oft ändern kann und dementsprechend die App aktuell gehalten werden muss. Gerade wenn die anzuzeigenden Webseiten nicht im eigenen Zugriff sind, bekommt man hier schnell ein Problem.

Wenn Sie diese Probleme vermeiden wollen oder wenn die hauptsächlich anzuzeigende Webseite Ihren Speicher zu stark belastet, können Sie anstatt des `WebBrowser`-Steuerelements den `WebBrowserTask` verwenden. Nach dem Aufruf dieses Tasks wird der *Internet Explorer Mobile* in einer neuen Instanz gestartet und die eigene Anwendung im Speicher pausiert. Da der *Internet Explorer Mobile* eine eigenständige App darstellt, müssen Sie sich beim Anzeigen der Webseite keine Sorgen mehr um den dafür notwendigen Speicherverbrauch innerhalb Ihrer eigenen App machen.

Der Nachteil dieser Lösung ist allerdings, dass durch die Verwendung des `WebBrowserTask` keine Integration und Interaktion mit der eigenen Anwendung möglich ist, da der Browser in einer separaten Instanz gestartet wird.

Weitere Information zur Verwendung des `WebBrowserTask` finden Sie in Kapitel 6 unseres Buchs ab Seite 477.

Verwendung des BingMap Steuerelements

Eine Bing-Karte im Steuerelement `Map` besteht aus sehr vielen zusammengesetzten Bildfragmenten. Die Menge der Fragmente ist dabei abhängig vom gewählten Ort, Zoomfaktor und Kartentyp. Bei jeder Interaktion innerhalb des Steuerelements werden neue Bildfragmente über die Datenverbindung nachgeladen und im Applikationsspeicher des Geräts gehalten.

Wird für die eigene Anwendung keine Navigation innerhalb des Steuerelements `Map` benötigt, so kann durch Setzen der Eigenschaft `IsHitTestVisible` der Klasse `Map` auf den Wert `false` diese komplett deaktiviert werden, und man erhält so einen starren Kartenausschnitt.

Sollte zwingend eine Navigation benötigt werden, sollte diese über die Verwendung des `BingMapsTask` implementiert werden, da dieser die Kartenanwendung in einer eigenen Instanz startet und die eigene Anwendung im Hintergrund pausiert.

Nähere Information zur Implementierung des `BingMapsTask` finden Sie in Kapitel 6 unseres Buchs ab Seite 488.

Limitierung von grafischen Listboxeinträgen

Die Verwendung von komplexen Listboxeinträgen mit Bildern verbrauchen viel Applikationsspeicher. Daher sollte man für 256-MB-Geräte die Einträge in einer `Listbox` entsprechend auf eine speicherverträgliche Anzahl reduzieren, die man durch die Verwendung der *Windows Phone Performance Analysis Tools* und entsprechende Praxistests bestimmen kann. Des Weiteren sollten in der `Listbox` statt skalierter Bilder nur kleinere Vorschaubilder verwendet werden. Um weitere Einträge in der zuvor ermittelten Anzahl anzeigen zu lassen, empfiehlt es sich, eine Art Blätterfunktion, ähnlich wie beim *Paging* von Listen auf Webseiten, selbst zu implementieren.

Verzicht auf animierte Seitenübergänge

Aufgrund des Speicherverbrauchs sollte man von animierten Seitenübergängen mithilfe von `TransitionFrame` und dem dazugehörigem `TransitionService` aus dem *Silverlight Toolkit for Windows Phone* bei 256-MB-Geräten absehen, da diese sehr speicherintensiv sind. Stattdessen sollte man hier die Seiten vom Typ `PhoneApplicationFrame` verwenden bzw. eine bedingte Abfrage des Gerätetyps berücksichtigen.

Optimierung Ihrer Windows Phone Spiele

Im Vergleich zu durchschnittlichen Apps sind Spiele im Regelfall weitaus ressourcenhungriger, da dort oftmals mit relativ vielen Grafiken und Sounds gearbeitet wird. Doch gerade diese wichtigen Elemente eines Spiels kann man natürlich nicht einfach reduzieren, sondern lediglich optimieren. Nachfolgend finden Sie eine kurze Liste von Optimierungen mit denen Sie Ihr Spiel ein wenig ressourcenschonender gestalten können.

Grafikformat

Hierbei geht es nicht darum, ob eine Textur im *PNG* oder *JPEG* Format vorliegt, sondern um die Eigenschaft der Farbtiefe und des damit verbundenen Farb-Formats. Im Regelfall erstellen Sie Ihre Grafiken mit einer recht hohen Bittiefe von jeweils 8 Bit für Rot, Grün, Blau sowie den Alpha-Kanal. Dies ist auch die Voreinstellung für den *Content-Importer*. Wird diese Grafik als Textur geladen, verbraucht sie weitaus mehr Speicher als Ihre Dateigröße vermuten lässt. Gerade auf mobilen Geräten ist das Arbeiten mit diesen 32-Bit-Grafiken noch nicht allzu weit verbreitet, jedoch ist dies unter Windows Phone 7.5 möglich. Allerdings müssen Sie dazu das dementsprechende Format für Ihr Spiel einstellen. Dazu setzen Sie die Eigenschaft `PreferredBackBufferFormat` Ihres Objekts `GraphicsDeviceManager` auf den Wert `SurfaceFormat.Color`. Wenn Sie dies nicht explizit gesetzt haben, verwendet das XNA Framework hier den Wert `SurfaceFormat.Bgr565`, welcher für ein 16-Bit-Format steht, bei dem jeweils 5 Bit für Blau und Rot sowie 6 Bit für Grün verwendet werden. Falls Sie also weder den `PreferredBackBufferFormat`-Wert noch die Eigenschaften für den *Content-Importer* geändert haben, verschwenden Sie also die Hälfte des für Texturen verwendeten Speichers, ohne einen wirklichen Nutzen daraus zu ziehen.

Grafikkomprimierung

Auch hierbei geht es vorrangig um die Größe der Textur im Speicher, und nicht auf der Festplatte beziehungsweise im XAP-Archiv. Das XNA Framework bietet einen ganz einfachen Weg, um Ihre Texturen zu komprimieren. Dazu können Sie in den Eigenschaften der jeweiligen Textur einfach das *Texture Format* auf *DxtCompressed* umstellen und sparen so einiges an Speicher. Die Kompressionsrate hierbei beträgt ca. 1:8, was einen erheblichen Speicherunterschied ausmacht, sich jedoch nur minimal auf die Bildqualität auswirkt. Eine weitere Möglichkeit, die speziell auf 3D-Texturen abzielt, ist die Möglichkeit, Vertex Daten zu normalisieren, was zwar zu einer geringeren Genauigkeit führt, jedoch auch 25% weniger Speicher verbraucht.

Soundformat

Windows Phone unterstützt eine ganze Reihe an Soundformaten, genau wie auch die XNA *ContentPipeline*. Alle diese Soundformate ermöglichen das Abspielen von Hintergrundmusik und Soundeffekten, aber es gibt in Bezug auf den Speicherverbrauch einen erheblichen Unterschied bei den diversen Formaten. Da die Ansprüche an die Musik beziehungsweise Soundeffekte eines Spiels auf dem Windows Phone in der Regel nicht so hoch sind wie etwa die Ansprüche bei einem Xbox-Spiel, kann hier durchaus ein speicherfreundliches Format verwendet werden. Beim Abspielen werden Sie im Regelfall nicht einmal einen Unterschied feststellen können. Microsoft empfiehlt hier das *ADPCM*-Format für WAV-Dateien.

Verwendung von Sounds

Auch mit den Klassen `SoundEffect` und `SoundEffectInstance` kann bei optimierter Verwendung einiges an Speicher gespart werden. Sie können beispielsweise generell die Anzahl der Sounddateien im Spiel reduzieren, indem Sie für zwei ähnliche Waffen, Ereignisse oder ähnliches dieselbe Sounddatei verwenden. Tritt das gleiche Ereignis mehrfach zur selben Zeit auf, wie beispielsweise drei parallele Explosionen, reicht es gegebenenfalls aus, nur einen einzelnen Sound abzuspielen, anstatt für jedes Ereignis einen eigenen Sound. Des Weiteren sollten Sie über einen Caching-Mechanismus oder ein erweitertes Management Ihrer Sounds nachdenken. Oftmals gibt es eine Vielzahl von Soundeffekten die in einem Spiel abgespielt werden. Das so genannte *Fire & Forget* Verhalten ist zwar relativ angenehm für den Entwickler, es kann jedoch je nach Anzahl der abzuspielenden Sounds zu einem Speicherproblem kommen. Sie sollten daher einen oft verwendeten Sound nur einmal erstellen und diesen wiederverwenden. Nicht mehr benötigte Sounds sollten über die `Dispose`-Methode freigegeben werden, um Speicher zu sparen.

Weiterführende Informationen

MSDN Infoseite zur Entwicklung mit 256-MB-Geräten:

http://msdn.microsoft.com/en-us/library/hh855081%28v=vs.92%29.aspx?ocid=aff-n-we-loc--ITPRO40898&WT.mc_id=aff-n-we-loc--ITPRO40898

Informationen zur Entwicklung von Apps mit 256-MB-Geräten (Nokia Dev Wiki):

http://www.developer.nokia.com/Community/Wiki/Best_practices_for_delivering_apps_to_256_MB

Blogpost von Maarten Struys mit Tipps und Tricks zu 256-MB-Geräten:

http://mstruys.com/2012/04/19/eventiles-and-256-mb-windows-phone-devices/

Eine detailliertere Erklärung zu XNA Backbuffer Formaten und Konvertierungen:

http://konaju.com/?p=33

Eine detaillierte Erklärung des DXT Formats:

http://www.fsdeveloper.com/wiki/index.php?title=DXT_compression_explained

Vertex Normalisierung:

http://blogs.msdn.com/b/shawnhar/archive/2010/11/19/compressed-vertex-data.aspx

Improving Memory Use for XNA Games:

http://msdn.microsoft.com/en-us/library/hh855082(v=vs.92).aspx

Korrekturen

In diesem Abschnitt finden Sie Korrekturen des bereits erschienen Inhalts.

QuickInfo (ToolTip)

Die in Kapitel 4 unseres Buchs enthaltene Beschreibung zu dem `QuickInfo`- oder auch `ToolTip`-Steuerelement ist leider nicht mehr gültig. Nach Aussagen der Entwicklergruppe bei Microsoft waren sowohl die Eigenschaften in *Expression Blend*, als auch die Klasse `ToolTipService` lediglich aus Kompatibilitätsgründen vorhanden, und die visuelle Darstellung der ToolTips nie für das Windows Phone optimiert und vorgesehen. Man zeigte sich gar überrascht, dass dies unter Windows Phone 7.0 funktioniert hat. Aufgrund der ab Windows Phone 7.5 in den Steuerelementen integrierten Gestenauswertung ist dieser »Fehler« nun aber beseitigt und die ToolTips werden nicht mehr angezeigt. Zum Zeitpunkt der Erstellung dieses Textes wurde uns seitens Microsoft mitgeteilt, dass die dementsprechende Dokumentation auf MSDN schnellstmöglich aktualisiert wird, um auch dort darauf hinzuweisen, dass die XAML-Attribute sowie die Klasse `ToolTipService` lediglich aus Kompatibilitätsgründen existieren, und die ToolTips auf dem Windows Phone nicht angezeigt werden. Wir bitten diese fehlerhafte Beschreibung zu entschuldigen.