Leistungsoptimierung für die ADOBE® FLASH®-PLATTFORM

Rechtliche Hinweise

Letzte Aktualisierung 9.5.2012

Inhalt

Kapitel 1: Einführung

Adobe® AIR®- und Adobe® Flash® Player-Anwendungen können auf zahlreichen Plattformen ausgeführt werden, zum Beispiel auf Desktops, mobilen Geräten, Tablets und Fernsehgeräten. Dieses Dokument zeigt anhand von Codebeispielen und Verwendungsszenarios beste Verfahren für Entwickler, die diese Anwendungen bereitstellen. Folgende Themen werden behandelt:

- Einsparen von Arbeitsspeicher
- Minimieren der CPU-Auslastung
- Verbessern der Leistung von ActionScript 3.0
- Erhöhen der Rendergeschwindigkeit
- Optimieren der Netzwerkinteraktion
- Arbeiten mit Audio und Video
- Optimieren der SQL-Datenbankleistung
- Vergleichswerte und Bereitstellung von Anwendungen

Die meisten dieser Optimierungen gelten für Anwendungen auf allen Geräten, sowohl für die AIR-Laufzeitumgebung als auch für die Flash Player-Laufzeitumgebung. Zusätze und Ausnahmen für bestimmte Geräte werden ebenfalls aufgeführt.

Einige dieser Optimierungen beziehen sich auf Funktionen, die in Flash Player 10.1 und AIR 2.5 eingeführt wurden. Viele dieser Optimierungen gelten jedoch auch für frühere Versionen von AIR und Flash Player.

Grundlagen der Laufzeitcodeausführung

Ein wichtiger Schritt beim Erkennen, wie die Anwendungsleistung verbessert werden kann, ist das Verständnis der Codeausführung durch die Flash Plattform-Laufzeitumgebung. Die Laufzeitumgebung läuft in einer Schleife ab, in der in jedem „Bild“ bestimmte Aktionen auftreten. Mit „Bild“ ist in diesem Fall ein Zeitabschnitt gemeint, der von der Bildrate bestimmt wird, die für die Anwendung festgelegt wurde. Die Zeitdauer, die jedem Bild zugewiesen ist, entspricht direkt der Bildrate. Wenn Sie zum Beispiel eine Bildrate von 30 Bildern pro Sekunde festlegen, versucht die Laufzeit, jedem Bild ein Dreißigstel einer Sekunde zuzuweisen.

Sie legen die anfängliche Bildrate für Ihre Anwendung beim Authoring fest. Sie können die Bildrate über Einstellungen in Adobe® Flash® Builder™ oder Flash Professional festlegen. Sie können die anfängliche Bildrate auch im Code festlegen. Legen Sie die Bildrate in einer reinen ActionScript-Anwendung fest, indem Sie das `[SWF(frameRate="24")]`-Metadaten-Tag auf die Stammdokumentklasse anwenden. In MXML legen Sie das `frameRate`-Attribut im Application- oder WindowedApplication-Tag fest.

Jede Bildschleife besteht aus zwei Phasen, die in drei Teile unterteilt ist: Ereignisse, das `enterFrame`-Ereignis und die Darstellung (Rendering).

Die erste Phase enthält zwei Teile (Ereignisse und das `enterFrame`-Ereignis), die beide potenziell dazu führen, dass Ihr Code aufgerufen wird. Im ersten Teil dieser ersten Phase werden Laufzeitereignisse empfangen und abgesetzt. Diese Ereignisse können den Abschluss oder den Fortschritt von asynchronen Vorgängen darstellen, zum Beispiel als Antwort auf das Laden von Daten über ein Netzwerk. Sie beinhalten auch Ereignisse aus Benutzereingaben. Wenn Ereignisse abgesetzt werden, führt die Laufzeitumgebung Ihren Code in Listenern aus, die Sie registriert haben. Wenn kein Ereignis auftritt, wartet die Laufzeitumgebung auf den Abschluss dieser Ausführungsphase, ohne Aktionen auszuführen. Die Laufzeit setzt die Bildrate auch bei fehlender Aktivität niemals herauf. Wenn während anderer Teile des Ausführungszyklus Ereignisse auftreten, stellt die Laufzeitumgebung diese Ereignisse in eine Warteschlange und setzt sie im nächsten Bild ab.

Der zweite Teil der ersten Phase ist das `enterFrame`-Ereignis. Dieses Ereignis unterscheidet sich von den anderen, da es immer einmal pro Bild abgesetzt wird.

Nachdem alle Ereignisse abgesetzt wurden, beginnt die Renderingphase der Bildschleife. Zu diesem Zeitpunkt berechnet die Laufzeitumgebung den Status aller sichtbaren Elemente auf dem Bildschirm und zeichnet sie auf dem Bildschirm. Dann wiederholt sich der Prozess, wie ein Läufer seine Stadionrunden auf der Rennbahn zieht.

Hinweis: *Bei Ereignissen, die eine `updateAfterEvent`-Eigenschaft enthalten, kann das Rendern sofort erzwungen werden, ohne dass auf die Renderphase gewartet werden muss. Verzichten Sie jedoch auf `updateAfterEvent`, wenn es häufig zu Leistungsproblemen führt.*

Stellen Sie sich am besten vor, dass die beiden Phasen in der Bildschleife gleich lange dauern. In diesem Fall werden in einer Hälfte jeder Bildschleife Ereignisprozeduren und Anwendungscode ausgeführt, während in der anderen Hälfte das Rendering erfolgt. In der Realität sieht es jedoch oft anders aus. Manchmal benötigt Anwendungscode mehr als die Hälfte der in einem Bild verfügbaren Zeit, dehnt seine Zeitzuweisung aus und verringert somit die für das Rendering verfügbare Zeit. In anderen Fällen, besonders bei komplexen grafischen Inhalten wie Filtern und Mischmodi, benötigt das Rendering mehr als die Hälfte der Bildzeit. Da die tatsächlich von den Phasen beanspruchte Zeit flexibel ist, kann man sich die Bildschleife als „elastische Rennbahn“ vorstellen.

Wenn die gesamten Operationen der Bildschleife (Codeausführung und Rendering) zu lange dauern, kann die Laufzeitumgebung die Bildrate nicht aufrechterhalten. Das Bild wird erweitert, nimmt mehr Zeit in Anspruch als vorgesehen, deshalb gibt es eine Verzögerung, bevor das nächste Bild ausgelöst wird. Wenn eine Bildschleife beispielsweise länger als eine dreißigstel Sekunde dauert, kann die Laufzeit den Bildschirm nicht mit 30 fps aktualisieren. Wenn sich die Bildrate verringert, verschlechtert sich das Benutzererlebnis. Im besten Fall wirken Animationen abgehackt. Im schlimmsten Fall reagiert die Anwendung nicht mehr und das Fenster ist leer.

Ausführliche Informationen zum Codeausführungs- und Renderingmodell der Flash Plattform-Laufzeitumgebung finden Sie in den folgenden Ressourcen:

- Flash Player Mental Model – The Elastic Racetrack (Artikel von Ted Patrick)
- Asynchronous ActionScript Execution (Artikel von Trevor McCauley)
- Optimizing Adobe AIR for code execution, memory & rendering unter http://www.adobe.com/go/learn_fp_air_perf_tv_de (Video der MAX-Konferenzpräsentation von Sean Christmann)

Wahrgenommene Leistung und tatsächliche Leistung

Letztendlich sind es die Benutzer der Anwendung, die beurteilen, ob Ihre Anwendung gut läuft. Entwickler können die Anwendungsleistung messen, indem sie feststellen, wie lange bestimmte Vorgänge dauern oder wie viele Instanzen von Objekten erstellt werden. Diese Messungen interessieren Endbenutzer jedoch nicht. Häufig bewerten Benutzer die Anwendungsleistung nach unterschiedlichen Kriterien. Funktioniert die Anwendung rasch und reibungslos und reagiert sie schnell auf Benutzereingaben? Wirkt sie sich negativ auf die Systemleistung aus? Mit den folgenden Fragen können Sie die wahrgenommene Leistung Ihrer Anwendung testen:

- Sind Animationen glatt oder abgehackt?
- Sehen Videoinhalte reibungslos oder ruckelig aus?
- Werden Audioclips kontinuierlich abgespielt oder gibt es Pausen?
- Flackert das Fenster oder wird dunkel, wenn längere Vorgänge ausgeführt werden?
- Werden Zeichen bei der Texteingabe sofort angezeigt oder tritt eine Verzögerung auf?
- Passiert beim Klicken sofort etwas oder erst nach einer Verzögerung?
- Wird der CPU-Lüfter lauter, wenn die Anwendung läuft?
- Bei einem Laptopcomputer oder mobilen Gerät: Wird der Akku beim Ausführen der Anwendung schnell entladen?
- Reagieren andere Programme langsamer, wenn die Anwendung ausgeführt wird?

Die Unterscheidung zwischen wahrgenommener und tatsächlicher Leistung ist wichtig. Der Weg zum Erreichen der besten wahrgenommenen Leistung entspricht nicht immer dem Weg zum Erzielen der absolut schnellsten Leistung. Achten Sie darauf, dass Ihre Anwendung niemals so viel Code ausführt, dass die Laufzeit den Bildschirm nicht oft aktualisieren und Benutzereingaben erfassen kann. Manchmal ist dieses Gleichgewicht nur zu erreichen, indem eine Programmaufgabe unterteilt wird, sodass die Laufzeitumgebung den Bildschirm zwischen den einzelnen Teilen aktualisieren kann. (Spezielle Informationen hierzu finden Sie unter „Renderleistung“ auf Seite 49.)

Die hier beschriebenen Tipps und Techniken zielen sowohl auf eine verbesserte tatsächliche Leistung bei der Codeausführung als auch auf eine verbesserte vom Benutzer wahrgenommene Leistung ab.

Optimierungsziele

Einige Leistungsverbesserungen machen sich bei Benutzern nicht unmittelbar bemerkbar. Legen Sie bei der Leistungsoptimierung den Schwerpunkt auf Bereiche, die für die jeweilige Anwendung problematisch sein können. Bei einigen Leistungsoptimierungen handelt es sich um bewährte Verfahren, die immer befolgt werden können. Bei anderen Optimierungen ist es von den Anforderungen Ihrer Anwendung und von der erwarteten Zielgruppe abhängig, ob sie sinnvoll sind oder nicht. Beispielsweise fällt die Anwendungsleistung immer besser aus, wenn Sie auf Animationen, Video, Grafikfilter oder Effekte verzichten. Allerdings benutzen Sie die Flash-Plattform ja zum Erstellen von Anwendungen, eben weil Sie damit so gute Medien- und Grafikfähigkeiten einsetzen können, die ausdrucksstarke Anwendungen ermöglichen. Überlegen Sie, ob die gewünschte Funktionalitätsvielfalt der Anwendung in einem angemessenen Verhältnis zu den Leistungsmerkmalen der Geräte steht, auf denen die Anwendung ausgeführt wird.

Ein allgemeiner Ratschlag ist, „nicht zu früh zu optimieren“. Für einige Leistungsoptimierungen muss Code auf eine Weise geschrieben werden, die schwieriger zu lesen oder weniger flexibel ist. Derartiger Code lässt sich nach der Optimierung schwieriger verwalten. Für diese Optimierungen ist es oft besser, etwas zu warten und zu sehen, ob ein bestimmter Codeabschnitt schlecht ausgeführt wird, bevor Sie den Code optimieren.

Zum Verbessern der Leistung müssen manchmal Kompromisse eingegangen werden. Im Idealfall führt eine Verringerung des Arbeitsspeicherbedarfs einer Anwendung gleichzeitig zu einer höheren Ausführungsgeschwindigkeit der Anwendung. Die ideale Verbesserung ist jedoch nicht immer möglich. Wenn eine Anwendung zum Beispiel während einer Operation einfriert, besteht die Lösung häufig darin, Aufgaben so zu teilen, dass sie in mehreren Bildern ausgeführt werden. Da die Arbeitsschritte aufgeteilt werden, nimmt der ganze Vorgang wahrscheinlich mehr Zeit in Anspruch. Möglicherweise nimmt der Benutzer diese zusätzliche Zeit jedoch nicht wahr, wenn die Anwendung auch weiterhin auf Eingaben reagiert und nicht hängenbleibt.

Eine Möglichkeit, um festzustellen, was optimiert werden kann und ob die Optimierung sinnvoll ist, ist das Durchführen von Leistungstests. Verschiedene Techniken und Tipps zum Testen der Leistung werden unter „Vergleichswerte und Bereitstellung“ auf Seite 97 beschrieben.

Weitere Informationen zum Bestimmen von Anwendungsteilen, die für die Optimierung in Frage kommen, finden Sie in den folgenden Ressourcen:

- Performance-tuning apps for AIR unter http://www.adobe.com/go/learn_fp_goldman_tv_de (Video der MAX-Konferenzpräsentation von Oliver Goldman)
- Performance-tuning Adobe AIR applications unter http://www.adobe.com/go/learn_fp_air_perf_devnet_de (Artikel von Oliver Goldman in der Adobe Developer Connection auf Grundlage der Präsentation)

Kapitel 2: Einsparen von Arbeitsspeicher

Das Einsparen von Arbeitsspeicher ist stets ein wichtiger Aspekt bei der Anwendungsentwicklung, selbst für Desktopanwendungen. Bei Mobilgeräten kommt der Arbeitsspeicherbeanspruchung jedoch eine wesentlich größere Bedeutung zu; deshalb sollten Sie darauf achten, dass Ihre Anwendung möglichst wenig Arbeitsspeicher belegt.

Anzeigeobjekte

Wählen Sie ein geeignetes Anzeigeobjekt.

ActionScript 3.0 umfasst zahlreiche Anzeigeobjekte. Eine der einfachsten Techniken zur Optimierung der Arbeitsspeichernutzung ist die Wahl des geeigneten Anzeigeobjekts. Für einfache, nicht interaktive Formen empfiehlt sich die Verwendung von Shape-Objekten. Für interaktive Objekte, die keine Zeitleiste benötigen, eignen sich Sprite-Objekte. Für Animationen mit einer Zeitleiste sollten MovieClip-Objekte verwendet werden. Wählen Sie stets den Objekttyp, der für Ihre Anwendung ein Höchstmaß ein Effizienz bietet.

Der folgende Code veranschaulicht die Arbeitsspeichernutzung für verschiedene Anzeigeobjekte:

```
trace(getSize(new Shape()));
// output: 236

trace(getSize(new Sprite()));
// output: 412

trace(getSize(new MovieClip()));
// output: 440
```

Die `getSize()`-Methode zeigt in Byte, wie viel Arbeitsspeicher ein Objekt belegt. Sie sehen, dass die Verwendung mehrerer MovieClip-Objekte anstelle von einfachen Shape-Objekten unnötig viel Arbeitsspeicher beansprucht, wenn die Funktionsmerkmale eines MovieClips-Objekts nicht benötigt werden.

Grundtypen

Verwenden Sie die `getSize()`-Methode, um Vergleichswerte für Code und das effizienteste Objekt für die jeweilige Aufgabe zu bestimmen.

Alle Grundtypen mit Ausnahme von „String“ belegen 4 - 8 Byte Arbeitsspeicher. Die Arbeitsspeichernutzung kann nicht durch Angabe eines bestimmten Grundtyps optimiert werden:

```
// Primitive types
var a:Number;
trace(getSize(a));
// output: 8

var b:int;
trace(getSize(b));
// output: 4

var c:uint;
trace(getSize(c));
// output: 4

var d:Boolean;
trace(getSize(d));
// output: 4

var e:String;
trace(getSize(e));
// output: 4
```

Die AVM (ActionScript Virtual Machine) weist einem Number-Datentyp, der einen 64-Bit-Wert darstellt, 8 Byte zu, wenn kein Wert zugeordnet wurde. Alle anderen Grundtypen werden mit 4 Byte gespeichert.

```
// Primitive types
var a:Number = 8;
trace(getSize(a));
// output: 4

a = Number.MAX_VALUE;
trace(getSize(a));
// output: 8
```

Das Verhalten für den String-Typ ist anders. Die zugewiesene Speichermenge richtet sich nach der String-Länge:

```
var name:String;
trace(getSize(name));
// output: 4

name = "";
trace(getSize(name));
// output: 24

name = "Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum
has been the industry's standard dummy text ever since the 1500s, when an unknown printer took
a galley of type and scrambled it to make a type specimen book. It has survived not only five
centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It
was popularized in the 1960s with the release of Letraset sheets containing Lorem Ipsum
passages, and more recently with desktop publishing software like Aldus PageMaker including
versions of Lorem Ipsum.";
trace(getSize(name));
// output: 1172
```

Verwenden Sie die `getSize()`-Methode, um Vergleichswerte für Code und das effizienteste Objekt für die jeweilige Aufgabe zu bestimmen.

Wiederverwenden von Objekten

Sie sollten Objekte nach Möglichkeit immer wiederverwenden, anstatt sie neu zu erstellen.

Eine weitere einfache Möglichkeit zur Optimierung der Arbeitsspeichernutzung besteht darin, Objekte nach Möglichkeit wiederzuverwenden, anstatt sie neu zu erstellen. Verwenden Sie in einer Schleife beispielsweise nicht den folgenden Code:

```
const MAX_NUM:int = 18;
const COLOR:uint = 0xCCCCCC;

var area:Rectangle;

for (var:int = 0; i < MAX_NUM; i++)
{
    // Do not use the following code
    area = new Rectangle(i,0,1,10);
    myBitmapData.fillRect(area,COLOR);
}
```

Die erneute Erstellung des Rectangle-Objekts in jeder Schleifeniteration belegt mehr Arbeitsspeicher und verlangsamt die Ausführung, da in jeder Iteration ein neues Objekt erstellt wird. Verwenden Sie das folgende Verfahren:

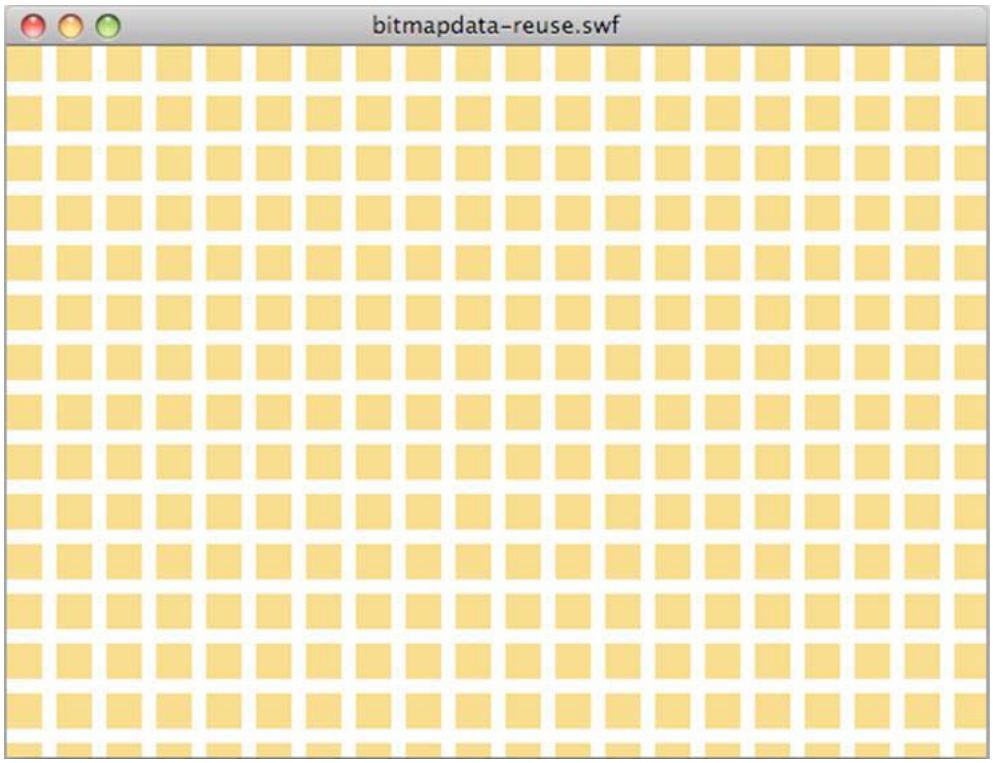
```
const MAX_NUM:int = 18;
const COLOR:uint = 0xCCCCCC;

// Create the rectangle outside the loop
var area:Rectangle = new Rectangle(0,0,1,10);

for (var:int = 0; i < MAX_NUM; i++)
{
    area.x = i;
    myBitmapData.fillRect(area,COLOR);
}
```

Das Objekt im vorherigen Beispiel belegt nur relativ wenig Arbeitsspeicher. Das nächste Beispiel zeigt, wie durch die Wiederverwendung eines BitmapData-Objekts noch mehr Arbeitsspeicher eingespart werden kann. Beim folgenden Code zur Erstellung eines Kacheleffekts wird unnötig viel Arbeitsspeicher belegt:

```
var myImage:BitmapData;
var myContainer:Bitmap;
const MAX_NUM:int = 300;

for (var i:int = 0; i< MAX_NUM; i++)
{
    // Create a 20 x 20 pixel bitmap, non-transparent
    myImage = new BitmapData(20,20,false,0xF0D062);

    // Create a container for each BitmapData instance
    myContainer = new Bitmap(myImage);

    // Add it to the display list
    addChild(myContainer);

    // Place each container
    myContainer.x = (myContainer.width + 8) * Math.round(i % 20);
    myContainer.y = (myContainer.height + 8) * int(i / 20);
}
```

Hinweis: *Bei positiven Werten ist eine Umwandlung des gerundeten Wertes in den int-Datentyp wesentlich schneller als die Verwendung der* `Math.floor()`*-Methode.*

Die folgende Abbildung zeigt das Ergebnis der Bitmapkacheln:



Ergebnis der Bitmapkacheln

Eine optimierte Version erstellt eine einzelne BitmapData-Instanz, auf die mehrere Bitmap-Instanzen verweisen. Dabei wird dasselbe Ergebnis erzielt:

```
// Create a single 20 x 20 pixel bitmap, non-transparent
var myImage:BitmapData = new BitmapData(20,20,false,0xF0D062);
var myContainer:Bitmap;
const MAX_NUM:int = 300;

for (var i:int = 0; i< MAX_NUM; i++)
{
    // Create a container referencing the BitmapData instance
    myContainer = new Bitmap(myImage);

    // Add it to the display list
    addChild(myContainer);

    // Place each container
    myContainer.x = (myContainer.width + 8) * Math.round(i % 20);
    myContainer.y = (myContainer.height + 8) * int(i / 20);
}
```

Bei diesem Verfahren können ca. 700 KB Arbeitsspeicher eingespart werden, was bei herkömmlichen Mobilgeräten eine bedeutende Einsparung darstellt. Jeder Bitmapcontainer kann über die Bitmap-Eigenschaften bearbeitet werden, ohne dass die ursprüngliche BitmapData-Instanz geändert werden muss:

```
// Create a single 20 x 20 pixel bitmap, non-transparent
var myImage:BitmapData = new BitmapData(20,20,false,0xF0D062);
var myContainer:Bitmap;
const MAX_NUM:int = 300;

for (var i:int = 0; i< MAX_NUM; i++)
{
    // Create a container referencing the BitmapData instance
    myContainer = new Bitmap(myImage);

    // Add it to the DisplayList
    addChild(myContainer);

    // Place each container
    myContainer.x = (myContainer.width + 8) * Math.round(i % 20);
    myContainer.y = (myContainer.height + 8) * int(i / 20);

    // Set a specific rotation, alpha, and depth
    myContainer.rotation = Math.random()*360;
    myContainer.alpha = Math.random();
    myContainer.scaleX = myContainer.scaleY = Math.random();
}
```

Die folgende Abbildung zeigt das Resultat der Bitmaptransformationen:

Resultat der Bitmaptransformationen

Verwandte Themen

„Bitmap-Zwischenspeicherung“ auf Seite 55

Objektpools

Verwenden Sie nach Möglichkeit Objektpools.

Eine weitere wichtige Optimierungsmethode ist die Verwendung von Objektpools, die die Wiederverwendung von Objekten ermöglichen. Während der Initialisierung der Anwendung erstellen Sie eine definierte Anzahl an Objekten, die in einem Pool gespeichert werden, wie beispielsweise ein Array- oder ein Vector-Objekt. Wenn ein Objekt nicht mehr benötigt wird, deaktivieren Sie es, damit es keine CPU-Ressourcen belegt. Außerdem entfernen Sie alle gegenseitigen Verweise. Sie stellen die Verweise jedoch nicht auf `null` ein, wodurch sie bei der Speicherbereinigung berücksichtigt würden. Sie verschieben das Objekt lediglich zurück in den Pool und rufen es wieder ab, wenn Sie ein neues Objekt benötigen.

Durch die Wiederverwendung von Objekten müssen Objekte weniger häufig instanziiert werden, was zahlreiche Ressourcen beanspruchen kann. Außerdem verringert sich die Wahrscheinlichkeit, dass die Speicherbereinigung ausgeführt wird, die sich ebenfalls negativ auf die Anwendungsgeschwindigkeit auswirken würde. Der folgende Code veranschaulicht die Verwendung von Objektpools:

```
package
{
    import flash.display.Sprite;

    public final class SpritePool
    {
        private static var MAX_VALUE:uint;
        private static var GROWTH_VALUE:uint;
        private static var counter:uint;
        private static var pool:Vector.<Sprite>;
        private static var currentSprite:Sprite;

        public static function initialize( maxPoolSize:uint, growthValue:uint ):void
        {
            MAX_VALUE = maxPoolSize;
            GROWTH_VALUE = growthValue;
            counter = maxPoolSize;

            var i:uint = maxPoolSize;

            pool = new Vector.<Sprite>(MAX_VALUE);
            while( --i > -1 )
                pool[i] = new Sprite();
        }

        public static function getSprite():Sprite
        {
            if ( counter > 0 )
                return currentSprite = pool[--counter];

            var i:uint = GROWTH_VALUE;
            while( --i > -1 )
                    pool.unshift ( new Sprite() );
            counter = GROWTH_VALUE;
            return getSprite();

        }

        public static function disposeSprite(disposedSprite:Sprite):void
        {
            pool[counter++] = disposedSprite;
        }
    }
}
```

Die SpritePool-Klasse erstellt bei der Initialisierung der Anwendung einen Pool mit neuen Objekten. Die `getSprite()`-Methode gibt Instanzen dieser Objekte zurück und die `disposeSprite()`-Methode gibt sie frei. Der Code bietet dem Pool die Möglichkeit, bei Bedarf zu wachsen. Es ist auch möglich, einen Pool mit einer festen Größe zu erstellen; in diesem Fall werden keine neuen Objekte zugewiesen, wenn der Pool erschöpft ist. Vermeiden Sie nach Möglichkeit, neue Objekte in Schleifen zu erstellen. Weitere Informationen finden Sie unter „Freigeben von Arbeitsspeicher“ auf Seite 12. Im folgenden Code wird die SpritePool-Klasse zum Abrufen neuer Instanzen verwendet:

```
const MAX_SPRITES:uint = 100;
const GROWTH_VALUE:uint = MAX_SPRITES >> 1;
const MAX_NUM:uint = 10;

SpritePool.initialize ( MAX_SPRITES,  GROWTH_VALUE );

var currentSprite:Sprite;
var container:Sprite = SpritePool.getSprite();

addChild ( container );

for ( var i:int = 0; i< MAX_NUM; i++ )
{
    for ( var j:int = 0; j< MAX_NUM; j++ )
    {
        currentSprite = SpritePool.getSprite();
        currentSprite.graphics.beginFill ( 0x990000 );
        currentSprite.graphics.drawCircle ( 10, 10, 10 );
        currentSprite.x = j * (currentSprite.width + 5);
        currentSprite.y = i * (currentSprite.width + 5);
        container.addChild ( currentSprite );
    }
}
```

Der folgende Code entfernt beim Klicken mit der Maus alle Anzeigeobjekte aus der Anzeigeliste und verwendet sie später für eine andere Aufgabe wieder:

```
stage.addEventListener ( MouseEvent.CLICK, removeDots );

function removeDots ( e:MouseEvent ):void
{
    while (container.numChildren > 0 )
        SpritePool.disposeSprite (container.removeChildAt(0) as Sprite );
}
```

Hinweis: *Der Pool-Vektor verweist immer auf die Sprite-Objekte. Wenn Sie das Objekt vollständig aus dem Arbeitsspeicher entfernen möchten, müssen Sie eine* `dispose()`*-Methode für die SpritePool-Klasse angeben, wodurch alle verbleibenden Verweise entfernt werden.*

Freigeben von Arbeitsspeicher


Löschen Sie alle Verweise auf Objekte, um sicherzustellen, dass die Speicherbereinigung ausgelöst wird.

Die Speicherbereinigung kann nicht direkt in der Release-Version von Flash Player gestartet werden. Um sicherzustellen, dass ein Objekt bei der Speicherbereinigung berücksichtigt wird, löschen Sie alle Verweise auf das Objekt. Beachten Sie, dass der alte `delete`-Operator aus ActionScript 1.0 und 2.0 sich in ActionScript 3.0 anders verhält. Er kann nur verwendet werden, um die dynamischen Eigenschaften eines dynamischen Objekts zu löschen.

Hinweis: *Sie können die Speicherbereinigung direkt in Adobe® AIR® und in der Debugger-Version von Flash Player aufrufen.*

Mit dem folgenden Code wird beispielsweise ein Sprite-Verweis auf `null` eingestellt:

```
var mySprite:Sprite = new Sprite();

// Set the reference to null, so that the garbage collector removes
// it from memory
mySprite = null;
```

Berücksichtigen Sie, dass ein Objekt nicht unbedingt aus dem Arbeitsspeicher entfernt wird, wenn es auf `null` eingestellt ist. In manchen Fällen wird die Speicherbereinigung nicht ausgeführt, wenn der verfügbare Arbeitsspeicher nicht als niedrig genug eingestuft wird. Das Verhalten der Speicherbereinigung lässt sich nicht mit Sicherheit voraussagen. Die Speicherbereinigung wird nicht durch das Löschen von Objekten, sondern durch die Zuweisung von Arbeitsspeicher ausgelöst. Wenn die Speicherbereinigung ausgeführt wird, werden Diagramme der Objekte gefunden, die noch nicht bereinigt wurden. Inaktive Objekte in den Diagrammen werden erkannt, indem Objekte ermittelt werden, die aufeinander verweisen, aber nicht mehr von der Anwendung verwendet werden. Inaktive Objekte, die auf diese Weise erkannt wurden, werden gelöscht.

Bei großen Anwendungen kann dieser Vorgang die CPU stark beanspruchen und die Leistung sowie die Anwendungsgeschwindigkeit deutlich beeinträchtigen. Reduzieren Sie die Ausführungshäufigkeit der Speicherbereinigung, indem Sie Objekte so oft wie möglich wiederverwenden. Außerdem sollten Sie die Verweise nach Möglichkeit auf Null setzen, damit bei der Speicherbereinigung weniger Zeit zum Suchen der Objekte anfällt. Stellen Sie sich die Speicherbereinigung als eine Art Versicherung vor, und versuchen Sie stets, den Lebenszyklus der Objekte explizit zu verwalten, sofern möglich.

Hinweis: *Wenn der Verweis auf ein Anzeigeobjekt auf Null gesetzt wird, ist nicht gewährleistet, dass das Objekt fixiert wird. Das Objekt belegt auch weiterhin CPU-Ressourcen, bis es der Speicherbereinigung zugeführt wird. Achten Sie darauf, dass das Objekt ordnungsgemäß deaktiviert wurde, bevor Sie seinen Verweis auf* `null` *setzen.*

Die Speicherbereinigung kann mithilfe der `System.gc()`-Methode gestartet werden, die in Adobe AIR sowie in der Debugger-Version von Flash Player zur Verfügung steht. Über den Profiler in Adobe® Flash® Builder™ können Sie die Speicherbereinigung manuell starten. Über die Ausführung der Speicherbereinigung können Sie die Reaktionsweise der Anwendung beobachten und feststellen, ob Objekte korrekt aus dem Arbeitsspeicher gelöscht werden.

Hinweis: *Wenn ein Objekt als Ereignis-Listener verwendet wurde, kann ein anderes Objekt darauf verweisen. In diesem Fall sollten Sie die Ereignis-Listener mithilfe der* `removeEventListener()`*-Methode entfernen, bevor Sie die Verweise auf* `null` *setzen.*

Die von Bitmaps belegte Arbeitsspeichermenge kann unmittelbar verringert werden. Beispielsweise enthält die BitmapData-Klasse die `dispose()`-Methode. Im nächsten Beispiel wird eine BitmapData-Instanz mit einer Größe von 1,8 MB erstellt. Die aktuell belegte Arbeitsspeichermenge erhöht sich auf 1,8 MB und die `System.totalMemory`-Eigenschaft gibt einen kleineren Wert zurück:

```
trace(System.totalMemory / 1024);
// output: 43100

// Create a BitmapData instance
var image:BitmapData = new BitmapData(800, 600);

trace(System.totalMemory / 1024);
// output: 44964
```

Nun wird die BitmapData-Instanz manuell aus dem Arbeitsspeicher entfernt und die Arbeitsspeicherbelegung wird erneut geprüft:

```
trace(System.totalMemory / 1024);
// output: 43100

// Create a BitmapData instance
var image:BitmapData = new BitmapData(800, 600);

trace(System.totalMemory / 1024);
// output: 44964

image.dispose();
image = null;

trace(System.totalMemory / 1024);
// output: 43084
```

Obwohl die `dispose()`-Methode die Pixel aus dem Arbeitsspeicher entfernt, muss der Verweis auf `null` eingestellt werden, um eine vollständige Freigabe zu erzielen. Wenn ein BitmapData-Objekt nicht mehr benötigt wird, sollten Sie immer die `dispose()`-Methode aufrufen und den Verweis auf `null` einstellen, um den Arbeitsspeicher unmittelbar freizugeben.

Hinweis: *Ab Flash Player 10.1 und AIR 1.5.2 steht eine neue Methode namens `disposeXML()` für die System-Klasse zur Verfügung. Mit dieser Methode können Sie ein XML-Objekt sofort für die Speicherbereinigung zur Verfügung stellen, indem Sie die XML-Struktur als Parameter übergeben.*

Verwandte Themen

„Fixieren und Freigeben von Objekten“ auf Seite 29

Verwenden von Bitmaps

Durch Verwendung von Vektoren anstelle von Bitmaps lässt sich Arbeitsspeicher einsparen. Dabei muss jedoch berücksichtigt werden, dass Vektoren die CPU und GPU stark beanspruchen, besonders wenn sie in großer Zahl verwendet werden. Durch die Verwendung von Bitmaps kann das Rendern optimiert werden, da die Laufzeitumgebung zum Zeichnen von Pixeln auf dem Bildschirm weniger Verarbeitungsleistung erfordert als zum Rendern von Vektorinhalt.

Verwandte Themen

„Manuelle Bitmap-Zwischenspeicherung“ auf Seite 63

Bitmap-Downsampling

Zur Optimierung der Arbeitsspeichernutzung werden undurchsichtige 32-Bit-Bilder auf 16 Bit reduziert, wenn Flash Player einen 16-Bit-Bildschirm erkennt. Dieses Downsampling belegt nur die Hälfte der Arbeitsspeicherressourcen; außerdem können Bilder schneller gerendert werden. Dieses Funktionsmerkmal ist nur in Flash Player 10.1 für Windows Mobile verfügbar.

Hinweis: *Vor Flash Player 10.1 wurden alle im Arbeitsspeicher erstellten Pixel mit 32 Bit (4 Byte) gespeichert. Ein einfaches Logo mit einer Größe von 300 x 300 Pixeln belegte 350 KB Arbeitsspeicher (300 x 300 x 4 : 1024). Bei Verwendung des neuen Verhaltens belegt das gleiche undurchsichtige Logo nur 175 KB. Wenn das Logo transparent ist, wird kein Downsampling auf 16 Bit vorgenommen und das Bild belegt dieselbe Arbeitsspeichermenge. Dieses Funktionsmerkmal gilt nur für eingebettete Bitmaps oder für Bilder, die zur Laufzeit geladen werden (PNG, GIF, JPG).*

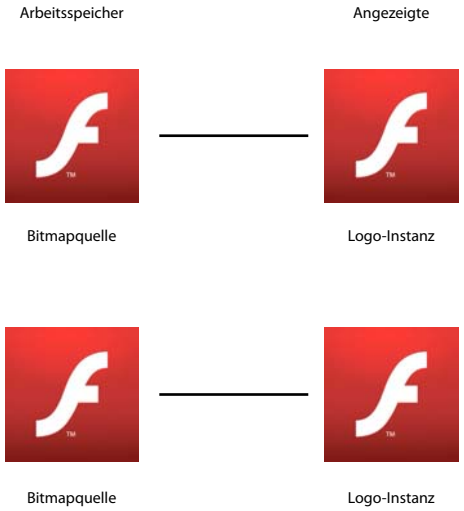
Auf Mobilgeräten ist der Unterschied zwischen einem mit 16 Bit gerenderten Bild und demselben Bild mit 32 Bit oft nur schwer zu erkennen. Bei einfachen Bildern, die nur wenige Farben aufweisen, besteht kein erkennbarer Unterschied. Selbst bei einem komplexeren Bild ist der Unterschied nicht sofort augenfällig. Beim Vergrößern des Bildes werden die Farben jedoch möglicherweise schlechter dargestellt und ein 16-Bit-Verlauf ist eventuell weniger gleichmäßig als die 32-Bit-Version.

BitmapData-Einzelverweis

Es ist wichtig, die Verwendung der BitmapData-Klasse zu optimieren, indem Instanzen so oft wie möglich wieder verwendet werden. In Flash Player 10.1 und AIR 2.5 wurde eine neue Funktion für alle Funktionen eingeführt, die einzelner BitmapData-Verweis heißt. Bei der Erstellung von BitmapData-Instanzen von einem eingebetteten Bild wird eine einzelne Version der Bitmap für alle BitmapData-Instanzen verwendet. Wenn eine Bitmap später geändert wird, erhält sie eine eigene Bitmap im Arbeitsspeicher. Das eingebettete Bild kann aus der Bibliothek oder aus einem [Embed]-Tag stammen.

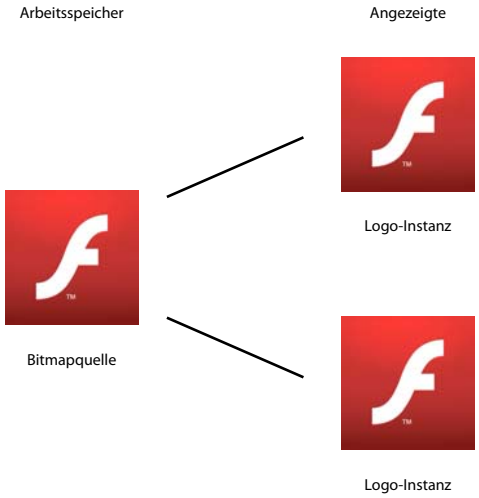
Hinweis: *Vorhandene Inhalte profitieren ebenfalls von dieser neuen Funktion, da Flash Player 10.1 und AIR 2.5 Bitmaps automatisch wiederverwenden.*

Beim Instanziieren eines eingebetteten Bildes wird im Arbeitsspeicher eine zugehörige Bitmap erstellt. Vor Flash Player 10.1 und AIR 2.5 wurde jeder Instanz eine separate Bitmap im Arbeitsspeicher zugeteilt, wie in der folgenden Darstellung:



Bitmaps im Arbeitsspeicher vor Flash Player 10.1 und AIR 2.5

In Flash Player 10.1 und AIR 2.5 wird eine einzelne Version der Bitmap für alle BitmapData-Instanzen verwendet, wenn mehrere Instanzen desselben Bildes erstellt werden. Dieses Konzept wird in der folgenden Abbildung veranschaulicht:

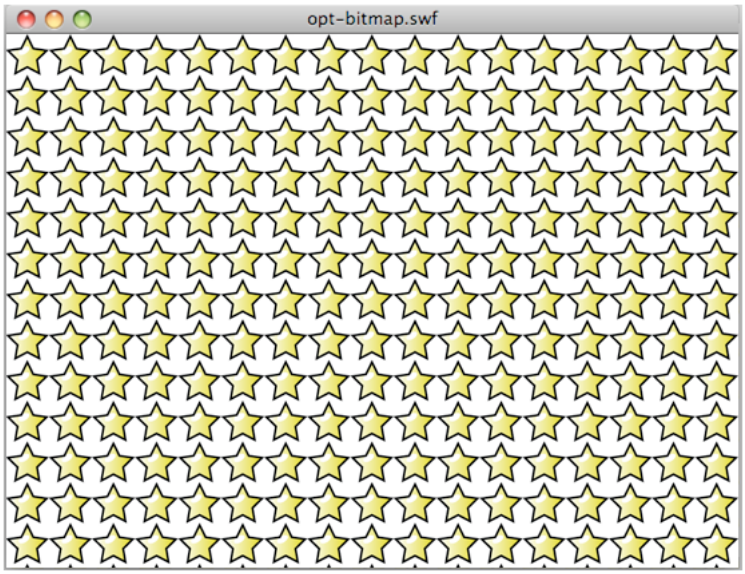


Bitmaps im Arbeitsspeicher in Flash Player 10.1 und AIR 2.5

Bei diesem Verfahren belegen Anwendungen, die zahlreiche Bitmaps enthalten, wesentlich weniger Arbeitsspeicher. Mit dem folgenden Code werden mehrere Instanzen eines `Star`-Symbols erstellt:

```
const MAX_NUM:int = 18;

var star:BitmapData;
var bitmap:Bitmap;

for (var i:int = 0; i<MAX_NUM; i++)
{
    for (var j:int = 0; j<MAX_NUM; j++)
    {
        star = new Star(0,0);
        bitmap = new Bitmap(star);
        bitmap.x = j * star.width;
        bitmap.y = i * star.height;
        addChild(bitmap)
    }
}
```

Die folgende Abbildung zeigt das Ergebnis des Codes:



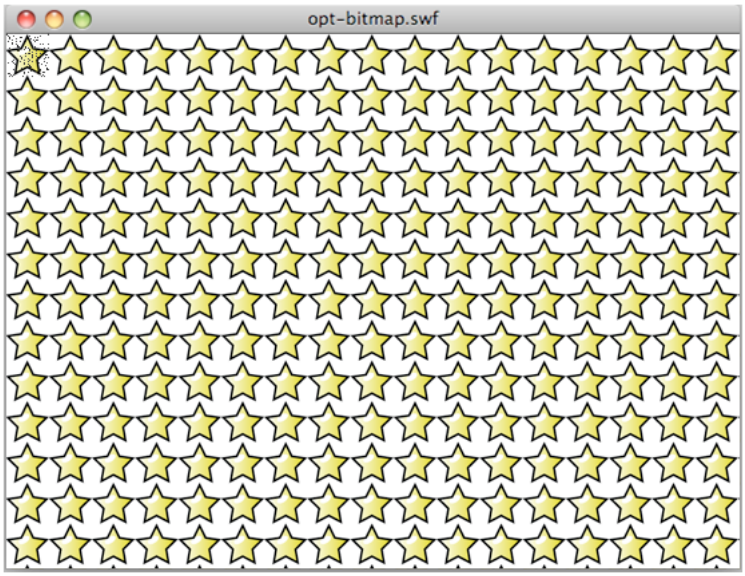
Ergebnis des Codes zur Erstellung mehrerer Instanzen eines Symbols

Mit Flash Player 10 benötigt die oben stehende Animation zum Beispiel ca. 1008 KB Arbeitsspeicher. Mit Flash Player 10.1 benötigt die Animation sowohl auf dem Desktop als auch auf einem mobilen Gerät nur 4 KB.

Mit dem folgenden Code wird eine BitmapData-Instanz geändert:

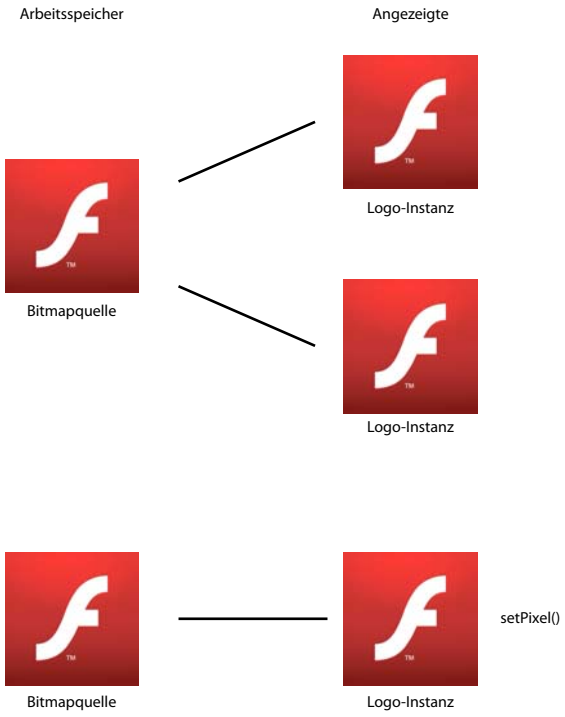
```
const MAX_NUM:int = 18;

var star:BitmapData;
var bitmap:Bitmap;

for (var i:int = 0; i<MAX_NUM; i++)
{
    for (var j:int = 0; j<MAX_NUM; j++)
    {
        star = new Star(0,0);
        bitmap = new Bitmap(star);
        bitmap.x = j * star.width;
        bitmap.y = i * star.height;
        addChild(bitmap)
    }
}

var ref:Bitmap = getChildAt(0) as Bitmap;

ref.bitmapData.pixelDissolve(ref.bitmapData, ref.bitmapData.rect, new
Point(0,0),Math.random()*200,Math.random()*200, 0x990000);
```

Die folgende Abbildung zeigt das Ergebnis, wenn eine `Star`-Instanz geändert wird:



Ergebnis nach Ändern einer Instanz

Intern erstellt die Laufzeitumgebung automatisch eine Bitmap im Arbeitsspeicher und weist diese zu, um die Pixeländerungen zu verarbeiten. Wenn eine Methode der BitmapData-Klasse aufgerufen wird, die zu Pixeländerungen führt, wird eine neue Instanz im Arbeitsspeicher erstellt; es werden keine anderen Instanzen aktualisiert. Dieses Konzept wird in der folgenden Abbildung veranschaulicht:



Ergebnis im Arbeitsspeicher nach Ändern einer Bitmap

Wenn ein Stern geändert wird, wird im Arbeitsspeicher eine neue Kopie erstellt. Die resultierende Animation benötigt in Flash Player 10.1 und AIR 2.5 ungefähr 8 KB Arbeitsspeicher.

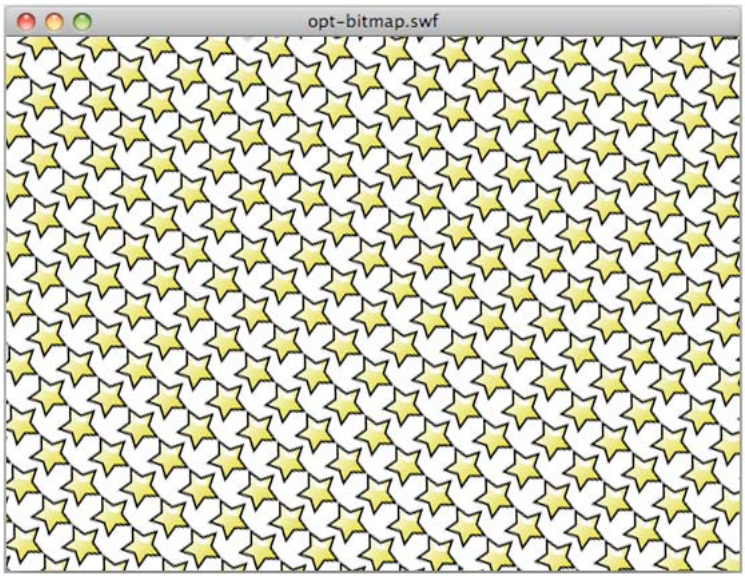
Im vorherigen Beispiel steht jede Bitmap einzeln zur Transformation zur Verfügung. Wenn nur der Kacheleffekt erstellt werden soll, ist die `beginBitmapFill()`-Methode am besten geeignet:

```
var container:Sprite = new Sprite();

var source:BitmapData = new Star(0,0);

// Fill the surface with the source BitmapData
container.graphics.beginBitmapFill(source);
container.graphics.drawRect(0,0,stage.stageWidth,stage.stageHeight);

addChild(container);
```

Mit diesem Verfahren wird dasselbe Ergebnis erzielt, es wird jedoch nur eine BitmapData-Instanz erstellt. Um die Sterne kontinuierlich zu drehen, verwenden Sie ein Matrix-Objekt, das in jedem Bild gedreht wird, anstatt auf jede Stern-Instanz einzeln zuzugreifen. Übergeben Sie dieses Matrix-Objekt an die `beginBitmapFill()`-Methode:

```
var container:Sprite = new Sprite();

container.addEventListener(Event.ENTER_FRAME, rotate);

var source:BitmapData = new Star(0,0);
var matrix:Matrix = new Matrix();

addChild(container);

var angle:Number = .01;

function rotate(e:Event):void
{
    // Rotate the stars
    matrix.rotate(angle);

    // Clear the content
    container.graphics.clear();

    // Fill the surface with the source BitmapData
    container.graphics.beginBitmapFill(source,matrix,true,true);
    container.graphics.drawRect(0,0,stage.stageWidth,stage.stageHeight);
}
```

Diese Technik erfordert keine ActionScript-Schleife, um den Effekt zu erzielen. Die Laufzeitumgebung verarbeitet alles intern. Die folgende Abbildung zeigt das Ergebnis der Transformation der Sterne:



Ergebnis nach dem Drehen der Sterne

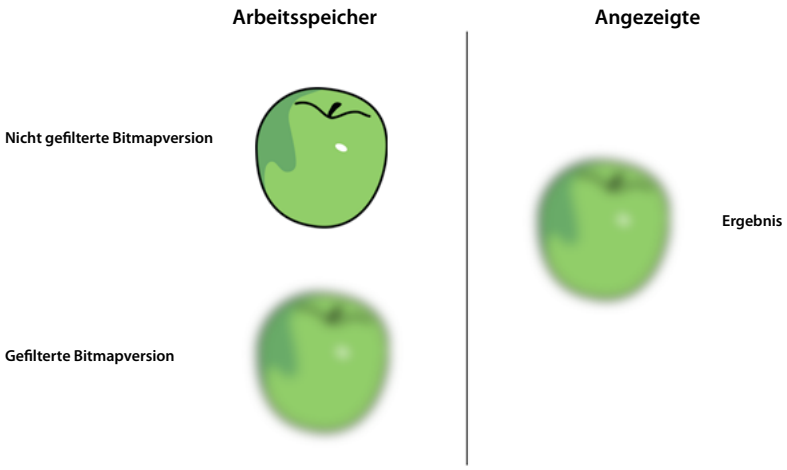
Bei diesem leistungsstarken Verfahren bewirkt eine Aktualisierung des ursprünglichen BitmapData-Quellobjekts automatisch eine Aktualisierung der weiteren Instanzen auf der Bühne. Mit diesem Verfahren ist es jedoch nicht möglich, jeden Stern einzeln zu skalieren, wie im vorherigen Beispiel.

Hinweis: *Bei Verwendung von mehreren Instanzen desselben Bildes richtet sich das Zeichnen danach, ob der ursprünglichen Bitmap im Arbeitsspeicher eine Klasse zugeordnet ist. Ist der Bitmap keine Klasse zugeordnet, werden Bilder als Shape-Objekte mit Bitmapfüllungen gezeichnet.*

Filter und dynamisches Entladen von Bitmaps

Vermeiden Sie Filter, auch solche, die über Pixel Bender verarbeitet werden.

Verwenden Sie möglichst wenig Effekte wie Filter, darunter auch Filter, die auf Mobilgeräten über Pixel Bender verarbeitet werden. Wenn ein Filter auf ein Anzeigeobjekt angewendet wird, erstellt die Laufzeitumgebung zwei Bitmaps im Arbeitsspeicher. Diese Bitmaps haben jeweils die Größe des Anzeigeobjekts. Die erste Bitmap wird als gerasterte Version des Anzeigeobjekts erstellt, die wiederum zur Erstellung der zweiten Bitmap mit angewendetem Filter verwendet wird:



Zwei Bitmaps im Arbeitsspeicher bei angewendetem Filter

Wenn eine Filtereigenschaft geändert wird, werden beide Bitmaps im Arbeitsspeicher aktualisiert, um die resultierende Bitmap zu erstellen. Dieser Prozess erfordert einige CPU-Ressourcen und die beiden Bitmaps können relativ viel Arbeitsspeicher belegen.

In Flash Player 10.1 und AIR 2.5 wird ein neues Filterverhalten auf allen Plattformen eingeführt. Wenn der Filter nicht innerhalb von 30 Sekunden geändert wird oder wenn er ausgeblendet oder offscreen ist, wird der Arbeitsspeicher, der von der nicht gefilterten Bitmap belegt wird, freigegeben.

Dadurch wird die erforderliche Arbeitsspeichermenge für einen Filter auf allen Plattformen halbiert. Nehmen wir als Beispiel ein Textobjekt, auf das ein Weichzeichnungsfilter angewendet wurde. Der Text dient in diesem Fall nur zu Dekorationszwecken und wird nicht geändert. Nach 30 Sekunden wird der Arbeitsspeicher, der vom nicht gefilterten Bild belegt wird, freigegeben. Dasselbe passiert, wenn der Text 30 Sekunden lang ausgeblendet oder offscreen ist. Wenn eine Filtereigenschaft geändert wird, wird die nicht gefilterte Bitmap im Arbeitsspeicher neu erstellt. Dieses Funktionsmerkmal wird dynamisches Entladen von Bitmaps genannt. Doch selbst bei Verwendung dieser Optimierungsverfahren sollten Sie Filter nur spärlich einsetzen, da sie bei einer Änderung eine hohe CPU- oder GPU-Verarbeitungsleistung erfordern.

Verwenden Sie vorzugsweise Bitmaps, die mit einem Authoring-Tool wie Adobe® Photoshop® erstellt wurden, um Filter zu emulieren. Verwenden Sie nach Möglichkeit keine dynamischen Bitmaps, die zur Laufzeit in ActionScript erstellt werden. Bei Verwendung von extern erstellten Bitmaps kann die Laufzeitumgebung die CPU- oder GPU-Belastung deutlich reduzieren, besonders wenn die Filtereigenschaften sich im Zeitverlauf nicht ändern. Erstellen Sie alle für eine Bitmap erforderlichen Effekte in einem Authoring-Tool, soweit möglich. Sie können die Bitmap dann in der Laufzeitumgebung anzeigen, ohne dass Verarbeitung dafür erforderlich ist, der Vorgang also deutlich beschleunigt werden kann.

Direktes Mipmapping

Mit dem Mipmapping können Sie große Bilder bei Bedarf skalieren.

Eine weitere neue Funktion in Flash Player 10.1 und AIR 2.5 auf allen Plattformen hat mit dem Mipmapping zu tun. In Flash Player 9 und AIR 1.0 wurde eine Mipmappingfunktion eingeführt, die die Qualität und Leistung von herunterskalierten Bitmaps verbesser hat.

Hinweis: *Das Mipmapping gilt nur für dynamisch geladene Bilder oder für eingebettete Bitmaps. Mipmapping ist nicht für Anzeigeobjekte verfügbar, die gefiltert oder zwischengespeichert wurden. Das Mipmapping kann nur verarbeitet werden, wenn Breite und Höhe der Bitmap einen geraden Wert aufweisen. Wird ein ungerader Wert für die Höhe oder Breite erkannt, wird das Mipmapping gestoppt. So kann ein Bild mit den Abmessungen 250 x 250 zwar per Mipmapping auf 125 x 125 verkleinert werden, ein weiteres Mipmapping ist jedoch nicht mehr möglich, da es sich bei mindestens einer Abmessung um eine ungerade Zahl handelt. Die besten Ergebnisse werden bei Bitmaps erzielt, deren Abmessungen eine Potenz von 2 sind, wie beispielsweise 256 x 256, 512 x 512 und 1024 x 1024.*

Angenommen, ein Bild mit den Abmessungen 1024 x 1024 wird geladen und soll für ein Miniaturbild in einer Galerie verkleinert werden. Beim Mipmapping wird das skalierte Bild korrekt gerendert, indem die per Downsampling erzielten Zwischenversionen der Bitmap als Texturen verwendet werden. Frühere Versionen der Laufzeitumgebung haben herunterskalierte Zwischenversionen der Bitmap im Arbeitsspeicher erstellt. Wenn ein Bild mit den Abmessungen 1024 x 1024 geladen und mit einer Größe von 64 x 64 angezeigt wurde, wurde in früheren Versionen der Laufzeitumgebung jede Bitmap mit der halben Größe erstellt; in diesem Beispiel wären dies Bitmaps mit den Abmessungen 512 x 512, 256 x 256, 128 x 128 und 64 x 64.

Flash Player 10.1 und AIR 2.5 unterstützen jetzt das direkte Mipmapping von der Quelle zur erforderlichen Zielgröße. In unserem Beispiel würden nur die Originalbitmap mit 4 MB (1024 x 1024) und die per Mipmapping erstellte Bitmap mit 16 KB (64 x 64) erstellt.

Die Mipmapping-Logik kann auch mit dem dynamischen Entladen von Bitmaps eingesetzt werden. Wenn nur die Bitmap mit der Größe 64 x 64 verwendet wird, wird die ursprüngliche 4-MB-Bitmap im Arbeitsspeicher freigegeben. Wenn die Mipmapping-Bitmap neu erstellt werden muss, wird das Original neu geladen. Wenn weitere Mipmapping-Versionen der Bitmap in verschiedenen Größen benötigt werden, wird dazu die per Mipmapping erstellte Bitmap-Kette verwendet. Zum Beispiel: Wenn eine 1:8-Bitmap benötigt wird, werden die Bitmaps in den Größen 1:4, 1:2 und 1:1 überprüft, um festzustellen, welche Bitmap zuerst in den Arbeitsspeicher geladen wird. Wenn keine anderen Versionen gefunden werden, wird die Originalbitmap mit 1:1 geladen und verwendet.

Die JPEG-Dekomprimierung kann das Mipmapping im eigenen Format durchführen. Durch dieses direkte Mipmapping kann eine große Bitmap direkt in das Mipmap-Format dekomprimiert werden, ohne dass das ganze dekomprimierte Bild geladen werden muss. Die Mipmap wird wesentlich schneller erstellt und von großen Bitmaps belegter Speicher muss nicht zugewiesen und dann freigegeben werden. Die JPEG-Bildqualität ist mit der allgemeinen Mipmapping-Technik vergleichbar.

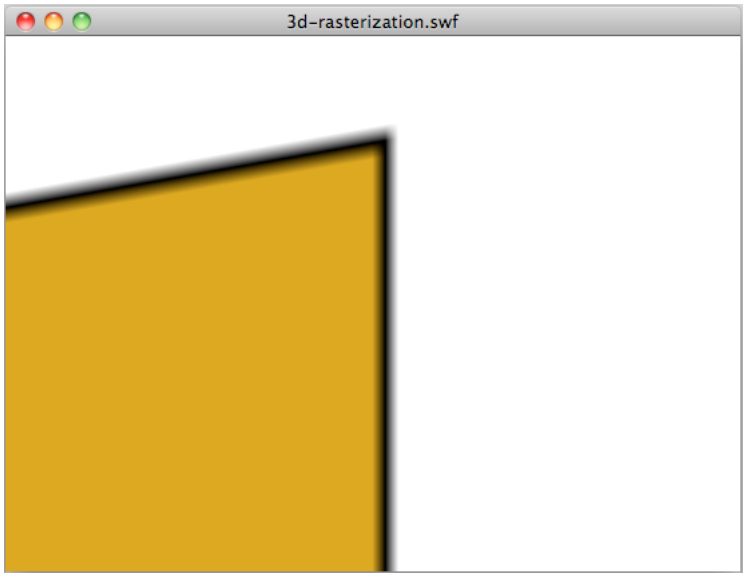
Hinweis: *Verwenden Sie das Mipmapping nicht zu häufig. Es verbessert zwar die Qualität verkleinerter Bitmaps, wirkt sich jedoch auf Bandbreite, Arbeitsspeicher und Geschwindigkeit aus. In manchen Fällen ist es besser, eine bereits skalierte Version der Bitmap aus einem externen Tool in Ihre Anwendung zu importieren. Beginnen Sie nicht mit großen Bitmaps, wenn Sie von Anfang an vorhaben, sie zu verkleinern.*

3D-Effekte


In manchen Fällen kann es empfehlenswert sein, 3D-Effekte manuell zu erstellen.

In Flash Player 10 und AIR 1.5 wurde eine 3D-Engine eingeführt, mit der Sie eine perspektivische Transformation auf Anzeigeobjekte anwenden können. Sie können diese Transformationen mit den Eigenschaften `rotationX` und `rotationY`oder mit der `drawTriangles()`-Methode der Graphics-Klasse anwenden. Außerdem können Sie Tiefe über die `z`-Eigenschaft anwenden. Beachten Sie jedoch, dass jedes perspektivisch transformierte Anzeigeobjekt als Bitmap gerastert wird und deshalb mehr Arbeitsspeicher belegt.

Die folgende Abbildung veranschaulicht das Anti-Aliasing, das bei Verwendung der perspektivischen Transformation durch das Rastern entsteht:



Resultierendes Anti-Aliasing nach perspektivischer Transformation

Das Anti-Aliasing entsteht, da Vektorinhalt dynamisch als Bitmap gerastert wird. Dieses Anti-Aliasing erfolgt, wenn Sie 3D-Effekte in der Desktopversion von AIR und Flash Player verwenden, sowie in AIR 2.0.1 und AIR 2.5 für mobile Geräte. Anti-Aliasing wird jedoch nicht in Flash Player für mobile Geräte angewendet.

Bei einer manuellen Erstellung des 3D-Effekts ohne Verwendung der nativen API wird möglicherweise weniger Arbeitsspeicher benötigt. Die in Flash Player 10 und AIR 1.5 eingeführten neuen 3D-Funktionen vereinfachen jedoch die Texturzuordnung durch Methoden wie `drawTriangles()`, die die Texturzuordnung nativ verarbeiten.

Entwickler müssen entscheiden, ob der gewünschte 3D-Effekt leistungsfähiger ist, wenn er manuell oder über die native API verarbeitet wird. Dabei müssen auch Aspekte wie die ActionScript-Ausführung, die Renderleistung sowie die Arbeitsspeicherbelegung berücksichtigt werden.

In mobilen AIR 2.0.1- und AIR 2.5-Anwendungen, in denen Sie die `renderMode`-Anwendungseigenschaft auf `GPU` einstellen, übernimmt die GPU die 3D-Transformationen. Wenn `renderMode` jedoch den Wert `CPU` hat, übernimmt die CPU und nicht die GPU die 3D-Transformationen. In Flash Player 10.1-Anwendungen führt die CPU die 3D-Transformationen aus.

Wenn die CPU die 3D-Transformationen ausführt, bedenken Sie, dass die Anwendung von 3D-Transformationen auf ein Anzeigeobjekt zwei Bitmaps im Arbeitsspeicher erfordert. Eine Bitmap ist für die Quellbitmap und eine zweite für die perspektivisch transformierte Version. So funktionieren 3D-Transformationen ähnlich wie Filter. Deshalb sollten Sie 3D-Eigenschaften sparsam einsetzen, wenn die CPU die 3D-Transformationen ausführt.

Textobjekte und Arbeitsspeicher

Verwenden Sie die Adobe® Flash® Text Engine für schreibgeschützten Text; verwenden Sie TextField-Objekte für Eingabetext.

In Flash Player 10 und AIR 1.5 wurde eine leistungsstarke neue Text-Engine eingeführt: die Adobe Flash Text Engine (FTE), die weniger Arbeitsspeicher benötigt. Bei der FTE handelt es sich jedoch um eine API auf niedriger Systemebene, die eine übergeordnete ActionScript 3.0-Ebene erfordert, bereitgestellt im flash.text.engine-Paket.

Bei schreibgeschütztem Text verwenden Sie am besten die Flash Text Engine, die weniger Arbeitsspeicher benötigt und besseres Rendering bietet. Für Eingabetext sind TextField-Objekte besser geeignet, da sie weniger ActionScript-Code erfordern, um typisches Verhalten zu erstellen, wie beispielsweise die Eingabeverarbeitung und den Zeilenumbruch.

Verwandte Themen

„Rendern von Textobjekten“ auf Seite 69

Ereignismodell und Rückrufe

Ziehen Sie die Verwendung einfacher Rückrufe anstelle des Ereignismodells in Betracht.

Das Ereignismodell von ActionScript 3.0 basiert auf dem Konzept der Objektauslösung. Das Ereignismodell ist objektorientiert und für die Wiederverwendung von Code optimiert. Die `dispatchEvent()`-Methode durchläuft die Liste der Listener in einer Schleife und ruft für jedes registrierte Objekt die Ereignisprozedurmethode auf. Das Ereignismodell hat jedoch den Nachteil, dass im Lauf der Zeit wahrscheinlich zahlreiche Objekte für die Anwendung erstellt werden.

Angenommen, Sie müssen ein Ereignis von der Zeitleiste auslösen, um das Ende einer Animationssequenz anzugeben. Für diese Benachrichtigung können Sie ein Ereignis von einem bestimmten Bild in der Zeitleiste auslösen, wie im folgenden Code gezeigt:

```
dispatchEvent( new Event ( Event.COMPLETE ) );
```

Die Document-Klasse kann mit der folgenden Codezeile auf dieses Ereignis warten:

```
addEventListener( Event.COMPLETE, onAnimationComplete );
```

Dieses Verfahren ist zwar korrekt, doch kann die Verwendung des nativen Ereignismodells mehr Ausführungszeit und Arbeitsspeicher erfordern als eine herkömmliche Rückruffunktion. Ereignisobjekte müssen erstellt und im Arbeitsspeicher zugewiesen werden, was sich negativ auf die Leistung auswirkt. So wird beispielsweise beim Warten auf das Event.ENTER_FRAME-Ereignis in jedem Bild ein neues Ereignisobjekt für die Ereignisprozedur erstellt. Die Leistung kann insbesondere bei Anzeigeobjekten leiden, da die Erfassungs- und Bubbling-Phasen bei einer komplexen Anzeigeliste sehr ressourcenintensiv sein können.

Kapitel 3: Minimieren der CPU-Auslastung

Bei den Optimierungsbestrebungen muss auch der CPU-Auslastung Beachtung eingeräumt werden. Die Optimierung der CPU-Verarbeitung verbessert die Leistung und verlängert damit die Akkulaufzeit auf Mobilgeräten.

Verbesserungen der CPU-Auslastung in Flash Player 10.1

In Flash Player 10.1 stehen zwei neue Funktionsmerkmale zur Verfügung, mit denen die CPU-Verarbeitungsleistung verbessert werden kann. Zu den Funktionen gehören das Anhalten und Fortsetzen von SWF-Inhalten, wenn sie sich außerhalb des Bildschirms befinden, und die Begrenzung der Anzahl Flash Player-Instanzen auf einer Seite.

Anhalten, Drosseln und Fortsetzen

Hinweis: *Die Funktionalität für das Anhalten, Drosseln und Fortsetzen gilt nicht für Adobe® AIR®-Anwendungen.*

Zur Optimierung der CPU- und Akkuauslastung steht in Flash Player 10.1 ein neues Funktionsmerkmal für inaktive Instanzen zur Verfügung. Mit diesem Funktionsmerkmal können Sie die CPU-Auslastung verringern, indem Sie eine SWF-Datei anhalten und wieder starten, wenn der Inhalt auf dem Bildschirm ein- oder ausgeblendet wird. Flash Player gibt dabei so viel Arbeitsspeicher wie möglich frei, indem Objekte entfernt werden, die neu erstellt werden können, wenn das Abspielen des Inhalts fortgesetzt wird. Der Inhalt muss vollständig vom Bildschirm ausgeblendet sein, damit er als nicht auf dem Bildschirm befindlich (offscreen) betrachtet wird.

In zwei Fällen gilt SWF-Inhalt als offscreen:

- Der Benutzer führt einen Bildlauf auf der Seite durch, wodurch der SWF-Inhalt offscreen verschoben wird.

  Wenn in diesem Fall Audio oder Video abgespielt wird, wird der Inhalt zwar weiter abgespielt, aber das Rendern wird gestoppt. Wenn kein Audio oder Video abgespielt wird, stellen Sie den HTML-Parameter `hasPriority` auf „true“ ein, um sicherzustellen, dass die Wiedergabe oder die ActionScript-Ausführung nicht angehalten wird. Beachten Sie jedoch, dass das Rendern von SWF-Inhalt angehalten wird, wenn Inhalt ausgeblendet oder offscreen ist, unabhängig vom Wert des HTML-Parameters `hasPriority`.

- Im Browser wird eine Registerkarte geöffnet, wodurch der SWF-Inhalt in den Hintergrund verschoben wird.

  In diesem Fall wird der SWF-Inhalt auf eine Geschwindigkeit zwischen 2 und 8 fps verlangsamt, oder *gedrosselt*, unabhängig vom Wert des HTML-Tags `hasPriority`. Die Wiedergabe von Audio und Video wird gestoppt und das Rendern von Inhalt wird nicht verarbeitet, es sei denn, der SWF-Inhalt wird wieder sichtbar.

Für Flash Player 11.2 und höher in Windows- und Mac-Desktopbrowsern können Sie das ThrottleEvent in Ihrer Anwendung verwenden. Flash Player setzt ein ThrottleEvent ab, wenn Flash Player die Wiedergabe anhält, drosselt oder fortsetzt.

Bei ThrottleEvent handelt es sich um ein broadcast-Ereignis. Das bedeutet, dass es von allen EventDispatcher-Objekten ausgelöst wird, die einen Listener für dieses Ereignis registriert haben. Weitere Informationen zu brodcast-Ereignissen finden Sie im Abschnitt zur DisplayObject-Klasse.

Instanzverwaltung

Hinweis: *Die Instanzverwaltungsfunktion gilt nicht für Adobe® AIR®-Anwendungen.*

Mit dem HTML-Parameter `hasPriority` kann das Laden von SWF-Dateien, die offscreen sind, verzögert werden.

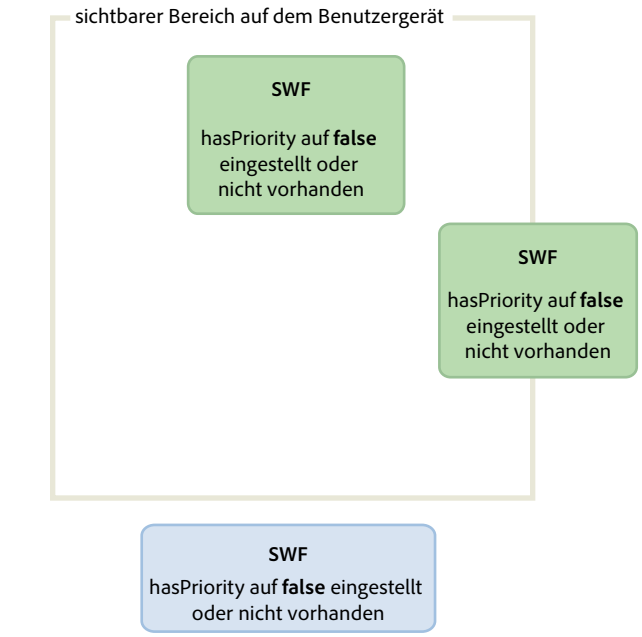
Ab Flash Player 10.1 steht der neue HTML-Parameter `hasPriority` zur Verfügung:

```
<param name="hasPriority" value="true" />
```

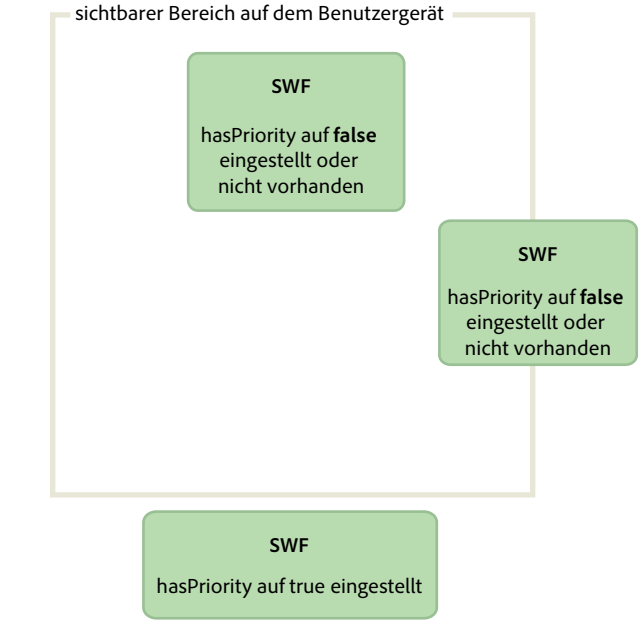
Dieses Funktionsmerkmal begrenzt die Anzahl der Flash Player-Instanzen, die auf einer Seite gestartet werden. Durch das Begrenzen der Instanzenanzahl werden CPU-Ressourcen und Batterieverbrauch eingespart. Damit kann dem SWF-Inhalt eine bestimmte Priorität im Vergleich mit anderem Inhalt auf der Seite zugewiesen werden. Ein einfaches Beispiel: Ein Benutzer navigiert in einer Website, deren Indexseite drei verschiedene SWF-Dateien umfasst. Eine dieser Dateien ist sichtbar, eine andere ist nur teilweise auf dem Bildschirm sichtbar und die dritte ist offscreen und erfordert einen Bildlauf. Die ersten beiden Animationen werden normal gestartet, aber die letzte wird verzögert, bis sie sichtbar wird. Dies ist das Standardverhalten, wenn der `hasPriority`-Parameter nicht vorhanden ist oder auf `false` eingestellt ist. Um sicherzustellen, dass eine SWF-Datei auch dann gestartet wird, wenn sie offscreen ist, setzen Sie den `hasPriority`-Parameter auf `true`. Doch unabhängig vom Wert des `hasPriority`-Parameters wird das Rendern einer SWF-Datei immer angehalten, wenn diese für den Anwender nicht sichtbar ist.

Hinweis: *Wenn nur noch wenig CPU-Ressourcen zur Verfügung stehen, werden Flash Player-Instanzen nicht mehr automatisch gestartet, selbst wenn der `hasPriority`-Parameter auf `true` eingestellt ist. Wenn nach dem Laden der Seite neue Instanzen über JavaScript erstellt werden, ignorieren diese Instanzen das `hasPriority`-Flag. Inhalt mit 1x1 Pixel oder 0x0 Pixel wird gestartet, wodurch verhindert wird, dass SWF-Helferdateien verzögert werden, wenn der Webmaster das `hasPriority`-Flag nicht angibt. SWF-Dateien können jedoch nach wie vor durch Klicken gestartet werden. Dieses Verhalten wird als „Zum Abspielen klicken“ (click to play) bezeichnet.*

In den folgenden Abbildungen werden die Auswirkungen verschiedener Werte des `hasPriority`-Parameters gezeigt:



Auswirkungen verschiedener Werte des hasPriority-Parameters



Auswirkungen verschiedener Werte des hasPriority-Parameters

Ruhemodus

In Flash Player 10.1 und AIR 2.5 wird eine neue Funktion auf mobilen Geräten eingeführt, die die CPU-Auslastung reduziert und damit auch die Akkuverwendung. Dieses Funktionsmerkmal betrifft die Hintergrundbeleuchtung, die zahlreiche Mobilgeräte bieten. Wenn beispielsweise ein Benutzer bei der Ausführung einer mobilen Anwendung unterbrochen wird und das Gerät nicht mehr benutzt, erkennt die Laufzeitumgebung, wenn die Hintergrundbeleuchtung in den Ruhemodus schaltet. Daraufhin reduziert Flash Player die Bildrate auf 4 fps (Bilder pro Sekunde) und hält das Rendern an. Bei AIR-Anwendungen beginnt auch der Energiesparmodus, wenn die Anwendung in den Hintergrund wechselt.

Der ActionScript-Code wird im Ruhemodus weiter ausgeführt, ähnlich wie beim Einstellen der `Stage.frameRate`-Eigenschaft auf 4 fps. Der Schritt zum Rendern wird jedoch übersprungen, sodass der Benutzer nicht sehen kann, dass der Player mit 4 fps weiter läuft. Die Bildrate von 4 fps anstelle von Null wurde gewählt, da so alle Verbindungen geöffnet bleiben können (NetStream, Socket und NetConnection). Beim Wechseln auf Null würden geöffnete Verbindungen unterbrochen. Außerdem wurde eine Bildwiederholfrequenz von 250 ms (4 fps) gewählt, da dieser Wert von zahlreichen Geräteherstellern verwendet wird. Mit diesem Wert bleibt auch die Bildrate der Laufzeitumgebung ungefähr im selben Bereich wie die des Geräts.

Hinweis: *Wenn sich die Laufzeitumgebung im Energiesparmodus befindet, gibt die `Stage.frameRate`-Eigenschaft die Bildrate der ursprünglichen SWF-Datei zurück und nicht 4 fps.*

Bei erneuter Aktivierung der Hintergrundbeleuchtung wird das Rendern fortgesetzt, und die Bildrate wird wieder auf den ursprünglichen Wert gesetzt. Nehmen wir als Beispiel eine Media-Player-Anwendung, in der Musik abgespielt wird. Wenn der Bildschirm in den Energiesparmodus wechselt, richtet sich die Reaktion der Laufzeitumgebung nach dem Typ des Inhalts, der gerade abgespielt wird. Die folgende Liste führt verschiedene Situationen mit dem entsprechenden Verhalten der Laufzeitumgebung auf:

- Die Hintergrundbeleuchtung wechselt in den Energiesparmodus und Inhalt ohne A/V wird abgespielt: Das Rendering wird angehalten und die Bildrate wird auf 4 fps gesetzt.
- Die Hintergrundbeleuchtung wechselt in den Ruhemodus und A/V-Inhalt wird abgespielt: Die Laufzeitumgebung erzwingt, dass die Hintergrundbeleuchtung immer eingeschaltet bleibt, sodass der Benutzer den abgespielten Inhalt weiter sehen kann.
- Die Hintergrundbeleuchtung wird aus dem Ruhemodus wieder aktiviert: Die Laufzeitumgebung stellt die Bildrate auf die ursprüngliche Einstellung der SWF-Datei ein und setzt das Rendern fort.
- Flash Player wird angehalten, während A/V-Inhalt abgespielt wird: Flash Player setzt den Status der Hintergrundbeleuchtung auf das Standardsystemverhalten zurück, da der A/V-Inhalt nicht mehr abgespielt wird.
- Das mobile Gerät empfängt einen Telefonanruf, während A/V-Inhalt abgespielt wird: Das Rendering wird angehalten und die Bildrate wird auf 4 fps gesetzt.
- Der Ruhemodus für die Hintergrundbeleuchtung wird auf dem Mobilgerät deaktiviert: Die Laufzeitumgebung zeigt das Standardverhalten.

Wenn die Hintergrundbeleuchtung in den Ruhemodus wechselt, wird das Rendering angehalten und die Bildrate reduziert. Dieses Funktionsmerkmal spart zwar CPU-Ressourcen, führt aber nicht zuverlässig eine echte Pause herbei, wie in einem Spiel.

Hinweis: *Es wird kein ActionScript-Ereignis abgesetzt, wenn die Laufzeitumgebung in den oder aus dem Ruhemodus wechselt.*

Fixieren und Freigeben von Objekten

Mithilfe der Ereignisse `REMOVED_FROM_STAGE` und `ADDED_TO_STAGE` können Sie Objekte ordnungsgemäß fixieren und wieder freigeben (ihre Fixierung wieder aufheben).

Zur Optimierung des Codes empfiehlt es sich, Objekte immer zu fixieren und wieder freizugeben. Das Fixieren und Freigeben ist für alle Objekte wichtig, ganz besonders jedoch für Anzeigeobjekte. Selbst wenn Anzeigeobjekte nicht mehr in der Anzeigeliste enthalten sind und auf die Speicherbereinigung warten, können sie immer noch CPU-lastigen Code nutzen, beispielsweise Event.ENTER_FRAME. Deshalb ist es äußerst wichtig, Objekte mithilfe der Ereignisse `Event.REMOVED_FROM_STAGE` und `Event.ADDED_TO_STAGE` ordnungsgemäß zu fixieren und freizugeben. Das folgende Beispiel zeigt einen Movieclip, der auf der Bühne abgespielt wird und eine Interaktion mit der Tastatur aufweist:

```
// Listen to keyboard events
stage.addEventListener(KeyboardEvent.KEY_DOWN, keyIsDown);
stage.addEventListener(KeyboardEvent.KEY_UP, keyIsUp);

// Create object to store key states
var keys:Dictionary = new Dictionary(true);

function keyIsDown(e:KeyboardEvent):void
{
    // Remember that the key was pressed
    keys[e.keyCode] = true;

    if (e.keyCode==Keyboard.LEFT || e.keyCode==Keyboard.RIGHT)
    {
        runningBoy.play();
    }
}

function keyIsUp(e:KeyboardEvent):void
{
    // Remember that the key was released
    keys[e.keyCode] = false;

    for each (var value:Boolean in keys)
        if ( value ) return;
    runningBoy.stop();
```

```
}

runningBoy.addEventListener(Event.ENTER_FRAME, handleMovement);
runningBoy.stop();

var currentState:Number = runningBoy.scaleX;
var speed:Number = 15;

function handleMovement(e:Event):void
{
    if (keys[Keyboard.RIGHT])
    {
        e.currentTarget.x += speed;
        e.currentTarget.scaleX = currentState;
    } else if (keys[Keyboard.LEFT])
    {
        e.currentTarget.x -= speed;
        e.currentTarget.scaleX = -currentState;
    }
}
```

Movieclip, der eine Interaktion mit der Tastatur aufweist

Beim Klicken auf die Schaltfläche „Entfernen“ wird der Movieclip aus der Anzeigeliste entfernt:

```
// Show or remove running boy
showBtn.addEventListener (MouseEvent.CLICK,showIt);
removeBtn.addEventListener (MouseEvent.CLICK,removeIt);

function showIt (e:MouseEvent):void
{
    addChild (runningBoy);
}

function removeIt(e:MouseEvent):void
{
    if (contains(runningBoy)) removeChild(runningBoy);
}
```

Auch nach dem Entfernen aus der Anzeigeliste löst der Movieclip noch das `Event.ENTER_FRAME`-Ereignis aus. Der Movieclip wird noch abgespielt, aber nicht gerendert. Um diese Situation richtig zu handhaben, muss Ihr Code auf die jeweiligen Ereignisse warten und Ereignis-Listener entfernen, damit CPU-lastiger Code nicht ausgeführt wird:

```
// Listen to Event.ADDED_TO_STAGE and Event.REMOVED_FROM_STAGE
runningBoy.addEventListener(Event.ADDED_TO_STAGE,activate);
runningBoy.addEventListener(Event.REMOVED_FROM_STAGE,deactivate);

function activate(e:Event):void
{
    // Restart everything
    e.currentTarget.addEventListener(Event.ENTER_FRAME,handleMovement);
}

function deactivate(e:Event):void
{
    // Freeze the running boy - consumes fewer CPU resources when not shown
    e.currentTarget.removeEventListener(Event.ENTER_FRAME,handleMovement);
    e.currentTarget.stop();
}
```

Beim Klicken auf die Schaltfläche „Anzeigen“ wird der Movieclip neu gestartet und wartet erneut auf die `Event.ENTER_FRAME`-Ereignisse. Der Movieclip wird korrekt von der Tastatur gesteuert.

Hinweis: *Wenn Sie ein Anzeigeobjekt aus der Anzeigeliste entfernen und danach seinen Verweis auf* `null` *setzen, ist nicht gewährleistet, dass das Objekt fixiert wird. Wenn die Speicherbereinigung nicht ausgeführt wird, belegt das Objekt weiterhin Arbeitsspeicher und CPU-Ressourcen, obwohl es nicht mehr angezeigt wird. Sie sollten das Objekt beim Entfernen aus der Anzeigeliste vollständig fixieren, um sicherzustellen, dass es so wenig CPU-Ressourcen wie möglich beansprucht.*

Ab Flash Player 10 und AIR 1.5 tritt auch das folgende Verhalten auf. Das Anzeigeobjekt wird automatisch fixiert, wenn der Abspielkopf ein leeres Bild erreicht, selbst wenn Sie kein spezifisches Fixierungsverhalten implementiert haben.

Das Fixieren ist auch beim Laden von Remote-Inhalt mit der Loader-Klasse von Bedeutung. Bei Verwendung der Loader-Klasse mit Flash Player 9 und AIR 1.0 war es notwendig, Inhalt manuell zu fixieren, indem ein Listener für das `Event.UNLOAD`-Ereignis verwendet wurde, das vom LoaderInfo-Objekt abgesetzt wurde. Jedes Objekt musste manuell fixiert werden, was mit einem hohen Arbeitsaufwand verbunden war. In Flash Player 10 und AIR 1.5 wurde eine wichtige neue Methode für die Loader-Klasse eingeführt: `unloadAndStop()`. Mit dieser Methode können Sie eine SWF-Datei entladen, automatisch jedes Objekt in der geladenen SWF-Datei fixieren und die Speicherbereinigung erzwingen.

Im folgenden Code wird die SWF-Datei geladen und dann mithilfe der `unload()`-Methode entladen, was ein manuelles Fixieren und mehr Verarbeitungsleistung erfordert:

```
var loader:Loader = new Loader();

loader.load ( new URLRequest ( "content.swf" ) );

addChild ( loader );

stage.addEventListener ( MouseEvent.CLICK, unloadSWF );

function unloadSWF ( e:MouseEvent ):void
{
    // Unload the SWF file with no automatic object deactivation
    // All deactivation must be processed manually
    loader.unload();
}
```

Es empfiehlt sich, die `unloadAndStop()`-Methode zu verwenden, die das Fixieren nativ durchführt und die Ausführung der Speicherbereinigung erzwingt:

```
var loader:Loader = new Loader();

loader.load ( new URLRequest ( "content.swf" ) );

addChild ( loader );

stage.addEventListener ( MouseEvent.CLICK, unloadSWF );

function unloadSWF ( e:MouseEvent ):void
{
    // Unload the SWF file with automatic object deactivation
    // All deactivation is handled automatically
    loader.unloadAndStop();
}
```

Die folgenden Aktionen werden beim Aufruf der `unloadAndStop()`-Methode ausgeführt:

- Sounds werden gestoppt.
- Listener, die für die Hauptzeitleiste der SWF-Datei registriert sind, werden entfernt.
- Timer-Objekte werden gestoppt.
- Peripheriegeräte (wie Kamera und Mikrofon) werden freigegeben.
- Jeder Movieclip wird gestoppt.
- Das Auslösen von `Event.ENTER_FRAME`, `Event.FRAME_CONSTRUCTED`, `Event.EXIT_FRAME`, `Event.ACTIVATE` und `Event.DEACTIVATE` wird gestoppt.

Aktivieren und Deaktivieren von Ereignissen

Mit `Event.ACTIVATE` - und `Event.DEACTIVATE`-Ereignissen können Sie Hintergrundinaktivität erkennen und Ihre Anwendung entsprechend optimieren.

Zwei Ereignisse (`Event.ACTIVATE` und `Event.DEACTIVATE`) sind bei der Feinabstimmung der Anwendung hilfreich, damit so wenig CPU-Durchläufe wie möglich verwendet werden. Mit diesen Ereignissen können Sie erkennen, wenn die Laufzeitumgebung den Fokus erhält oder verliert. So kann der Code optimiert werden, um auf Kontextänderungen zu reagieren. Im folgenden Code werden Listener für beide Ereignisse verwendet und die Bildrate wird auf null gesetzt, wenn die Anwendung den Fokus verliert. Die Anwendung kann zum Beispiel den Fokus verlieren, wenn der Benutzer auf einen anderen Tab wechselt oder die Anwendung in den Hintergrund verschiebt:

```
var originalFrameRate:uint = stage.frameRate;
var standbyFrameRate:uint = 0;

stage.addEventListener ( Event.ACTIVATE, onActivate );
stage.addEventListener ( Event.DEACTIVATE, onDeactivate );

function onActivate ( e:Event ):void
{
    // restore original frame rate
    stage.frameRate = originalFrameRate;
}

function onDeactivate ( e:Event ):void
{
    // set frame rate to 0
    stage.frameRate = standbyFrameRate;
}
```

Wenn die Anwendung wieder den Fokus erhält, wird die Bildrate auf den ursprünglichen Wert zurückgesetzt. Anstatt die Bildrate dynamisch zu ändern, können Sie auch andere Optimierungen in Betracht ziehen, zum Beispiel das Fixieren von Objekten bzw. das Aufheben der Fixierung.

Das Aktivieren und Deaktivieren von Ereignissen ermöglicht Ihnen, einen ähnlichen Mechanismus wie die Funktion „Anhalten und Fortsetzen“, die auf bestimmten mobilen Geräten und Netbooks zu finden ist, zu verwenden.

Verwandte Themen

„Bildrate der Anwendung“ auf Seite 54

„Fixieren und Freigeben von Objekten“ auf Seite 29

Mausinteraktionen

Es empfiehlt sich, Mausinteraktionen nach Möglichkeit zu deaktivieren.

Wenn Sie ein interaktives Objekt, zum Beispiel ein MovieClip- oder Sprite-Objekt, verwenden, führt die Laufzeitumgebung nativen Code aus, um Mausinteraktionen zu erkennen und zu verarbeiten. Das Erkennen von Mausinteraktionen kann die CPU stark beanspruchen, wenn zahlreiche interaktive Objekte auf dem Bildschirm angezeigt werden, besonders wenn diese Objekte sich überlappen. Dieser Verarbeitungsaufwand lässt sich auf einfache Weise vermeiden, indem die Mausinteraktionen für Objekte deaktiviert werden, die keine Mausinteraktion erfordern. Der folgende Code veranschaulicht die Verwendung der Eigenschaften `mouseEnabled` und `mouseChildren`:

```
// Disable any mouse interaction with this InteractiveObject
myInteractiveObject.mouseEnabled = false;
const MAX_NUM:int = 10;

// Create a container for the InteractiveObjects
var container:Sprite = new Sprite();

for ( var i:int = 0; i< MAX_NUM; i++ )
{
    // Add InteractiveObject to the container
    container.addChild( new Sprite() );
}

// Disable any mouse interaction on all the children
container.mouseChildren = false;
```

Nach Möglichkeit sollten Sie Mausinteraktionen deaktivieren, da die Anwendung so weniger CPU-Ressourcen erfordert und die Akkulaufzeit von mobilen Geräten sich verlängert.

Timer und ENTER_FRAME-Ereignisse im Vergleich

Wählen Sie entweder Timer oder `ENTER_FRAME`-Ereignisse, je nachdem, ob der Inhalt animiert ist oder nicht.

Bei nicht animiertem Inhalt, der über einen langen Zeitraum ausgeführt wird, sind Timer den `Event.ENTER_FRAME`-Ereignissen vorzuziehen.

In ActionScript 3.0 gibt es zwei Möglichkeiten, um eine Funktion in bestimmten Intervallen aufzurufen. Die erste Technik ist die Verwendung des `Event.ENTER_FRAME`-Ereignisses, das von Anzeigeobjekten ausgelöst wird (DisplayObject). Beim zweiten Verfahren wird ein Timer verwendet. ActionScript-Entwickler verwenden häufig das `ENTER_FRAME`-Ereignis. Das `ENTER_FRAME`-Ereignis wird für jedes Bild ausgelöst. Deshalb bezieht sich das Intervall, in dem die Funktion aufgerufen wird, auf die aktuelle Bildrate. Die Bildrate kann über die `Stage.frameRate`-Eigenschaft bestimmt werden. Manchmal ist ein Timer jedoch dem `ENTER_FRAME`-Ereignis vorzuziehen. Wenn Sie beispielsweise keine Animation verwenden, der Code aber in bestimmten Intervallen aufgerufen werden soll, ist ein Timer möglicherweise die bessere Wahl.

Ein Timer kann sich ähnlich wie ein `ENTER_FRAME`-Ereignis verhalten, doch ein Ereignis kann ausgelöst werden, ohne dass es an die Bildrate gebunden ist. Über dieses Verhalten lassen sich in manchen Fällen deutliche Optimierungsvorteile erzielen. Nehmen wir als Beispiel eine Videoplayer-Anwendung. In diesem Fall wird keine hohe Bildrate benötigt, da nur die Anwendungssteuerungen sich bewegen.

***Hinweis:** Die Bildrate wirkt sich nicht auf das Video aus, da das Video nicht in der Zeitleiste eingebettet ist. Stattdessen wird das Video dynamisch über den progressiven Download oder per Streaming geladen.*

In diesem Beispiel ist die Bildrate mit 10 fps auf einen niedrigen Wert eingestellt. Der Timer aktualisiert die Steuerungen mit einer Rate von einer Aktualisierung pro Sekunde. Die höhere Aktualisierungsrate wird durch die `updateAfterEvent()`-Methode ermöglicht, die für das TimerEvent-Objekt zur Verfügung steht. Diese Methode erzwingt immer dann eine Bildschirmaktualisierung, wenn der Timer ein Ereignis auslöst (sofern erforderlich). Dieses Konzept wird im folgenden Code veranschaulicht:

```
// Use a low frame rate for the application
stage.frameRate = 10;

// Choose one update per second
var updateInterval:int = 1000;
var myTimer:Timer = new Timer(updateInterval,0);

myTimer.start();
myTimer.addEventListener( TimerEvent.TIMER, updateControls );

function updateControls( e:TimerEvent ):void
{
    // Update controls here
    // Force the controls to be updated on screen
    e.updateAfterEvent();
}
```

Durch Aufruf der `updateAfterEvent()`-Methode ändert sich die Bildrate nicht. Die Methode zwingt die Laufzeitumgebung nur, den Inhalt auf dem Bildschirm zu aktualisieren, der sich geändert hat. Die Zeitleiste läuft immer noch mit einer Bildrate von 10 fps. Beachten Sie, dass Timer und `ENTER_FRAME`-Ereignisse auf Geräten mit niedriger Leistung nicht hundertprozentig genau sind, und auch dann nicht, wenn Ereignisprozedurfunktionen verarbeitungsintensiven Code enthalten. Genau wie die Bildrate der SWF-Datei kann auch die Aktualisierungsbildrate des Timers in manchen Situationen variieren.

Halten Sie die Anzahl der Timer-Objekte und registrierten `enterFrame`-Prozeduren in Ihrer Anwendung gering.

In jedem Bild setzt die Laufzeitumgebung ein `enterFrame`-Ereignis an alle Anzeigeobjekte in ihrer Anzeigeliste ab. Sie können zwar Listener für das `enterFrame`-Ereignis mit mehreren Anzeigeobjekten registrieren, dies bedeutet jedoch, dass in jedem Bild mehr Code ausgeführt wird. Überlegen Sie stattdessen, eine einzelne zentrale `enterFrame`-Prozedur zu verwenden, die den gesamten Code ausführt, der in jedem Bild ausgeführt werden soll. Dadurch kann häufig ausgeführter Code einfacher verwaltet werden.

Auf ähnliche Weise entsteht beim Verwenden von Timer-Objekten eine höhere Verarbeitungslast durch das Erstellen und Absetzen von Ereignissen aus mehreren Timer-Objekten. Wenn unterschiedliche Operationen in verschiedenen Intervallen ausgelöst werden müssen, kommen folgende Alternativen infrage:

- Verwenden Sie möglichst wenige Timer-Objekte und gruppieren Sie Operationen nach der Häufigkeit ihres Auftretens..

  Verwenden Sie zum Beispiel einen Timer für häufige Operationen, der alle 100 Millisekunden ausgelöst wird. Verwenden Sie einen anderen Timer für weniger häufige oder Hintergrundoperationen, der alle 2000 Millisekunden ausgelöst wird.

- Verwenden Sie ein einzelnes Timer-Objekt und lassen Sie Operationen zu Vielfachen der `delay`-Eigenschaft des Timer-Objekts auslösen.

  Zum Beispiel könnten Sie einige Operationen haben, die alle 100 Millisekunden ausgeführt werden sollen, und anderen, die alle 200 Millisekunden ausgeführt werden sollen. Verwenden Sie in diesem Fall ein einzelnes Timer-Objekt mit einem `delay`-Wert von 100 Millisekunden. Fügen Sie in der `timer`-Ereignisprozedur eine bedingte Anweisung hinzu, die die 200-Millisekunden-Operation nur jedes zweite Mal ausführt. Diese Vorgehensweise wird im folgenden Beispiel veranschaulicht:

```
var timer:Timer = new Timer(100);
timer.addEventListener(TimerEvent.Timer, timerHandler);
timer.start();

var offCycle:Boolean = true;

function timerHandler(event:TimerEvent):void
{
    // Do things that happen every 100 ms

    if (!offCycle)
    {
        // Do things that happen every 200 ms
    }

    offCycle = !offCycle;
}
```

Beenden Sie Timer-Objekte, die nicht verwendet werden.

Wenn die `timer`-Ereignisprozedur eines Timer-Objekts Operationen nur unter bestimmten Bedingungen ausführt, rufen Sie die `stop()`-Methode des Timer-Objekts auf, wenn keine der Bedingungen erfüllt ist.

Minimieren Sie in `enterFrame`-Ereignisprozeduren oder Timer-Prozeduren die Anzahl der Änderungen am Erscheinungsbild von Anzeigeobjekten, die zum Neuzeichnen des Bildschirms führen.

In jedem Bild zeichnet die Renderphase den Teil der Bühne neu, der in diesem Bild geändert wurde. Wenn der neu zu zeichnende Bereich groß ist oder zwar klein ist, aber zahlreiche oder komplexe Anzeigeobjekte enthält, braucht die Laufzeitumgebung mehr Zeit für das Rendering. Um zu testen, wie viel neu gezeichnet werden muss, verwenden Sie die Funktion „Bildaktualisierungsbereiche anzeigen“ in der Debugversion von Flash Player oder AIR.

Weitere Informationen zum Verbessern der Leistung bei wiederholten Aktionen finden Sie im folgenden Artikel:

- Writing well-behaved, efficient, AIR applications (Artikel und Beispielanwendung von Arno Gourdol)

Verwandte Themen

„Isolieren von Verhalten“ auf Seite 66

Tweening-Syndrom

Setzen Sie das Tweening nur spärlich ein, um CPU-Ressourcen und Arbeitsspeicher einzusparen und die Akkulaufzeit zu verlängern.

Designer und Entwickler von Inhalt für Flash auf dem Desktop statten ihre Anwendungen häufig mit zahlreichen Bewegungs-Tweens aus. Bei der Erstellung von Inhalt für Mobilgeräte mit niedriger Leistung sollten Sie so weit wie möglich auf Bewegungs-Tweens verzichten. Dadurch kann Inhalt auf Geräten der unteren Leistungsklasse schneller ausgeführt werden.

Kapitel 4: Leistung in ActionScript 3.0

Vector-Klasse und Array-Klasse im Vergleich

Verwenden Sie nach Möglichkeit die Vector-Klasse anstelle der Array-Klasse.

Die Vector-Klasse ermöglicht schnelleren Lese- und Schreibzugriff als die Array-Klasse.

Ein einfacher Vergleichswert verdeutlicht die Vorteile der Vector-Klasse gegenüber der Array-Klasse. Der folgende Code zeigt einen Vergleichswert für die Array-Klasse:

```
var coordinates:Array = new Array();
var started:Number = getTimer();

for (var i:int = 0; i< 300000; i++)
{
    coordinates[i] = Math.random()*1024;
}

trace(getTimer() - started);
// output: 107
```

Der folgende Code zeigt einen Vergleichswert für die Vector-Klasse:

```
var coordinates:Vector.<Number> = new Vector.<Number>();
var started:Number = getTimer();

for (var i:int = 0; i< 300000; i++)
{
    coordinates[i] = Math.random()*1024;
}

trace(getTimer() - started);
// output: 72
```

Das Beispiel kann noch weiter optimiert werden, indem dem Vektor eine bestimmte Länge als fester Wert zugewiesen wird:

```
// Specify a fixed length and initialize its length
var coordinates:Vector.<Number> = new Vector.<Number>(300000, true);

var started:Number = getTimer();

for (var i:int = 0; i< 300000; i++)
{
    coordinates[i] = Math.random()*1024;
}

trace(getTimer() - started);
// output: 48
```

Wenn die Vektorgröße nicht vorab angegeben wird, nimmt die Größe zu, wenn der Vektor nicht mehr über ausreichend Platz verfügt. Jedes Mal, wenn die Vektorgröße zunimmt, wird ein neuer Arbeitsspeicherblock zugewiesen. Der aktuelle Inhalt des Vektors wird in den neuen Arbeitsspeicherblock kopiert. Diese zusätzliche Speicherzuweisung und das Kopieren der Daten wirken sich negativ auf die Leistung aus. Der oben gezeigte Code ist hinsichtlich der Leistung optimiert, da die anfängliche Vektorgröße angegeben wird. Der Code ist jedoch nicht für die Pflegefreundlichkeit optimiert. Um auch die Codepflege zu vereinfachen, muss der wieder verwendete Wert in einer Konstante gespeichert werden:

```
// Store the reused value to maintain code easily
const MAX_NUM:int = 300000;

var coordinates:Vector.<Number> = new Vector.<Number>(MAX_NUM, true);
var started:Number = getTimer();

for (var i:int = 0; i< MAX_NUM; i++)
{
    coordinates[i] = Math.random()*1024;
}

trace(getTimer() - started);
// output: 47
```

Verwenden Sie nach Möglichkeit die APIs für Vector-Objekte, da sie wahrscheinlich schneller ausgeführt werden können.

Zeichnungs-API

Verwenden Sie die Zeichnungs-API für eine schnellere Ausführung des Codes.

Flash Player 10 und AIR 1.5 haben eine neue Zeichnungs-API bereitgestellt, mit der eine bessere Leistung bei der Codeausführung erzielt wurde. Diese neue API führt zwar nicht zu einer höheren Leistung beim Rendern, kann jedoch die Anzahl der erforderlichen Codezeilen deutlich verringern. Weniger Codezeilen können wiederum zu einer höheren ActionScript-Ausführungsleistung führen.

Die neue Zeichnungs-API enthält die folgenden Methoden:

- `drawPath()`
- `drawGraphicsData()`
- `drawTriangles()`

Hinweis: *Dieses Thema befasst sich nicht mit der `drawTriangles()`-Methode, die sich auf 3D bezieht. Diese Methode kann jedoch die ActionScript-Leistung verbessern, da sie die native Texturzuordnung verarbeitet.*

Mit dem folgenden Code wird die passende Methode für jede gezeichnete Linie explizit aufgerufen:

```
var container:Shape = new Shape();
container.graphics.beginFill(0x442299);

var coords:Vector.<Number> = Vector.<Number>([132, 20, 46, 254, 244, 100, 20, 98, 218, 254]);

container.graphics.moveTo ( coords[0], coords[1] );
container.graphics.lineTo ( coords[2], coords[3] );
container.graphics.lineTo ( coords[4], coords[5] );
container.graphics.lineTo ( coords[6], coords[7] );
container.graphics.lineTo ( coords[8], coords[9] );

addChild( container );
```

Der folgende Code wird schneller ausgeführt als das vorherige Beispiel, da weniger Codezeilen ausgeführt werden. Je komplexer der Pfad, desto höher die Leistungssteigerungen, die mit der `drawPath()`-Methode erzielt werden können:

```
var container:Shape = new Shape();
container.graphics.beginFill(0x442299);

var commands:Vector.<int> = Vector.<int>([1,2,2,2,2]);
var coords:Vector.<Number> = Vector.<Number>([132, 20, 46, 254, 244, 100, 20, 98, 218, 254]);

container.graphics.drawPath(commands, coords);

addChild( container );
```

Die `drawGraphicsData()`-Methode liefert vergleichbare Leistungssteigerungen.

Ereigniserfassung und Bubbling

Mithilfe von Ereigniserfassung und Bubbling lassen sich Ereignisprozeduren reduzieren.

Seit dem Ereignismodell von ActionScript 3.0 stehen die Konzepte „Ereigniserfassung“ und „Ereignis-Bubbling“ zur Verfügung. Das Ereignis-Bubbling bietet die Möglichkeit, die Ausführungszeit des ActionScript-Codes zu optimieren. Sie können eine Ereignisprozedur für ein Objekt anstelle für mehrere Objekte registrieren, um die Leistung zu verbessern.

Dies lässt sich anhand eines Spiels erläutern, in dem der Anwender so schnell wie möglich auf Äpfel klicken muss, um sie zu zerstören. Bei jedem Klicken auf einen Apfel wird dieser vom Bildschirm entfernt und der Anwender erhält Punkte. Zum Warten auf das von den einzelnen Äpfeln ausgelöste `MouseEvent.CLICK`-Ereignis mag der folgende Code geeignet erscheinen:

```
const MAX_NUM:int = 10;
var sceneWidth:int = stage.stageWidth;
var sceneHeight:int = stage.stageHeight;
var currentApple:InteractiveObject;
var currentAppleClicked:InteractiveObject;

for ( var i:int = 0; i< MAX_NUM; i++ )
{
    currentApple = new Apple();
    currentApple.x = Math.random()*sceneWidth;
    currentApple.y = Math.random()*sceneHeight;
    addChild ( currentApple );

    // Listen to the MouseEvent.CLICK event
    currentApple.addEventListener ( MouseEvent.CLICK, onAppleClick );
}

function onAppleClick ( e:MouseEvent ):void
{
    currentAppleClicked = e.currentTarget as InteractiveObject;
    currentAppleClicked.removeEventListener(MouseEvent.CLICK, onAppleClick );
    removeChild ( currentAppleClicked );
}
```

Der Code ruft die `addEventListener()`-Methode für jede Apple-Instanz auf. Außerdem entfernt er beim Klicken auf einen Apfel den zugehörigen Listener unter Verwendung der `removeEventListener()`-Methode. Das Ereignismodell in ActionScript 3.0 bietet jedoch eine Erfassungs- und Bubbling-Phase für einige Ereignisse, sodass Sie über ein übergeordnetes InteractiveObject auf diese Ereignisse warten können. Deshalb ist es möglich, den oben gezeigten Code zu optimieren und die Methoden `addEventListener()` und `removeEventListener()` weniger oft aufzurufen. Der folgende Code verwendet die Erfassungsphase, um auf die Ereignisse vom übergeordneten Objekt zu warten:

```
const MAX_NUM:int = 10;
var sceneWidth:int = stage.stageWidth;
var sceneHeight:int = stage.stageHeight;
var currentApple:InteractiveObject;
var currentAppleClicked:InteractiveObject;
var container:Sprite = new Sprite();

addChild ( container );

// Listen to the MouseEvent.CLICK on the apple's parent
// Passing true as third parameter catches the event during its capture phase
container.addEventListener ( MouseEvent.CLICK, onAppleClick, true );

for ( var i:int = 0; i< MAX_NUM; i++ )
{
    currentApple = new Apple();
    currentApple.x = Math.random()*sceneWidth;
    currentApple.y = Math.random()*sceneHeight;
    container.addChild ( currentApple );
}

function onAppleClick ( e:MouseEvent ):void
{
    currentAppleClicked = e.target as InteractiveObject;
    container.removeChild ( currentAppleClicked );
}
```

Dieser optimierte Code ist wesentlich einfacher und erfordert nur einen Aufruf der `addEventListener()`-Methode für den übergeordneten Container. Da keine Listener mehr für die Apple-Instanzen registriert sind, müssen sie beim Klicken auf einen Apfel auch nicht mehr entfernt werden. Die `onAppleClick()`-Prozedur kann noch weiter optimiert werden, indem die Fortsetzung des Ereignisses gestoppt wird:

```
function onAppleClick ( e:MouseEvent ):void
{
    e.stopPropagation();
    currentAppleClicked = e.target as InteractiveObject;
    container.removeChild ( currentAppleClicked );
}
```

Auch die Bubbling-Phase kann zum Abfangen des Ereignisses verwendet werden, indem `false` als dritter Parameter an die `addEventListener()`-Methode übergeben wird:

```
// Listen to the MouseEvent.CLICK on apple's parent
// Passing false as third parameter catches the event during its bubbling phase
container.addEventListener ( MouseEvent.CLICK, onAppleClick, false );
```

De Standardwert für den Parameter der Erfassungsphase ist `false`; Sie können ihn also weglassen:

```
container.addEventListener ( MouseEvent.CLICK, onAppleClick );
```

Arbeiten mit Pixeln

Verwenden Sie zum Zeichnen von Pixeln die `setVector()`-Methode.

Beim Zeichnen von Pixeln lassen sich einige einfache Optimierungen erzielen, indem die geeigneten Methoden der BitmapData-Klasse verwendet werden. Eine schnelle Möglichkeit, Pixel zu zeichnen, ist die Verwendung der `setVector()`-Methode:

```
// Image dimensions
var wdth:int = 200;
var hght:int = 200;
var total:int = wdth*hght;

// Pixel colors Vector
var pixels:Vector.<uint> = new Vector.<uint>(total, true);

for ( var i:int = 0; i< total; i++ )
{
    // Store the color of each pixel
    pixels[i] = Math.random()*0xFFFFFF;
}

// Create a non-transparent BitmapData object
var myImage:BitmapData = new BitmapData ( wdth, hght, false );
var imageContainer:Bitmap = new Bitmap ( myImage );

// Paint the pixels
myImage.setVector ( myImage.rect, pixels );
addChild ( imageContainer );
```

Bei langsamen Methoden, wie `setPixel()` oder `setPixel32()`, sollten Sie die Methoden `lock()` und `unlock()` verwenden, um die Ausführung zu beschleunigen. Im folgenden Code werden die Methoden `lock()` und `unlock()` verwendet, um die Leistung zu erhöhen:

```
var buffer:BitmapData = new BitmapData(200,200,true,0xFFFFFFFF);
var bitmapContainer:Bitmap = new Bitmap(buffer);
var positionX:int;
var positionY:int;

// Lock update
buffer.lock();
var starting:Number=getTimer();

for (var i:int = 0; i<2000000; i++)
{
    // Random positions
    positionX = Math.random()*200;
    positionY = Math.random()*200;
    // 40% transparent pixels
    buffer.setPixel32( positionX, positionY, 0x66990000 );
}

// Unlock update
buffer.unlock();
addChild( bitmapContainer );

trace( getTimer () - starting );
// output : 670
```

Die `lock()`-Methode der BitmapData-Klasse sperrt ein Bild und verhindert, dass Objekte, die auf das Bild verweisen, bei einer Änderung des BitmapData-Objekts aktualisiert werden. Wenn beispielsweise ein Bitmap-Objekt auf ein BitmapData-Objekt verweist, können Sie das BitmapData-Objekt sperren, ändern und dann wieder entsperren. Das Bitmap-Objekt wird erst geändert, nachdem das BitmapData-Objekt entsperrt wurde. Um die Leistung zu verbessern, verwenden Sie diese Methode zusammen mit der `unlock()`-Methode vor und nach mehreren Aufrufen der Methode `setPixel()` oder `setPixel32()`. Durch Aufrufen der Methoden `lock()` und `unlock()` werden unnötige Aktualisierungen des Bildschirms vermieden.

Hinweis: *Bei der Verarbeitung von Pixeln in einer Bitmap, die sich nicht in der Anzeigeliste befindet (doppelte Pufferung), führt diese Technik nicht in allen Fällen zu Leistungssteigerungen. Wenn ein Bitmap-Objekt nicht auf den Bitmap-Puffer verweist, bewirkt die Verwendung von `lock()` und `unlock()` keine höhere Leistung. Flash Player erkennt, dass nicht auf den Puffer verwiesen wird, und die Bitmap wird nicht auf dem Bildschirm gerendert.*

Methoden, die iterativ über Pixel ausgeführt werden, wie beispielsweise `getPixel()`, `getPixel32()`, `setPixel()` und `setPixel32()`, sind normalerweise langsam, besonders auf Mobilgeräten. Verwenden Sie nach Möglichkeit Methoden, die alle Pixel in einem Aufruf abrufen. Verwenden Sie zum Lesen von Pixeln die `getVector()`-Methode, die schneller als die `getPixels()`-Methode ist. Verwenden Sie nach Möglichkeit auch APIs, die sich auf Vector-Objekte stützen, da sie in aller Regel schneller ausgeführt werden.

Reguläre Ausdrücke

Verwenden Sie Methoden der String-Klasse wie zum Beispiel `indexOf()`, `substr()` oder `substring()` anstelle von regulären Ausdrücken, um grundlegende Vorgänge zum Suchen und Extrahieren von Strings auszuführen.

Bestimmte Operationen, die mithilfe regulärer Ausdrücke ausgeführt werden, lassen sich auch mit Methoden der String-Klasse ausführen. Um zum Beispiel herauszufinden, ob ein String einen anderen String enthält, können Sie entweder die `String.indexOf()`-Methode oder einen regulären Ausdruck verwenden. Wenn jedoch eine Methode der String-Klasse verfügbar ist, erfolgt die Ausführung schneller als mit dem entsprechenden regulären Ausdruck und es braucht kein weiteres Objekt erstellt zu werden.

Verwenden Sie eine nicht erfassende Gruppe („`(?:xxxx)`“) anstelle einer Gruppe („`(xxxx)`“) in einem regulären Ausdruck, wenn Sie Elemente gruppieren möchten, ohne den Inhalt der Gruppe im Ergebnis zu isolieren.

In regulären Ausdrücken, die nicht zu komplex sind, gruppieren Sie häufig Teile des Ausdrucks. Im folgenden regulären Ausdruck wird zum Beispiel mit Klammern eine Gruppe um den Text „ab“ erstellt. Dementsprechend gilt der Quantifizierer „+“ für die ganze Gruppe statt nur für ein einzelnes Zeichen:

```
/(ab)+/
```

Standardmäßig wird der Inhalt jeder Gruppe „erfasst“. Sie können den Inhalt jeder Gruppe in Ihrem Muster als Teil des Ergebnisses der Ausführung des regulären Ausdrucks abrufen. Das Erfassen dieser Gruppenergebnisse dauert länger und benötigt mehr Arbeitsspeicher, da Objekte erstellt werden, die die Gruppenergebnisse enthalten. Alternativ dazu können Sie die nicht erfassende Gruppensyntax verwenden, indem Sie ein Fragezeichen und einen Doppelpunkt nach der öffnenden Klammer eingeben. Diese Syntax legt fest, dass sich die Zeichen als Gruppe verhalten, aber nicht für das Ergebnis erfasst werden:

```
/(?:ab)+/
```

Die Verwendung der nicht erfassenden Gruppensyntax ist schneller und benötigt weniger Arbeitsspeicher als die Verwendung der standardmäßigen Gruppensyntax.

Verwenden Sie ggf. einen alternativen regulären Ausdruck, wenn der zunächst verwendete nicht gut funktioniert.

Manchmal lassen sich mehrere reguläre Ausdruckmuster verwenden, um dasselbe Textmuster zu finden oder zu identifizieren. Aus bestimmten Gründen werden bestimmte Muster schneller ausgeführt als andere. Wenn Sie feststellen, dass ein regulärer Ausdruck dazu führt, dass Ihr Code langsamer als nötig ausgeführt wird, sollten Sie reguläre Ausdrücke mit Alternativmustern in Betracht ziehen, die dasselbe Ergebnis liefern. Testen Sie diese alternativen Ausdruckmuster, um herauszufinden, welches am schnellsten ist.

Verschiedene Optimierungen

Verwenden Sie für ein TextField-Objekt die `appendText()`-Methode anstelle des Operators `+=`.

Beim Arbeiten mit der `text`-Eigenschaft der TextField-Klasse sollten Sie die `appendText()`-Methode anstelle des Operators `+=` verwenden. Die `appendText()`-Methode bietet eine höhere Leistung.

Im folgenden Codebeispiel wird der Operator `+=` verwendet. Die Ausführung der Schleife dauert 1120 ms:

```
addChild ( myTextField );

myTextField.autoSize = TextFieldAutoSize.LEFT;
var started:Number = getTimer();

for (var i:int = 0; i< 1500; i++ )
{
    myTextField.text += "ActionScript 3";
}

trace( getTimer() - started );
// output : 1120
```

Im folgenden Beispiel wird der Operator `+=` durch die `appendText()`-Methode ersetzt:

```
var myTextField:TextField = new TextField();
addChild ( myTextField );
myTextField.autoSize = TextFieldAutoSize.LEFT;

var started:Number = getTimer();

for (var i:int = 0; i< 1500; i++ )
{
    myTextField.appendText ( "ActionScript 3" );
}

trace( getTimer() - started );
// output : 847
```

Die Ausführung des Codes dauert nun nur 847 ms.

Aktualisieren Sie Textfelder außerhalb von Schleifen, sofern möglich.

Dieser Code kann mithilfe einer einfachen Technik noch weiter optimiert werden. Das Aktualisieren des Textfeldes in jeder einzelnen Schleife erfordert sehr viel interne Verarbeitungsleistung. Die Codeausführung kann wesentlich beschleunigt werden, wenn ein String verkettet und dem Textfeld außerhalb der Schleife zugewiesen wird. Der Code wird nun in nur 2 ms ausgeführt.

```
var myTextField:TextField = new TextField();
addChild ( myTextField );
myTextField.autoSize = TextFieldAutoSize.LEFT;

var started:Number = getTimer();
var content:String = myTextField.text;

for (var i:int = 0; i< 1500; i++ )
{
    content += "ActionScript 3";
}

myTextField.text = content;

trace( getTimer() - started );
// output : 2
```

Bei der Arbeit mit HTML-Text ist die zuerst genannte Vorgehensweise so langsam, dass in Flash Player manchmal eine `Timeout`-Ausnahme ausgelöst wird. Beispielsweise kann eine Ausnahme auftreten, wenn die verwendete Hardware zu langsam ist.

Hinweis: *In Adobe® AIR® wird diese Ausnahme nicht ausgelöst.*

```
var myTextField:TextField = new TextField();
addChild ( myTextField );
myTextField.autoSize = TextFieldAutoSize.LEFT;

var started:Number = getTimer();

for (var i:int = 0; i< 1500; i++ )
{
    myTextField.htmlText += "ActionScript <b>2</b>";
}

trace( getTimer() - started );
```

Wenn der Wert einem String außerhalb der Schleife zugewiesen wird, dauert die Ausführung des Codes nur 29 ms.

```
var myTextField:TextField = new TextField();
addChild ( myTextField );
myTextField.autoSize = TextFieldAutoSize.LEFT;

var started:Number = getTimer();
var content:String = myTextField.htmlText;

for (var i:int = 0; i< 1500; i++ )
{
    content += "<b>ActionScript<b> 3";
}

myTextField.htmlText = content;

trace ( getTimer() - started );
// output : 29
```

Hinweis: *In Flash Player 10.1 und AIR 2.5 wurde die String-Klasse verbessert, sodass Strings weniger Speicher benötigen.*

Vermeiden Sie eckige Klammern als Operatoren, sofern möglich.

Die Verwendung von eckigen Klammern als Operatoren kann sich negativ auf die Leistung auswirken. Sie können diese Operatoren vermeiden, indem Sie Verweise in lokale Variablen einfügen. Das folgende Codebeispiel veranschaulicht eine ineffiziente Verwendung von eckigen Klammern als Operatoren:

```
var lng:int = 5000;
var arraySprite:Vector.<Sprite> = new Vector.<Sprite>(lng, true);
var i:int;

for ( i = 0; i< lng; i++ )
{
    arraySprite[i] = new Sprite();
}

var started:Number = getTimer();

for ( i = 0; i< lng; i++ )
{
    arraySprite[i].x = Math.random()*stage.stageWidth;
    arraySprite[i].y = Math.random()*stage.stageHeight;
    arraySprite[i].alpha = Math.random();
    arraySprite[i].rotation = Math.random()*360;
}

trace( getTimer() - started );
// output : 16
```

In der folgenden optimierten Version werden weniger eckige Klammern verwendet:

```
var lng:int = 5000;
var arraySprite:Vector.<Sprite> = new Vector.<Sprite>(lng, true);
var i:int;

for ( i = 0; i< lng; i++ )
{
    arraySprite[i] = new Sprite();
}

var started:Number = getTimer();
var currentSprite:Sprite;

for ( i = 0; i< lng; i++ )
{
    currentSprite = arraySprite[i];
    currentSprite.x = Math.random()*stage.stageWidth;
    currentSprite.y = Math.random()*stage.stageHeight;
    currentSprite.alpha = Math.random();
    currentSprite.rotation = Math.random()*360;
}

trace( getTimer() - started );
// output : 9
```


Verwenden Sie nach Möglichkeit Inline-Code, um die Anzahl der Funktionsaufrufe im Code zu verringern.

Der Aufruf von Funktionen kann viel Arbeitsleistung in Anspruch nehmen. Versuchen Sie, die Anzahl der Funktionsaufrufe zu verringern, indem Sie Code inline platzieren. Die Inline-Platzierung von Code eignet sich gut, wenn die Leistungsoptimierung im Vordergrund steht. Beachten Sie jedoch, dass Inline-Code die Wiederverwendung von Code erschweren und zu einer größeren SWF-Datei führen kann. Einige Funktionsaufrufe, wie beispielsweise die Methoden der Math-Klasse, können problemlos inline platziert werden. Im folgenden Code wird die `Math.abs()`-Methode verwendet, um absolute Werte zu berechnen:

```
const MAX_NUM:int = 500000;
var arrayValues:Vector.<Number>=new Vector.<Number>(MAX_NUM,true);
var i:int;

for (i = 0; i< MAX_NUM; i++)
{
    arrayValues[i] = Math.random()-Math.random();
}

var started:Number = getTimer();
var currentValue:Number;

for (i = 0; i< MAX_NUM; i++)
{
    currentValue = arrayValues[i];
    arrayValues[i] = Math.abs ( currentValue );
}

trace( getTimer() - started );
// output : 70
```

Die von `Math.abs()` durchgeführte Berechnung kann manuell erfolgen und inline platziert werden:

```
const MAX_NUM:int = 500000;
var arrayValues:Vector.<Number>=new Vector.<Number>(MAX_NUM,true);
var i:int;

for (i = 0; i< MAX_NUM; i++)
{
    arrayValues[i] = Math.random()-Math.random();
}

var started:Number = getTimer();
var currentValue:Number;

for (i = 0; i< MAX_NUM; i++)
{
    currentValue = arrayValues[i];
    arrayValues[i] = currentValue > 0 ? currentValue : -currentValue;
}

trace( getTimer() - started );
// output : 15
```

Durch die Inline-Platzierung des Funktionsaufrufs kann der Code viermal so schnell ausgeführt werden. Diese Strategie bietet zahlreiche Vorteile, sie kann sich jedoch negativ auf die Wiederverwendbarkeit und Pflegefreundlichkeit des Codes auswirken.

Hinweis: *Die Codegröße beeinflusst die allgemeine Player-Ausführung ganz erheblich. Wenn die Anwendung große Mengen an ActionScript-Code enthält, verbringt die Virtual Machine sehr viel Zeit mit der Überprüfung von Code und der JIT-Kompilierung. Das Nachschlagen von Eigenschaften kann länger dauern, da tiefe Vererbungshierarchien vorliegen und weil die internen Cache-Speicher häufiger ausgelastet sind. Um die Codegröße zu verringern, verzichten Sie nach Möglichkeit auf das Adobe® Flex® Framework, die TLF-Framework-Bibliothek und auf große ActionScript-Bibliotheken von Drittanbietern.*


Vermeiden Sie die Auswertung von Anweisungen in Schleifen.

Eine weitere Optimierung lässt sich erzielen, wenn eine Anweisung nicht in einer Schleife ausgewertet wird. Der folgende Code wird in Iterationen über ein Array ausgeführt, er ist aber nicht optimiert, da die Array-Länge für jede Iteration ausgewertet wird:

```
for (var i:int = 0; i< myArray.length; i++)
{
}
```

Stattdessen sollte der Wert gespeichert und wieder verwendet werden:

```
var lng:int = myArray.length;

for (var i:int = 0; i< lng; i++)
{
}
```


Verwenden Sie die umgekehrte Reihenfolge für while-Schleifen.

Eine while-Schleife in umgekehrter Reihenfolge ist schneller als eine Schleife in Vorwärtsrichtung:

```
var i:int = myArray.length;

while (--i > -1)
{
}
```

Diese Tipps helfen Ihnen bei der ActionScript-Optimierung. Sie zeigen, wie eine einzelne Codezeile Auswirkungen auf Leistung und Arbeitsspeicher haben kann. ActionScript lässt sich mit zahlreichen anderen Techniken optimieren. Weitere Informationen finden Sie unter dem folgenden Link: http://www.rozengain.com/blog/2007/05/01/some-actionscript-30-optimizations/.

Kapitel 5: Renderleistung

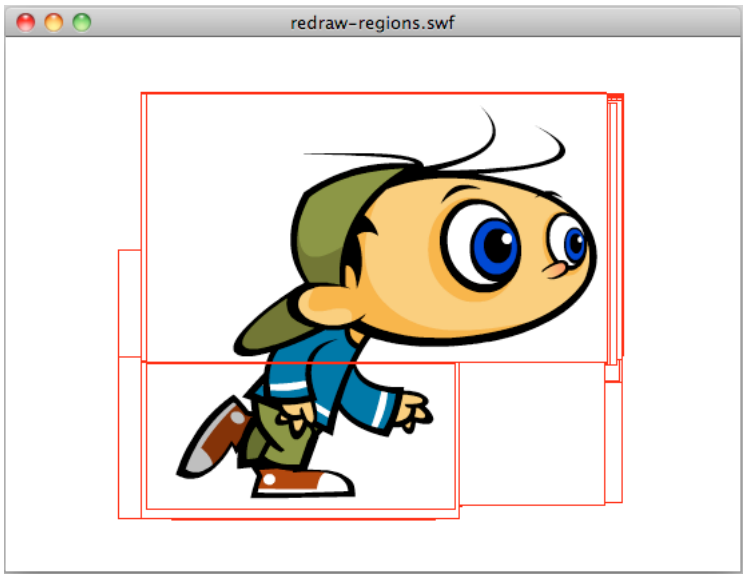
Bildaktualisierungsbereiche

Verwenden Sie beim Erstellen Ihres Projekts immer die Option für Bildaktualisierungsbereiche.

Zum Verbessern des Renderns ist es wichtig, beim Erstellen von Projekten immer die Option für Bildaktualisierungsbereiche zu verwenden. Mithilfe dieser Option können Sie die Bereiche sehen, die Flash Player rendert und verarbeitet. Sie können diese Option aktivieren, indem Sie im Kontextmenü der Debugversion von Flash Player „Bildaktualisierungsbereiche anzeigen“ wählen.

Hinweis: *Die Option „Bildaktualisierungsbereiche anzeigen“ ist in Adobe AIR oder in der Veröffentlichungsversion von Flash Player nicht verfügbar. (In Adobe AIR ist das Kontextmenü nur in Desktopanwendungen verfügbar, es enthält jedoch keine integrierten oder standardmäßigen Einträge wie „Bildaktualisierungsbereiche anzeigen“.)*

In der folgenden Abbildung wird die aktivierte Option anhand eines einfachen animierten Movieclips in der Zeitleiste veranschaulicht:



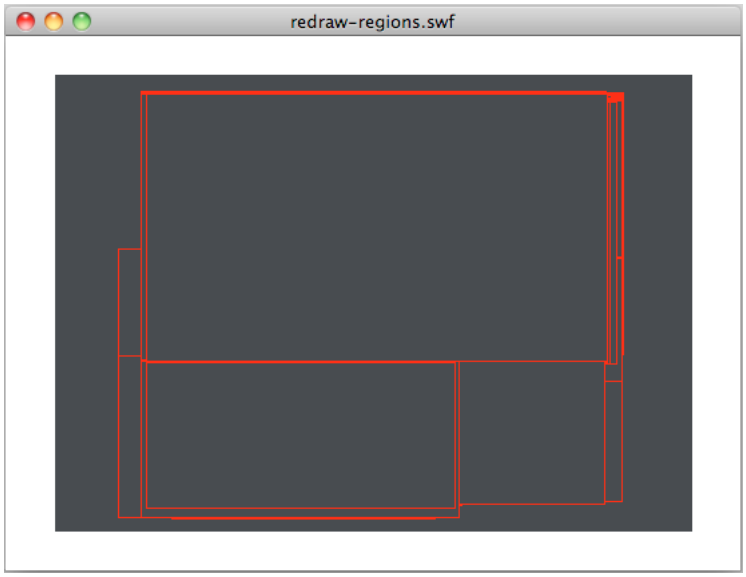
Option „Bildaktualisierungsbereiche anzeigen“ aktiviert

Sie können diese Option auch durch Programmierung aktivieren, indem Sie die `flash.profiler.showRedrawRegions()`-Methode verwenden:

```
// Enable Show Redraw Regions
// Blue color is used to show redrawn regions
flash.profiler.showRedrawRegions ( true, 0x0000FF );
```

In Adobe AIR-Anwendungen ist diese Methode die einzige Möglichkeit, die Option „Bildaktualisierungsbereiche anzeigen“ zu aktivieren.

Verwenden Sie diese Option, um Möglichkeiten der Optimierung zu identifizieren. Wie bereits erwähnt, werden einige Anzeigeobjekte auch dann gerendert, wenn sie nicht angezeigt werden, und belegen deshalb CPU-Ressourcen. Dieses Konzept wird in der folgenden Abbildung veranschaulicht. Eine schwarze Vektorform verdeckt die animierte Figur. Die Abbildung zeigt, dass das Anzeigeobjekt nicht aus der Anzeigeliste entfernt wurde und immer noch gerendert wird. Dadurch werden unnötigerweise CPU-Ressourcen beansprucht:



Bildaktualisierungsbereiche

Um die Leistung zu verbessern, stellen Sie die `visible`-Eigenschaft der verborgenen animierten Figur auf `false` ein, oder nehmen Sie die Figur ganz aus der Anzeigeliste. Sie sollten auch ihre Zeitleiste beenden. So wird gewährleistet, dass das Anzeigeobjekt fixiert wird und nur eine minimale CPU-Leistung erfordert.

Verwenden Sie die Option „Bildaktualisierungsbereiche anzeigen“ während des ganzen Entwicklungszyklus. Mit dieser Option erleben Sie am Ende des Projekts keine unangenehmen Überraschungen, wenn Sie unnötige Bildaktualisierungsbereiche und verpasste Optimierungsbereiche entdecken.

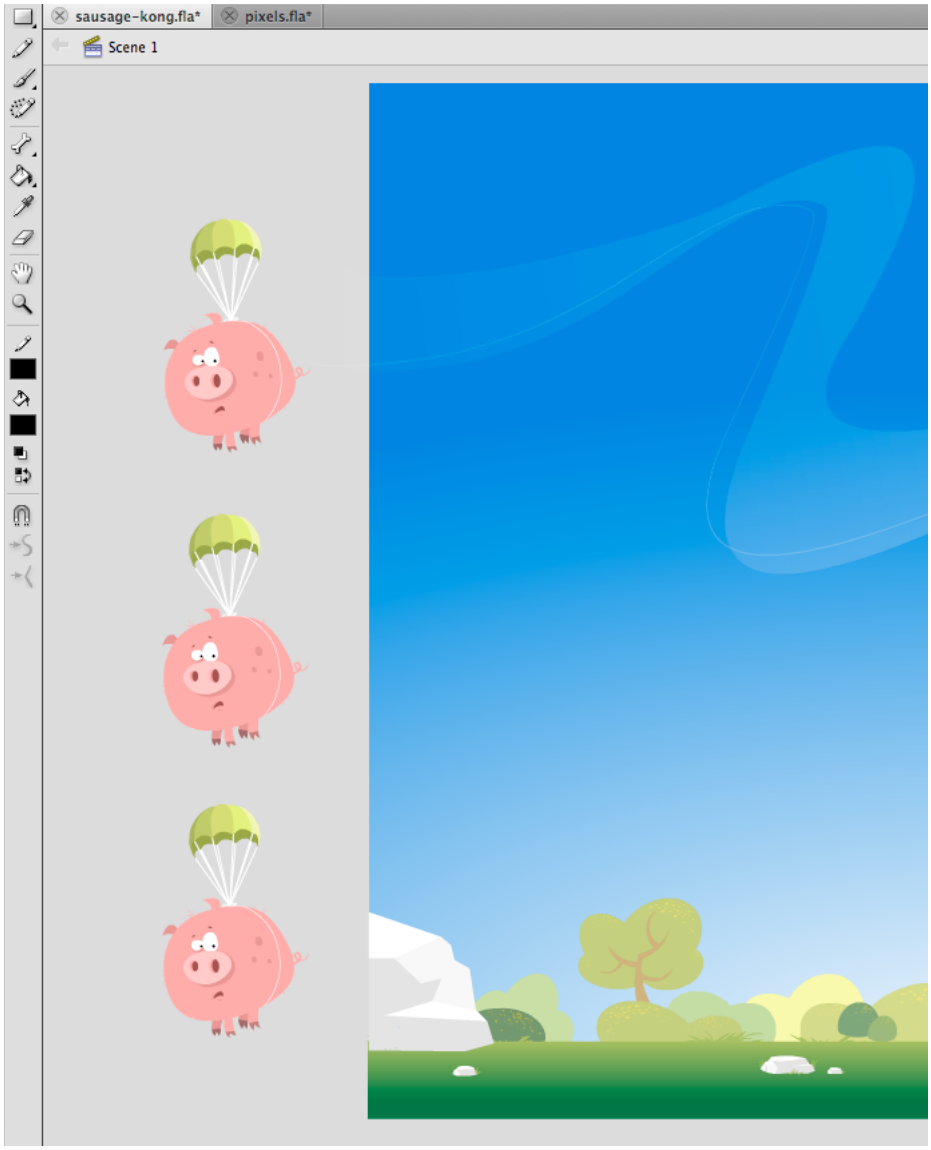
Verwandte Themen

„Fixieren und Freigeben von Objekten“ auf Seite 29

Inhalt außerhalb der Bühne

Vermeiden Sie es, Inhalte außerhalb der Bühne zu platzieren. Setzen Sie Objekte vielmehr nur dann auf die Anzeigeliste, wenn sie gebraucht werden.

Platzieren Sie nach Möglichkeit keine grafischen Inhalte außerhalb der Bühne. Designer und Entwickler platzieren Elemente im Allgemeinen außerhalb der Bühne, um sie im Verlauf der Anwendung wiederzuverwenden. Die folgende Abbildung veranschaulicht diese gebräuchliche Vorgehensweise:



Inhalt außerhalb der Bühne

Auch wenn die Elemente außerhalb der Bühne nicht auf dem Bildschirm zu sehen sind und nicht gerendert werden, sind sie in der Anzeigeliste noch vorhanden. Die Laufzeitumgebung führt weiterhin interne Tests für diese Elemente aus, um sicherzustellen, dass sie sich immer noch außerhalb der Bühne befinden und der Benutzer nicht mit ihnen interagiert. Deshalb sollten Sie so weit wie möglich vermeiden, Objekte außerhalb der Bühne zu platzieren und sie stattdessen besser aus der Anzeigeliste entfernen.

Filmqualität

Verwenden Sie die geeignete Einstellung für die Bühnenqualität, um das Rendern zu verbessern.

Bei der Entwicklung von Inhalt für Mobilgeräte mit kleinen Bildschirmen, wie Telefone, spielt die Bildqualität eine weniger wichtige Rolle als bei Desktopanwendungen. Über die richtige Einstellung der Bühnenqualität kann die Renderleistung verbessert werden.

Für die Bühnenqualität stehen die folgenden Einstellungen zur Verfügung:

- `StageQuality.LOW`: Die Wiedergabegeschwindigkeit hat Vorrang vor der Darstellungsqualität; Anti-Aliasing wird nicht verwendet. Diese Einstellung wird in Adobe AIR für Desktop oder TV nicht unterstützt.
- `StageQuality.MEDIUM`: Das Anti-Aliasing wird in eingeschränktem Maße angewendet, skalierte Bitmaps werden jedoch nicht geglättet. Diese Einstellung ist der Standardwert für AIR auf mobilen Geräten, wird in AIR für Desktop oder TV jedoch nicht unterstützt.
- `StageQuality.HIGH`: (Standardeinstellung auf dem Desktop) Die Darstellungsqualität hat Vorrang vor der Wiedergabegeschwindigkeit; Anti-Aliasing wird grundsätzlich verwendet. Wenn die SWF-Datei keine Animationen enthält, werden skalierte Bitmaps geglättet; andernfalls werden Bitmaps nicht geglättet.
- `StageQuality.BEST`: Bietet die beste Anzeigequalität ohne Berücksichtigung der Wiedergabegeschwindigkeit. Für die gesamte Ausgabe erfolgt ein Anti-Aliasing und skalierte Bitmaps werden immer geglättet.

Die Einstellung `StageQuality.MEDIUM` bietet häufig eine ausreichende Qualität für Anwendungen auf Mobilgeräten. In einigen Fällen kann auch mit der Einstellung `StageQuality.LOW` eine angemessene Qualität erzielt werden. Ab Flash Player 8 kann Text, auf den Anti-Aliasing angewendet wurde, präzise gerendert werden, selbst wenn die Bühnenqualität auf `LOW` eingestellt ist.

Hinweis: *Bei einigen Mobilgeräten wird, auch wenn die Qualität auf `HIGH` gesetzt ist, `MEDIUM` für eine bessere Leistung in Flash Player-Anwendungen verwendet. Die Qualitätseinstellung HIGH führt jedoch häufig nicht zu einem merkbaren Unterschied, da die Bildschirme von Mobilgeräten meist einen höheren DPI-Wert aufweisen. (Der DPI-Wert kann von Gerät zu Gerät unterschiedlich sein.)*

Für die folgende Abbildung ist die Filmqualität auf `MEDIUM` eingestellt und das Rendern von Text auf „Anti-Aliasing für Animation“:

Mittlere Bühnenqualität und „Anti-Aliasing für Animation“ für das Rendern von Text

Die Bühnenqualität wirkt sich auf die Textqualität aus, da die entsprechende Einstellung für das Rendern von Text nicht verwendet wird.

Die Laufzeitumgebung ermöglicht Ihnen, „Anti-Aliasing für Lesbarkeit“ für das Rendern von Text zu verwenden. Bei dieser Einstellung wird stets eine perfekte Qualität des Textes (mit Anti-Aliasing) beibehalten, unabhängig davon, welche Bühnenqualität verwendet wird:

Niedrigere Bühnenqualität und „Anti-Aliasing für Lesbarkeit“ für das Rendern von Text

Dieselbe Renderqualität wird erzielt, indem das Rendern von Text auf „Bitmaptext (kein Anti-Alias)“ eingestellt wird:

Niedrigere Bühnenqualität und „Bitmaptext (kein Anti-Alias)“ für das Rendern von Text

Die beiden letzten Beispiele zeigen, dass Text mit hoher Qualität unabhängig von der Einstellung für die Bühnenqualität erreicht werden kann. Dieses Funktionsmerkmal ist seit Flash Player 8 verfügbar und kann auf Mobilgeräten eingesetzt werden. Beachten Sie, dass Flash Player 10.1 auf manchen Geräten automatisch zur Einstellung `StageQuality.MEDIUM` wechselt, um die Leistung zu verbessern.

Alpha-Mischung

Die Alpha-Eigenschaft sollte nach Möglichkeit nicht verwendet werden.

Vermeiden Sie Effekte, die eine Alpha-Mischung erfordern, wenn die `alpha`-Eigenschaft verwendet wird, wie beispielsweise Ein- und Ausblendeffekte. Wenn ein Anzeigeobjekt eine Alpha-Mischung verwendet, muss die Laufzeitumgebung die Farbwerte jedes gestapelten Anzeigeobjekts und die Hintergrundfarbe kombinieren, um die endgültige Farbe zu bestimmen. Deshalb kann eine Alpha-Mischung rechenintensiver sein als das Zeichnen einer undurchsichtigen Farbe. Diese zusätzliche Rechenzeit kann die Leistung bei langsameren Geräten beeinträchtigen. Die `alpha`-Eigenschaft sollte nach Möglichkeit nicht verwendet werden.

Verwandte Themen

„Bitmap-Zwischenspeicherung“ auf Seite 55

„Rendern von Textobjekten“ auf Seite 69

Bildrate der Anwendung

Im Allgemeinen sollte die verwendete Bildrate so niedrig wie möglich sein, um eine bessere Leistung zu erzielen..

Die Bildrate der Anwendung bestimmt, wie viel Zeit für jeden Zyklus aus Anwendungscode und Rendering verfügbar ist, wie unter „Grundlagen der Laufzeitcodeausführung“ auf Seite 1 beschrieben. Je höher die Bildrate, desto gleichmäßiger die Animation. Wenn jedoch keine Animationen oder andere visuellen Änderungen auftreten, ist eine hohe Bildrate häufig unnötig. Eine höhere Bildrate benötigt mehr CPU-Zyklen und damit Akkuenergie als eine niedrigere Bildrate.

Nachstehend finden Sie einige allgemeine Richtlinien für eine angemessene Standardbildrate in Ihrer Anwendung:

- Wenn Sie das Flex-Framework verwenden, lassen Sie die anfängliche Bildrate auf den Standardwert eingestellt..
- Wenn Ihre Anwendung Animationen enthält, sollte die Bildrate mindestens 20 Bilder pro Sekunde (fps) betragen. Eine höhere Bildrate als 30 Bilder pro Sekunde ist häufig nicht nötig.
- Enthält Ihre Anwendung keine Animation, ist eine Bildrate von 12 Bildern pro Sekunde wahrscheinlich ausreichend.

Die „niedrigstmögliche Bildrate“ kann je nach der aktuellen Aktivität der Anwendung variieren. Weitere Informationen finden Sie unter dem nächsten Tipp, „Ändern Sie die Bildrate in der Anwendung dynamisch“.

Verwenden Sie eine niedrige Bildrate, wenn Video der einzige dynamische Inhalt in Ihrer Anwendung ist.

Die Laufzeitumgebung spielt Videoinhalte mit ihrer nativen Bildrate ab, unabhängig von der Bildrate der Anwendung. Wenn Ihre Anwendung keine Animationen oder andere sich schnell ändernden grafischen Inhalte enthält, verschlechtert eine niedrige Bildrate das Benutzererlebnis nicht.

Ändern Sie die Bildrate in der Anwendung dynamisch.

Sie definieren die anfängliche Bildrate der Anwendung in den Projekt- oder Compilereinstellungen, die Bildrate ist jedoch nicht auf diesen Wert festgelegt. Sie können die Bildrate ändern, indem Sie die `Stage.frameRate`-Eigenschaft (oder die `WindowedApplication.frameRate`-Eigenschaft in Flex) einstellen.

Ändern Sie die Bildrate je nach den aktuellen Anforderungen der Anwendung. Verringern Sie die Bildrate zum Beispiel, wenn die Anwendung keine Animation ausführt. Erhöhen Sie die Bildrate kurz vor Beginn des animierten Übergangs. Wenn die Anwendung im Hintergrund ausgeführt wird (also nicht mehr den Fokus hat), kann die Bildrate normalerweise noch weiter verringert werden. Der Benutzer konzentriert sich wahrscheinlich auf eine andere Anwendung oder Aufgabe.

Die folgenden allgemeinen Richtlinien können Sie als Ausgangspunkt für das Bestimmen der angemessenen Bildrate für verschiedene Aktivitäten verwenden:

- Wenn Sie das Flex-Framework verwenden, lassen Sie die anfängliche Bildrate auf den Standardwert eingestellt..
- Wenn eine Animation abgespielt wird, sollte die Bildrate mindestens 20 Bilder pro Sekunde betragen. Eine höhere Framerate als 30 Bilder pro Sekunde ist häufig nicht nötig.

- Wenn keine Animation abgespielt wird, ist eine Bildrate von 12 Bildern pro Sekunde wahrscheinlich ausreichend.
- Geladenes Video wird unabhängig von der Bildrate der Anwendung mit seiner nativen Bildrate geladen. Wenn Video der einzige bewegte Inhalt in Ihrer Anwendung ist, ist eine Bildrate von 12 Bildern pro Sekunde wahrscheinlich ausreichend.
- Wenn die Anwendung nicht den Eingabefokus hat, ist eine Bildrate von 5 Bildern pro Sekunde vermutlich ausreichend.
- Wenn eine AIR-Anwendung nicht sichtbar ist, eignet sich normalerweise eine Bildrate von 2 fps oder weniger. Diese Richtlinie gilt zum Beispiel, wenn eine Anwendung minimiert wird. Sie gilt auch für Desktopgeräte, wenn die `visible`-Eigenschaft des nativen Fensters den Wert `false` hat.

Für in Flex erstellte Anwendungen verfügt die spark.components. WindowedApplication-Klasse über integrierte Unterstützung für das dynamische Ändern der Anwendungsbildrate. Die `backgroundFrameRate`-Eigenschaft definiert die Bildrate der Anwendung, wenn die Anwendung nicht aktiv ist. Der Standardwert 1 ändert die Bildrate einer Anwendung, die mit der Spark-Architektur erstellt wurde, in 1 fps. Sie können die Bildrate für den Hintergrund über die `backgroundFrameRate`-Eigenschaft ändern. Sie können die Eigenschaft auf einen anderen Wert einstellen. Wenn Sie die automatische Verringerung der Bildrate deaktivieren möchten, wählen Sie den Wert -1.

Weitere Informationen zum dynamischen Ändern der Anwendungsbildrate finden Sie in den folgenden Artikeln:

- Reducing CPU usage in Adobe AIR (Adobe Developer Center-Artikel und Beispielcode von Jonnie Hallman)
- Writing well-behaved, efficient, AIR applications (Artikel und Beispielanwendung von Arno Gourdol)

Grant Skinner hat eine Klasse zum Verringern der Bildrate entwickelt. Mithilfe dieser Klasse können Sie die Bildrate in einer Anwendung automatisch reduzieren, wenn die Anwendung im Hintergrund ausgeführt wird. Weitere Informationen und eine Downloadmöglichkeit für den Quellcode der FramerateThrottler-Klasse finden Sie in Grant Skinners Artikel „Idle CPU Usage in Adobe AIR and Flash Player“ unter http://gskinner.com/blog/archives/2009/05/idle_cpu_usage.html.

Adaptive Bildrate

Beim Kompilieren einer SWF-Datei legen Sie eine bestimmte Bildrate für den Film fest. In Umgebungen mit eingeschränkten Ressourcen und einer langsamen CPU kommt es manchmal vor, dass die Bildrate sich während der Wiedergabe verringert. Um auch in einer solchen Situation eine akzeptable Bildrate beizubehalten, lässt die Laufzeitumgebung einige Bilder beim Rendern aus. Indem einige Bilder nicht gerendert werden, wird verhindert, dass die Bildrate unter einen bestimmten noch akzeptablen Wert fällt.

Hinweis: *In diesem Fall überspringt die Laufzeitumgebung keine Bilder, sondern sie überspringt das Rendern des Inhalts dieser Bilder. Der Code wird weiterhin ausgeführt und die Anzeigeliste wird aktualisiert, aber die Aktualisierungen werden auf dem Bildschirm nicht angezeigt. Es gibt keine Möglichkeit, einen fps-Wert für die Anzahl der zu überspringenden Bilder anzugeben, wenn die Laufzeitumgebung keine stabile Bildrate beibehalten kann.*

Bitmap-Zwischenspeicherung

Verwenden Sie das Zwischenspeichern von Bitmaps für komplexen Vektorinhalt, sofern geeignet.

Eine gute Optimierung lässt sich durch das Zwischenspeichern von Bitmaps erzielen. Dabei wird ein Vektorobjekt im Cache zwischengespeichert und intern als Bitmap gerendert. Diese Bitmap wird dann zum Rendern verwendet. Die Renderleistung kann auf diese Weise ganz erheblich verbessert werden, der Nachteil ist jedoch, dass der Arbeitsspeicher stark beansprucht werden kann. Verwenden Sie das Zwischenspeichern von Bitmaps für komplexen Vektorinhalt, wie beispielsweise komplexe Verläufe oder Text.

Das Aktivieren der Bitmap-Zwischenspeicherung für ein animiertes Objekt mit komplexen Vektorgrafiken (wie Text oder Farbverläufe) verbessert die Leistung. Wenn die Bitmap-Zwischenspeicherung jedoch für ein Anzeigeobjekt aktiviert ist, dessen Zeitleiste abgespielt wird (wie bei einem Movieclip), tritt das Gegenteil ein. In diesem Fall muss die Laufzeitumgebung die zwischengespeicherte Bitmap bei jedem Bild aktualisieren und auf dem Bildschirm neu zeichnen, was die CPU stark beansprucht. Die Bitmap-Zwischenspeicherung bietet nur dann Vorteile, wenn die zwischengespeicherte Bitmap nur einmal erstellt und dann nicht mehr aktualisiert werden muss.

Wenn Sie die Bitmap-Zwischenspeicherung für ein Sprite-Objekt aktivieren, kann das Objekt verschoben werden, ohne dass die Laufzeitumgebung die zwischengespeicherte Bitmap neu generiert. Durch Ändern der Eigenschaften `x` und `y` wird das Objekt nicht neu generiert. Wenn jedoch versucht wird, das Objekt zu drehen, zu skalieren oder seinen Alphawert zu ändern, generiert die Laufzeitumgebung die zwischengespeicherte Bitmap neu, was sich negativ auf die Leistung auswirkt.

Hinweis: *Die `DisplayObject.cacheAsBitmapMatrix`-Eigenschaft, die in AIR und im Packager for iPhone verfügbar ist, hat diese Einschränkung nicht. Indem Sie die `cacheAsBitmapMatrix`-Eigenschaft verwenden, können Sie den Alphawert eines Objekts drehen, skalieren, neigen und ändern, ohne dass die Bitmap neu generiert wird.*

Eine zwischengespeicherte Bitmap belegt möglicherweise mehr Arbeitsspeicher als eine reguläre Movieclip-Instanz. Beispielsweise belegt ein Movieclip, der auf der Bühne eine Größe von 250 x 250 Pixel hat, im Cache ca. 250 KB anstelle von regulär 1 KB.

Das folgende Beispiel betrifft ein Sprite-Objekt, das ein Bild eines Apfels enthält. Die folgende Klasse wird an das Symbol des Apfels angefügt:

```
package org.bytearray.bitmap
{
    import flash.display.Sprite;
    import flash.events.Event;

    public class Apple extends Sprite
    {
        private var destinationX:Number;
        private var destinationY:Number;

        public function Apple ()
        {
            addEventListener(Event.ADDED_TO_STAGE,activation);
            addEventListener(Event.REMOVED_FROM_STAGE,deactivation);
        }

        private function activation(e:Event):void
        {
            initPos();
            addEventListener (Event.ENTER_FRAME,handleMovement);
        }

        private function deactivation(e:Event):void
```

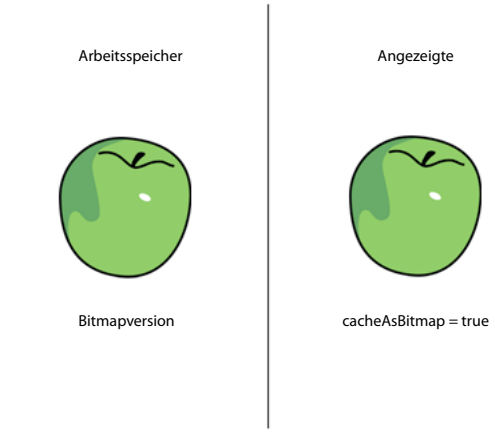
```
        {
            removeEventListener(Event.ENTER_FRAME,handleMovement);
        }

        private function initPos():void
        {
            destinationX = Math.random()*(stage.stageWidth - (width>>1));
            destinationY = Math.random()*(stage.stageHeight - (height>>1));
        }

        private function handleMovement(e:Event):void
        {
            x -= (x - destinationX)*.5;
            y -= (y - destinationY)*.5;

            if (Math.abs(x - destinationX) < 1 && Math.abs(y - destinationY) < 1)
                initPos();
        }
    }
}
```

In diesem Code wird die Sprite-Klasse anstelle der MovieClip-Klasse verwendet, da keine separate Zeitleiste für jeden Apfel erforderlich ist. Verwenden Sie das Objekt, das am wenigsten Ressourcen belegt, um eine optimale Leistung zu erreichen. Als Nächstes wird die Klasse mit dem folgenden Code instanziiert:

```
import org.bytearray.bitmap.Apple;

stage.addEventListener(MouseEvent.CLICK,createApples);
stage.addEventListener(KeyboardEvent.KEY_DOWN,cacheApples);

const MAX_NUM:int = 100;
var apple:Apple;
var holder:Sprite = new Sprite();

addChild(holder);

function createApples(e:MouseEvent):void
{
    for (var i:int = 0; i< MAX_NUM; i++)
    {
        apple = new Apple();

        holder.addChild(apple);
    }
}

function cacheApples(e:KeyboardEvent):void
{
    if (e.keyCode == 67)
    {
        var lng:int = holder.numChildren;

        for (var i:int = 0; i < lng; i++)
        {
            apple = holder.getChildAt (i) as Apple;

            apple.cacheAsBitmap = Boolean(!apple.cacheAsBitmap);
        }
    }
}
```

Wenn der Benutzer mit der Maus klickt, werden die Äpfel ohne Zwischenspeicherung erstellt. Wenn der Benutzer die Taste „C“ drückt (Tastencode 67), werden die Vektoren der Äpfel als Bitmaps im Cache zwischengespeichert und auf dem Bildschirm angezeigt. Mit dieser Technik kann die Renderleistung bei einer langsamen CPU sowohl auf Desktop- als auch auf Mobilgeräten deutlich verbessert werden.

Dabei ist jedoch zu beachten, dass das Zwischenspeichern von Bitmaps zwar die Renderleistung verbessert, andererseits aber große Mengen Arbeitsspeicher in Anspruch nehmen kann. Sobald ein Objekt im Cache zwischengespeichert wird, wird seine Oberfläche als transparente Bitmap erfasst und im Arbeitsspeicher gespeichert, wie in der folgenden Abbildung gezeigt:

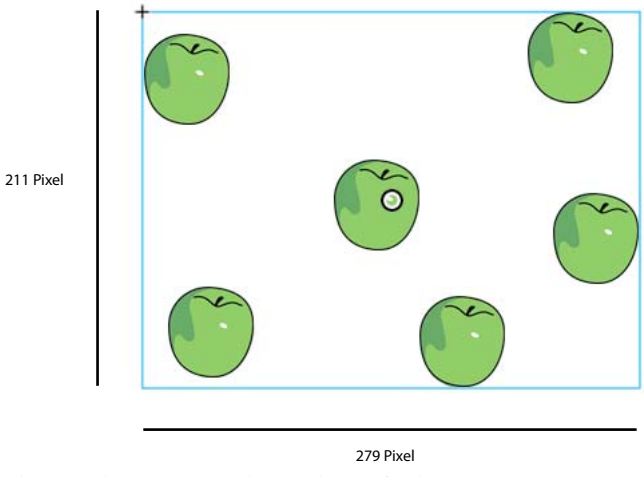


Objekt und seine Oberflächen-Bitmap im Arbeitsspeicher

Flash Player 10.1 und AIR 2.5 optimieren die Nutzung des Arbeitsspeichers mit demselben Ansatz, der unter „Filter und dynamisches Entladen von Bitmaps“ auf Seite 20 beschrieben ist. Wenn ein zwischengespeichertes Anzeigeobjekt ausgeblendet oder offscreen ist, wird der von der Bitmap belegte Arbeitsspeicher freigegeben, nachdem die Bitmap eine Weile nicht verwendet wurde.

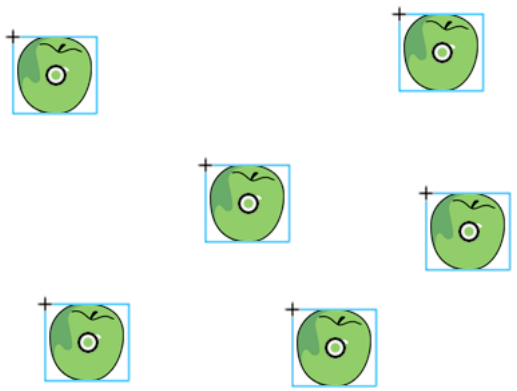
Hinweis: *Wenn die* `opaqueBackground`*-Eigenschaft des Anzeigeobjekts auf eine bestimmte Farbe eingestellt ist, geht die Laufzeitumgebung davon aus, dass das Anzeigeobjekt undurchsichtig ist. Bei Verwendung mit der* `cacheAsBitmap`*-Eigenschaft erstellt die Laufzeitumgebung eine undurchsichtige 32-Bit-Bitmap im Arbeitsspeicher. Der Alphakanal ist auf 0xFF eingestellt, was die Leistung verbessert, da keine Transparenz erforderlich ist, um die Bitmap auf dem Bildschirm zu zeichnen. Durch die Vermeidung der Alpha-Mischung wird das Rendern noch schneller. Wenn die aktuelle Bildschirmtiefe auf 16 Bit beschränkt ist, wird die Bitmap als 16-Bit-Bild im Arbeitsspeicher gespeichert. Durch Verwendung der* `opaqueBackground`*-Eigenschaft wird die Bitmap-Zwischenspeicherung nicht implizit aktiviert.*

Um Arbeitsspeicher einzusparen, empfiehlt es sich, die `cacheAsBitmap`-Eigenschaft zu verwenden und für jedes Anzeigeobjekt einzeln anstatt für den Container zu aktivieren. Wenn die Bitmap-Zwischenspeicherung für den Container aktiviert wird, ist die endgültige Bitmap im Arbeitsspeicher wesentlich größer, da eine transparente Bitmap mit den Abmessungen 211 x 279 Pixel erstellt wird. Das Bild belegt ungefähr 229 KB Arbeitsspeicher.



Aktivieren der Bitmap-Zwischenspeicherung für den Container

Außerdem birgt das Zwischenspeichern des Containers das Risiko, dass die ganze Bitmap im Arbeitsspeicher aktualisiert wird, wenn ein Apfel sich in einem Bild bewegt. Wenn die Bitmap-Zwischenspeicherung für die einzelnen Instanzen aktiviert wird, werden sechs Oberflächen mit jeweils 7 KB im Arbeitsspeicher zwischengespeichert, also insgesamt nur 42 KB.



Aktivieren der Bitmap-Zwischenspeicherung für Instanzen

Wenn Sie auf jede Apfel-Instanz über die Anzeigeliste zugreifen und die `getChildAt()`-Methode aufrufen, werden Verweise in einem Vector-Objekt gespeichert, was den Zugriff vereinfacht:

```
import org.bytearray.bitmap.Apple;

stage.addEventListener(KeyboardEvent.KEY_DOWN, cacheApples);

const MAX_NUM:int = 200;
var apple:Apple;
var holder:Sprite = new Sprite();

addChild(holder);

var holderVector:Vector.<Apple> = new Vector.<Apple>(MAX_NUM, true);

for (var i:int = 0; i< MAX_NUM; i++)
{
    apple = new Apple();

    holder.addChild(apple);

    holderVector[i] = apple;
}

function cacheApples(e:KeyboardEvent):void
{
    if (e.keyCode == 67)
    {
        var lng:int = holderVector.length

        for (var i:int = 0; i < lng; i++)
        {
            apple = holderVector[i];

            apple.cacheAsBitmap = Boolean(!apple.cacheAsBitmap);
        }
    }
}
```

Beachten Sie, dass die Bitmap-Zwischenspeicherung das Rendern verbessert, wenn der zwischengespeicherte Inhalt nicht in jedem Bild gedreht, skaliert oder geändert wird. Bei anderen Transformationen als der Verschiebung entlang der x- und y-Achsen wird das Rendern jedoch nicht verbessert. In diesen Fällen aktualisiert Flash Player die zwischengespeicherte Kopie der Bitmap für jede Transformation des Anzeigeobjekts. Die Aktualisierung der zwischengespeicherten Kopie kann zu einer hohen CPU-Auslastung, einer schlechteren Leistung und zu einer hohen Akkunutzung führen. Wiederum gilt diese Einschränkung nicht für die `cacheAsBitmapMatrix`-Eigenschaft, die in AIR und im Packager for iPhone zur Verfügung steht.

Der folgende Code ändert den Alphawert in der Verschiebungsmethode, wodurch die Deckkraft des Apfels in jedem Bild geändert wird:

```
private function handleMovement(e:Event):void
{
        alpha = Math.random();
        x -= (x - destinationX)*.5;
        y -= (y - destinationY)*.5;

        if (Math.abs(x - destinationX) < 1 && Math.abs(y - destinationY) < 1)
            initPos();
}
```

Die Verwendung der Bitmap-Zwischenspeicherung führt zu Leistungseinbußen. Wegen des geänderten Alphawertes muss die Laufzeitumgebung die im Arbeitsspeicher zwischengespeicherte Bitmap bei jeder Änderung des Alphawertes aktualisieren.

Filter benötigen Bitmaps, die immer aktualisiert werden, wenn der Abspielkopf eines zwischengespeicherten Movieclips sich bewegt. Bei Verwendung eines Filters wird die `cacheAsBitmap`-Eigenschaft deshalb automatisch auf `true` gesetzt. Die folgende Abbildung zeigt einen animierten Movieclip:

Animierter Movieclip

Vermeiden Sie Filter für animierte Inhalte, da dies zu Leistungsproblemen führen kann. In der folgenden Abbildung wurde ein Schlagschatten-Filter hinzugefügt:

Animierter Movieclip mit Schlagschatten-Filter

Deshalb muss die Bitmap neu generiert werden, wenn die Zeitleiste innerhalb des Movieclips abgespielt wird. Die Bitmap muss auch neu generiert werden, wenn der Inhalt auf eine Weise geändert wird, bei der es sich nicht um eine einfache x- oder y-Transformation handelt. Bei jedem Bild muss die Laufzeitumgebung die Bitmap neu zeichnen. Dies erfordert mehr CPU-Ressourcen, beeinträchtigt die Leistung und beansprucht den Akku stärker.

Paul Trani stellt in den folgenden Schulungsvideos Beispiele für die Verwendung von Flash Professional und ActionScript zur Optimierung von Grafiken mit Bitmaps bereit:

- Optimizing Graphics
- Optimizing Graphics with ActionScript

Transformationsmatrizes zwischengespeicherter Bitmaps in AIR

Legen Sie die `cacheAsBitmapMatrix`-Eigenschaft fest, wenn Sie zwischengespeicherte Bitmaps in mobilen AIR-Anwendungen verwenden.

Im Mobil-AIR-Profil können Sie der `cacheAsBitmapMatrix`-Eigenschaft eines Anzeigeobjekts ein Matrix-Objekt zuweisen. Wenn Sie diese Eigenschaft festlegen, können Sie eine beliebige zweidimensionale Transformation auf das Objekt anwenden, ohne dass die zwischengespeicherte Bitmap neu generiert werden muss. Sie können auch die alpha-Eigenschaft ändern, ohne eine Neugenerierung der zwischengespeicherten Bitmap auszulösen. Die `cacheAsBitmap`-Eigenschaft muss ebenfalls den Wert `true` haben, und für das Objekt dürfen keine 3D-Eigenschaften festgelegt sein.

Das Festlegen der `cacheAsBitmapMatrix`-Eigenschaft generiert die zwischengespeicherte Bitmap, selbst wenn sich das Anzeigeobjekt außerhalb des Bildschirms befindet, verdeckt wird oder eine `visible`-Eigenschaft mit dem Wert `false` aufweist. Das Zurücksetzen der `cacheAsBitmapMatrix`-Eigenschaft durch ein Matrix-Objekt, das eine andere Transformation enthält, generiert die zwischengespeicherte Bitmap ebenfalls neu.

Die Matrix-Transformation, die Sie auf die `cacheAsBitmapMatrix`-Eigenschaft anwenden, wird auf das Anzeigeobjekt angewendet, wenn es in den Bitpmapcache gerendert wird. Somit ist das Bitmaprendering doppelt so groß wie das Vektorrendering, wenn die Transformation eine Zweifachskalierung enthält. Der Renderer wendet diese umgekehrte Transformation auf die zwischengespeicherte Bitmap an, sodass die endgültige Anzeige genauso aussieht. Sie können die zwischengespeicherte Bitmap auf eine kleinere Größe skalieren, um weniger Arbeitsspeicher zu benötigen; dies geht vermutlich auf Kosten der Renderinggenauigkeit. Unter Umständen möchten Sie die Bitmap auch größer skalieren, um die Renderingqualität zu verbessern, was dann mehr Arbeitsspeicher benötigt. Im Allgemeinen sollten Sie jedoch eine Identitätsmatrix verwenden, also eine Matrix, die keine Transformationen anwendet, um Änderungen am Erscheinungsbild zu vermeiden, wie im folgenden Beispiel dargestellt:

```
displayObject.cacheAsBitMap = true;
displayObject.cacheAsBitmapMatrix = new Matrix();
```

Nachdem Sie die `cacheAsBitmapMatrix`-Eigenschaft festgelegt haben, können Sie das Objekt skalieren, neigen, drehen und versetzen, ohne die erneute Generierung der Bitmap auszulösen.

Sie können auch den Alphawert im Bereich 0 bis 1 ändern. Wenn Sie den Alphawert über die `transform.colorTransform`-Eigenschaft mit einer Farbtransformation ändern, muss der im Transformationsobjekt verwendete Alphawert zwischen 0 und 255 liegen. Wenn Sie die Farbtransformation auf andere Weise ändern, wird die zwischengespeicherte Bitmap neu generiert.

Legen Sie immer die `cacheAsBitmapMatrix`-Eigenschaft fest, wenn Sie `cacheAsBitmap` in Inhalten, der für Mobilgeräte erstellt wird, auf `true` einstellen. Bedenken Sie jedoch die folgende mögliche Beeinträchtigung. Nachdem ein Objekt gedreht, skaliert oder geneigt wurde, können anders als beim normalen Vektorrendering beim endgültigen Rendering Bitmapskalierungs- oder Aliasing-Artefakte auftreten.

Manuelle Bitmap-Zwischenspeicherung

Verwenden Sie die BitmapData-Klasse, um ein benutzerdefiniertes Verhalten für die Bitmap-Zwischenspeicherung zu erstellen.

Im folgenden Beispiel wird eine einzelne gerasterte Bitmapversion eines Anzeigeobjekts wiederverwendet, wobei auf dasselbe BitmapData-Objekt verwiesen wird. Beim Skalieren der einzelnen Anzeigeobjekte wird das ursprüngliche BitmapData-Objekt weder im Arbeitsspeicher aktualisiert noch neu gezeichnet. Mit dieser Technik können Sie CPU-Ressourcen einsparen und die Anwendungsausführung beschleunigen. Beim Skalieren des Anzeigeobjekts wird die enthaltene Bitmap gestreckt.

Dies ist die aktualisierte `BitmapApple`-Klasse:

```
package org.bytearray.bitmap
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.events.Event;

    public class BitmapApple extends Bitmap
    {
        private var destinationX:Number;
        private var destinationY:Number;

        public function BitmapApple(buffer:BitmapData)
        {
            super(buffer);

            addEventListener(Event.ADDED_TO_STAGE,activation);
            addEventListener(Event.REMOVED_FROM_STAGE,deactivation);
        }

        private function activation(e:Event):void
        {
            initPos();
            addEventListener(Event.ENTER_FRAME,handleMovement);
        }

        private function deactivation(e:Event):void
        {
            removeEventListener(Event.ENTER_FRAME,handleMovement);
        }

        private function initPos():void
        {
            destinationX = Math.random()*(stage.stageWidth - (width>>1));
            destinationY = Math.random()*(stage.stageHeight - (height>>1));
        }

        private function handleMovement(e:Event):void
        {
            alpha = Math.random();

            x -= (x - destinationX)*.5;
            y -= (y - destinationY)*.5;

            if ( Math.abs(x - destinationX) < 1 && Math.abs(y - destinationY) < 1)
                initPos();
        }
    }
}
```

Der Alphawert wird immer noch in jedem Bild geändert. Der folgende Code übergibt den ursprünglichen Quellpuffer an jede BitmapApple-Instanz:

```
import org.bytearray.bitmap.BitmapApple;

const MAX_NUM:int = 100;
var holder:Sprite = new Sprite();

addChild(holder);

var holderVector:Vector.<BitmapApple> = new Vector.<BitmapApple>(MAX_NUM, true);
var source:AppleSource = new AppleSource();
var bounds:Object = source.getBounds(source);

var mat:Matrix = new Matrix();
mat.translate(-bounds.x,-bounds.y);

var buffer:BitmapData = new BitmapData(source.width+1, source.height+1, true, 0);
buffer.draw(source,mat);

var bitmapApple:BitmapApple;

for (var i:int = 0; i< MAX_NUM; i++)
{
    bitmapApple = new BitmapApple(buffer);

    holderVector[i] = bitmapApple;

    holder.addChild(bitmapApple);
}
```

Für diese Technik ist nur wenig Arbeitsspeicher erforderlich, da nur eine zwischengespeicherte Bitmap im Arbeitsspeicher verwendet und von allen `BitmapApple`-Instanzen gemeinsam genutzt wird. Außerdem wird die ursprüngliche Quellbitmap nie aktualisiert, unabhängig von Änderungen, die an den `BitmapApple`-Instanzen vorgenommen werden (wie beispielsweise Alphawert, Drehungen oder Skalierungen). Diese Technik verhindert einen Leistungsabfall.

Damit die endgültige Bitmap glatt aussieht, setzen Sie die `smoothing`-Eigenschaft auf `true`:

```
public function BitmapApple(buffer:BitmapData)
{
    super (buffer);

    smoothing = true;

    addEventListener(Event.ADDED_TO_STAGE, activation);
    addEventListener(Event.REMOVED_FROM_STAGE, deactivation);
}
```

Auch eine Anpassung der Bühnenqualität kann die Leistung verbessern. Setzen Sie die Bühnenqualität vor dem Rastern auf `HIGH` und danach auf `LOW`:

```
import org.bytearray.bitmap.BitmapApple;

const MAX_NUM:int = 100;
var holder:Sprite = new Sprite();

addChild ( holder );

var holderVector:Vector.<BitmapApple> = new Vector.<BitmapApple>(MAX_NUM, true);
var source:AppleSource = new AppleSource();
var bounds:Object = source.getBounds ( source );

var mat:Matrix = new Matrix();
mat.translate ( -bounds.x, -bounds.y );

var buffer:BitmapData = new BitmapData ( source.width+1, source.height+1, true, 0 );

stage.quality = StageQuality.HIGH;

buffer.draw ( source, mat );

stage.quality = StageQuality.LOW;

var bitmapApple:BitmapApple;

for (var i:int = 0; i< MAX_NUM; i++ )
{
    bitmapApple = new BitmapApple( buffer );

    holderVector[i] = bitmapApple;

    holder.addChild ( bitmapApple );
}
```

Das Wechseln der Bühnenqualität vor und nach dem Zeichnen eines Vektors als Bitmap kann eine äußerst leistungsstarke Strategie sein, um Anti-Aliasing-Text auf dem Bildschirm anzuzeigen. Diese Technik kann unabhängig von der endgültigen Bühnenqualität gute Ergebnisse liefern. Beispielsweise lässt sich eine Anti-Aliasing-Bitmap mit Anti-Aliasing-Text erzielen, selbst wenn die Bühnenqualität auf `LOW` eingestellt ist. Diese Technik kann mit der `cacheAsBitmap`-Eigenschaft nicht eingesetzt werden. In diesem Fall bewirkt die Einstellung `LOW` für die Filmqualität, dass die Vektorqualität aktualisiert wird. Dadurch werden die Bitmapoberflächen im Arbeitsspeicher und dann auch die endgültige Qualität aktualisiert.

Isolieren von Verhalten

Ereignisse wie `Event.ENTER_FRAME` sollten nach Möglichkeit in einer einzelnen Prozedur isoliert werden.

Sie können den Code noch weiter optimieren, indem Sie das `Event.ENTER_FRAME`-Ereignis in der `Apple`-Klasse in einer einzelnen Prozedur isolieren. Durch diese Technik werden CPU-Ressourcen eingespart. Das folgende Beispiel zeigt dieses Verfahren, bei dem die BitmapApple-Klasse nicht mehr für das Verhalten der Bewegung zuständig ist:

```
package org.bytearray.bitmap
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;

    public class BitmapApple extends Bitmap
    {
        private var destinationX:Number;
        private var destinationY:Number;

        public function BitmapApple(buffer:BitmapData)
        {
            super (buffer);

            smoothing = true;
        }
}
```

Der folgende Code instanziiert die Äpfel und verarbeitet ihre Bewegung in einer einzelnen Prozedur:

```
import org.bytearray.bitmap.BitmapApple;

const MAX_NUM:int = 100;
var holder:Sprite = new Sprite();

addChild(holder);

var holderVector:Vector.<BitmapApple> = new Vector.<BitmapApple>(MAX_NUM, true);
var source:AppleSource = new AppleSource();
var bounds:Object = source.getBounds(source);

var mat:Matrix = new Matrix();
mat.translate(-bounds.x,-bounds.y);

stage.quality = StageQuality.BEST;

var buffer:BitmapData = new BitmapData(source.width+1,source.height+1, true,0);
buffer.draw(source,mat);

stage.quality = StageQuality.LOW;

var bitmapApple:BitmapApple;

for (var i:int = 0; i< MAX_NUM; i++)
{
    bitmapApple = new BitmapApple(buffer);

    bitmapApple.destinationX = Math.random()*stage.stageWidth;
    bitmapApple.destinationY = Math.random()*stage.stageHeight;

    holderVector[i] = bitmapApple;

    holder.addChild(bitmapApple);
}

stage.addEventListener(Event.ENTER_FRAME,onFrame);
```

```
var lng:int = holderVector.length

function onFrame(e:Event):void
{
        for (var i:int = 0; i < lng; i++)
        {
            bitmapApple = holderVector[i];
            bitmapApple.alpha = Math.random();

            bitmapApple.x -= (bitmapApple.x - bitmapApple.destinationX) *.5;
            bitmapApple.y -= (bitmapApple.y - bitmapApple.destinationY) *.5;

            if (Math.abs(bitmapApple.x - bitmapApple.destinationX ) < 1 &&
                Math.abs(bitmapApple.y - bitmapApple.destinationY ) < 1)
            {
                    bitmapApple.destinationX = Math.random()*stage.stageWidth;
                    bitmapApple.destinationY = Math.random()*stage.stageHeight;
            }
        }
}
```

Daraus ergibt sich ein einzelnes `Event.ENTER_FRAME`-Ereignis zur Verarbeitung der Bewegung anstelle von 200 Prozeduren zur Bewegung der einzelnen Äpfel. Die ganze Animation kann problemlos angehalten werden, was besonders in einem Spiel nützlich sein kann.

In einem einfachen Spiel kann beispielsweise die folgende Prozedur zum Einsatz kommen:

```
stage.addEventListener(Event.ENTER_FRAME, updateGame);
function updateGame (e:Event):void
{
    gameEngine.update();
}
```

Im nächsten Schritt muss die Interaktion der Äpfel mit der Maus oder der Tastatur implementiert werden. Dazu sind Änderungen an der `BitmapApple`-Klasse erforderlich.

```
package org.bytearray.bitmap
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.Sprite;

    public class BitmapApple extends Sprite
    {
        public var destinationX:Number;
        public var destinationY:Number;
        private var container:Sprite;
        private var containerBitmap:Bitmap;

        public function BitmapApple(buffer:BitmapData)
        {
            container = new Sprite();
            containerBitmap = new Bitmap(buffer);
            containerBitmap.smoothing = true;
            container.addChild(containerBitmap);
            addChild(container);
        }
}
```

Dadurch entstehen interaktive `BitmapApple`-Instanzen, ähnlich wie herkömmliche Sprite-Objekte. Die Instanzen sind jedoch mit einer einzelnen Bitmap verknüpft, die nicht neu berechnet wird, wenn die Anzeigeobjekte transformiert werden.

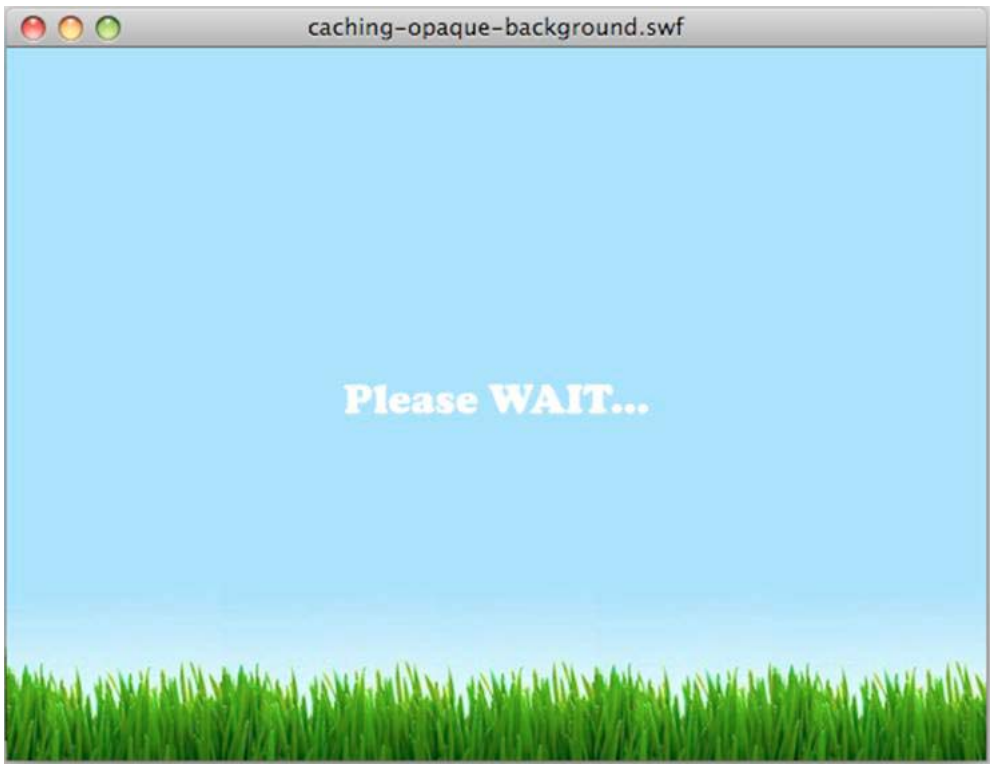
Rendern von Textobjekten

Verwenden Sie die Bitmap-Zwischenspeicherung und die `opaqueBackground`-Eigenschaft, um die Leistung beim Rendern von Text zu verbessern.

Die Flash Text Engine ermöglicht einige sehr gute Optimierungen. Es sind jedoch zahlreiche Klassen erforderlich, um eine einzelne Textzeile anzuzeigen. Deshalb erfordert die Erstellung eines bearbeitbaren Textfeldes mit der TextLine-Klasse relativ viel Arbeitsspeicher und zahlreiche Zeilen mit ActionScript-Code. Die TextLine-Klasse eignet sich am besten für statischen und nicht bearbeitbaren Text, da sie hier ein schnelleres Rendern ermöglicht und weniger Arbeitsspeicher beansprucht.

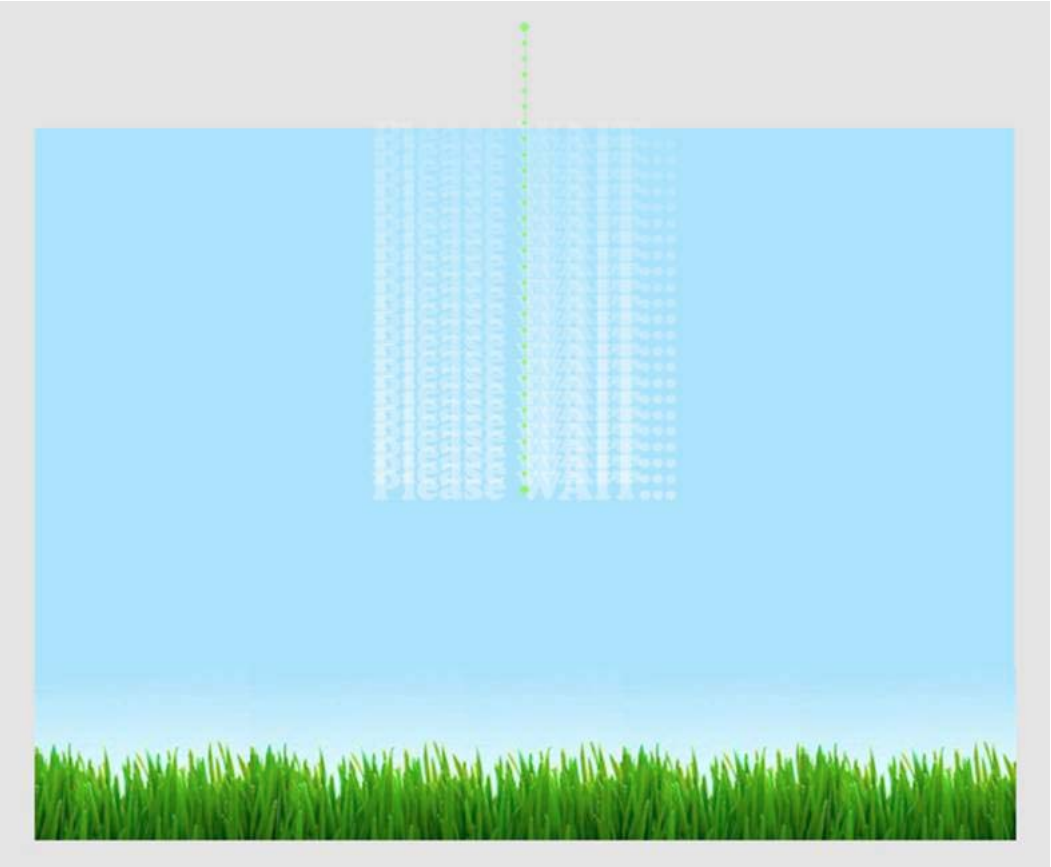
Mit der Bitmap-Zwischenspeicherung können Sie Vektorinhalt in Form von Bitmaps im Cache zwischenspeichern, um die Renderleistung zu verbessern. Dieses Funktionsmerkmal eignet sich für komplexe Vektorinhalte und für die Verwendung von Textinhalt, der beim Rendern Verarbeitungsleistung erfordert.

Das folgende Beispiel zeigt, wie Sie mit der Bitmap-Zwischenspeicherung und der `opaqueBackground`-Eigenschaft die Renderleistung verbessern können. Die folgende Abbildung zeigt einen typischen Willkommensbildschirm, der beispielsweise angezeigt wird, während der Benutzer darauf wartet, dass Objekte geladen werden:



Willkommensbildschirm

Die folgende Abbildung zeigt die Beschleunigung, die programmatisch auf das TextField-Objekt angewendet wird. Der Text wird langsam vom oberen Szenenrand bis zur Mitte der Szene beschleunigt.



Textbeschleunigung

Mit dem folgenden Code wird die Beschleunigung erstellt. Die `preloader`-Variable enthält das aktuelle Zielobjekt, damit Eigenschaften weniger häufig nachgeschlagen werden müssen, was zu Leistungseinbußen führen könnte:

```
wait_mc.addEventListener( Event.ENTER_FRAME, movePosition );

var destX:Number=stage.stageWidth/2;
var destY:Number=stage.stageHeight/2;
var preloader:DisplayObject;

function movePosition( e:Event ):void
{
    preloader = e.currentTarget as DisplayObject;

    preloader.x -= ( preloader.x - destX ) * .1;
    preloader.y -= ( preloader.y - destY ) * .1;

    if (Math.abs(preloader.y-destY)<1)
        preloader.removeEventListener( Event.ENTER_FRAME, movePosition );
}
```

Die `Math.abs()`-Funktion könnte hier inline platziert werden, um die Anzahl der Funktionsaufrufe zu verringern und weitere Leistungsvorteile zu erzielen. Es empfiehlt sich, den int-Datentyp für die Eigenschaften `destX` und `destY` zu verwenden, damit Festkommawerte entstehen. Durch Verwendung des int-Datentyps erzielen Sie eine perfekte Pixelausrichtung, ohne dass Werte mithilfe von langsamen Methoden wie `Math.ceil()` oder `Math.round()` manuell gerundet werden müssen. Dieser Code rundet die Koordinaten nicht in den int-Datentyp, da durch eine kontinuierliche Rundung der Werte keine gleichmäßige Bewegung des Objekts entsteht. Das Objekt kann sich ruckartig bewegen, da die Koordinaten in jedem Bild an die nächsten gerundeten Ganzzahlen ausgerichtet werden. Diese Technik kann jedoch nützlich sein, um die endgültige Position eines Anzeigeobjekts festzulegen. Der folgende Code sollte nicht verwendet werden:

```
// Do not use this code
var destX:Number = Math.round ( stage.stageWidth / 2 );
var destY:Number = Math.round ( stage.stageHeight / 2);
```

Der folgende Code ist wesentlich schneller:

```
var destX:int = stage.stageWidth / 2;
var destY:int = stage.stageHeight / 2;
```

Der vorherige Code kann noch weiter optimiert werden, indem Operatoren zur bitweisen Verschiebung verwendet werden, um die Werte zu teilen:

```
var destX:int = stage.stageWidth >> 1;
var destY:int = stage.stageHeight >> 1;
```

Dank der Bitmap-Zwischenspeicherung kann die Laufzeitumgebung Objekte mithilfe von dynamischen Bitmaps einfacher rendern. Im aktuellen Beispiel wird der Movieclip, der das TextField-Objekt enthält, im Cache zwischengespeichert.

```
wait_mc.cacheAsBitmap = true;
```

Eine weitere Leistungssteigerung lässt sich erzielen, indem die Alphatransparenz entfernt wird. Alphatransparenz belastet die Laufzeitumgebung zusätzlich, wenn transparente Bitmapbilder gezeichnet werden wie im vorherigen Codebeispiel. Sie können dies vermeiden, indem Sie die `opaqueBackground`-Eigenschaft verwenden und eine Hintergrundfarbe angeben.

Bei Verwendung der `opaqueBackground`-Eigenschaft belegt die im Arbeitsspeicher erstellte Bitmapoberfläche immer noch 32 Bit. Der Alphaversatz ist jedoch auf 255 eingestellt und es wird keine Transparenz verwendet. Deshalb verringert die `opaqueBackground`-Eigenschaft zwar nicht die Arbeitsspeicherbelastung, aber sie verbessert die Renderleistung, wenn die Bitmap-Zwischenspeicherung verwendet wird. Der folgende Code enthält alle Optimierungen:

```
wait_mc.addEventListener( Event.ENTER_FRAME, movePosition );
wait_mc.cacheAsBitmap = true;

// Set the background to the color of the scene background
wait_mc.opaqueBackground = 0x8AD6FD;
var destX:int = stage.stageWidth >> 1;
var destY:int = stage.stageHeight >> 1;
var preloader:DisplayObject;

function movePosition ( e:Event ):void
{
    preloader = e.currentTarget as DisplayObject;

    preloader.x -= ( preloader.x - destX ) * .1;
    preloader.y -= ( preloader.y - destY ) * .1;

    if ( Math.abs ( preloader.y - destY ) < 1 )
        e.currentTarget.removeEventListener ( Event.ENTER_FRAME, movePosition );
}
```

Die Animation ist nun optimiert. Die Bitmap-Zwischenspeicherung wurde durch Entfernen der Transparenz optimiert. Bei Verwendung der Bitmap-Zwischenspeicherung empfiehlt es sich auf Mobilgeräten möglicherweise, die Bühnenqualität je nach Animationsstatus zwischen `LOW` und `HIGH` zu wechseln:

```
wait_mc.addEventListener( Event.ENTER_FRAME, movePosition );
wait_mc.cacheAsBitmap = true;
wait_mc.opaqueBackground = 0x8AD6FD;

// Switch to low quality
stage.quality = StageQuality.LOW;
var destX:int = stage.stageWidth>>1;
var destY:int = stage.stageHeight>>1;
var preloader:DisplayObject;

function movePosition( e:Event ):void
{
    preloader = e.currentTarget as DisplayObject;

    preloader.x -= ( preloader.x - destX ) * .1;
    preloader.y -= ( preloader.y - destY ) * .1;

    if (Math.abs(e.currentTarget.y-destY)<1)
    {
        // Switch back to high quality
        stage.quality = StageQuality.HIGH;
        preloader.removeEventListener( Event.ENTER_FRAME, movePosition );
    }
}
```

In diesem Fall bewirkt eine Änderung der Bühnenqualität jedoch, dass die Laufzeitumgebung die Bitmapoberfläche des TextField-Objekts neu generieren muss, um sie an die aktuelle Bühnenqualität anzupassen. Deshalb sollte die Bühnenqualität bei Verwendung der Bitmap-Zwischenspeicherung nicht geändert werden.

Sie hätten hier auch ein manuelles Verfahren zur Bitmap-Zwischenspeicherung verwenden können. Zur Simulation der `opaqueBackground`-Eigenschaft kann der Movieclip als nicht transparentes BitmapData-Objekt gezeichnet werden; bei diesem Verfahren muss die Laufzeitumgebung die Bitmapoberfläche nicht neu generieren.

Dieses Verfahren eignet sich gut für Inhalt, der sich im Laufe der Zeit nicht ändert. Wenn der Inhalt des Textfeldes sich jedoch ändern kann, sollten Sie eine andere Strategie in Betracht ziehen. Nehmen wir als Beispiel ein Textfeld, das kontinuierlich mit einem Prozentsatz aktualisiert wird, um zu zeigen, wie weit die Anwendung bereits geladen wurde. Wenn das Textfeld oder das zugehörige Anzeigeobjekt als Bitmap zwischengespeichert wurde, muss seine Oberfläche bei jeder Änderung des Inhalts neu generiert werden. Die manuelle Bitmap-Zwischenspeicherung kann hier nicht verwendet werden, da der Inhalt des Anzeigeobjekts sich fortlaufend ändert. Wegen dieser konstanten Änderung müssten Sie die `BitmapData.draw()`-Methode manuell aufrufen, um die zwischengespeicherte Bitmap zu aktualisieren.

Denken Sie daran, dass seit Flash Player 8 (und AIR 1.0) unabhängig von der Qualitätseinstellung der Bühne für ein Textfeld mit der Einstellung „Anti-Aliasing für Lesbarkeit“ ein vollkommenes Anti-Aliasing erfolgt. Dieses Verfahren belegt etwas weniger Arbeitsspeicher, erfordert aber mehr CPU-Leistung und etwas mehr Zeit zum Rendern als die Bitmap-Zwischenspeicherung.

Diese Technik wird im folgenden Code verwendet:

```
wait_mc.addEventListener( Event.ENTER_FRAME, movePosition );

// Switch to low quality
stage.quality = StageQuality.LOW;
var destX:int = stage.stageWidth >> 1;
var destY:int = stage.stageHeight >> 1;
var preloader:DisplayObject;
function movePosition ( e:Event ):void
{
    preloader = e.currentTarget as DisplayObject;

    preloader.x -= ( preloader.x - destX ) * .1;
    preloader.y -= ( preloader.y - destY ) * .1;

    if ( Math.abs ( preloader.y - destY ) < 1 )
    {
        // Switch back to high quality
        stage.quality = StageQuality.HIGH;
        preloader.removeEventListener ( Event.ENTER_FRAME, movePosition );
    }
}
```

Die Verwendung dieser Option („Anti-Aliasing für Lesbarkeit“) für beweglichen Text wird nicht empfohlen. Beim Skalieren von Text wird durch diese Option versucht, die Ausrichtung des Textes beizubehalten, wodurch ein Verschiebungseffekt entsteht. Wenn der Inhalt des Anzeigeobjekts sich ständig ändert und skalierter Text erforderlich ist, können Sie die Leistung bei Mobilgeräten jedoch verbessern, indem Sie die Qualität auf `LOW` einstellen. Nach beendeter Bewegung wechseln Sie wieder zur Qualitätseinstellung `HIGH`.

GPU

GPU-Rendering in Flash Player-Anwendungen

Ein wichtiges neues Funktionsmerkmal in Flash Player 10.1 ist die Möglichkeit, die GPU zum Rendern von grafischen Inhalten auf Mobilgeräten zu verwenden. Bisher wurden Grafiken ausschließlich über die CPU gerendert. Durch Verwendung der GPU wird das Rendern von Filtern, Bitmaps, Video und Text optimiert. Dabei sollten Sie jedoch beachten, dass das Rendern per GPU nicht immer genauso präzise ist wie das Rendern über die Software. Bei Verwendung des Hardware-Rendermoduls kann Inhalt etwas grob aussehen. Außerdem gilt in Flash Player 10.1 eine Einschränkung, die das Rendern von Pixel Bender-Effekten auf dem Bildschirm verhindern kann. Diese Effekte werden möglicherweise als schwarzes Quadrat dargestellt, wenn die Hardwarebeschleunigung verwendet wird.

In Flash Player 10 stand zwar eine GPU-Beschleunigung zur Verfügung, die GPU wurde jedoch nicht zur Berechnung der Grafiken verwendet, sondern nur zum Senden aller Grafiken an den Bildschirm. In Flash Player 10.1 wird die GPU auch zum Berechnen der Grafiken verwendet, wodurch die Geschwindigkeit beim Rendern erheblich verbessert wird. Außerdem verringert dieses Verfahren die CPU-Auslastung und bietet damit einen Vorteil für Geräte mit eingeschränkten Ressourcen, wie beispielsweise Mobilgeräte.

Der GPU-Modus wird beim Ausführen von Inhalt auf Mobilgeräten automatisch eingestellt, um eine optimale Leistung zu gewährleisten. Es ist nicht mehr erforderlich, `wmode` auf `gpu` einzustellen, um das GPU-Rendern festzulegen; doch die GPU-Beschleunigung wird deaktiviert, wenn `wmode` auf `opaque` oder `transparent` eingestellt wird.

Hinweis: *Flash Player für Desktopanwendungen setzt weiterhin die CPU für das Software-Rendern ein. Das Software-Rendern wird verwendet, da bei den Treibern auf dem Desktop große Unterschiede bestehen. Dies kann bewirken, dass Unterschiede beim Rendern besonders deutlich ausfallen. Es können auch Unterschiede beim Rendern zwischen dem Desktop und einigen Mobilgeräten auftreten.*

GPU-Rendering in mobilen AIR-Anwendungen

Aktivieren Sie die Hardwaregrafikbeschleunigung in AIR-Anwendungen, in dem Sie `<renderMode>gpu</renderMode>` in den Anwendungsdeskriptor aufnehmen. Sie können den Rendermodus zur Laufzeit nicht ändern. Auf Desktopcomputern wird die Einstellung für `renderMode` ignoriert; die GPU-Grafikbeschleunigung wird zurzeit nicht unterstützt.

Einschränkungen beim GPU-Rendering

Die folgenden Einschränkungen gelten, wenn der GPU-Rendermodus in AIR 2.5 verwendet wird:

- Wenn die GPU ein Objekt nicht rendern kann, wird es überhaupt nicht angezeigt. Das CPU-Rendering wird nicht als Fallback verwendet.
- Die folgenden Mischmodi werden nicht unterstützt: Ebene, Alpha, Löschen, Überlagern, Hartes Licht, Aufhellen und Abdunkeln.
- Filter werden nicht unterstützt.
- PixelBender wird nicht unterstützt.
- Viele GPUs haben eine maximale Texturgröße von 1024x1024. In ActionScript bezieht sich dies auf die maximal zulässige endgültig gerenderte Größe eines Anzeigeobjekts, nachdem alle Transformationen angewendet wurden.
- Adobe empfiehlt, das GPU-Rendering für AIR-Anwendungen, die Video abspielen, nicht zu verwenden.

- Beim GPU-Rendering werden Textfelder nicht immer auf eine sichtbare Position verschoben, wenn die virtuelle Tastatur geöffnet wird. Verwenden Sie eine der folgenden Möglichkeiten, um sicherzustellen, dass Ihr Textfeld sichtbar ist, wenn der Benutzer Text eingibt. Platzieren Sie das Textfeld in der oberen Hälfte des Bildschirms oder verschieben Sie es in die obere Hälfte des Bildschirms, wenn das Textfeld den Fokus erhält.
- Das GPU-Rendering ist bei einigen Geräten, auf denen dieser Modus nicht zuverlässig funktioniert, deaktiviert. Die neuesten Informationen hierzu finden Sie in den AIR-Versionshinweisen für Entwickler.

Bewährte Verfahren für das GPU-Rendering

Die folgenden Tipps können das GPU-Rendern beschleunigen:

- Halten Sie die Anzahl der sichtbaren Elemente auf der Bühne möglichst gering. Jedes Element braucht Zeit für die Darstellung und die Zusammensetzung mit den umgebenden Objekten. Wenn ein Anzeigeobjekt nicht mehr angezeigt werden soll, stellen Sie seine `visible`-Eigenschaft auf `false` ein. Sie sollten es nicht einfach von der Bühne verschieben, hinter einem anderen Objekt verstecken oder seine `alpha`-Eigenschaft auf 0 setzen. Wenn das Anzeigeobjekt gar nicht mehr benötigt wird, entfernen Sie es mithilfe von `removeChild()` von der Bühne.
- Anstatt Objekte zu erstellen und zu löschen, sollten Sie sie wiederverwenden.
- Erstellen Sie Bitmaps mit Größen, die kleiner als 2^n mal 2^m Bit sind, aber diese Werte fast erreichen. Die Abmessungen müssen keine genauen Zweierpotenzen sein, aber sie sollten nahe an einer Zweierpotenz sein, jedoch nicht größer. Zum Beispiel wird ein Bild der Größe 31 x 15 Pixel schneller dargestellt als ein Bild der Größe 33 x 17. (31 und 15 sind knapp kleiner als die Zweierpotenzen 32 bzw. 16.)
- Setzen Sie den repeat-Parameter nach Möglichkeit auf false, wenn Sie die `Graphic.beginBitmapFill()`-Methode aufrufen.
- Weniger ist mehr. Verwenden Sie die Hintergrundfarbe als Hintergrund. Legen Sie große Formen nicht übereinander ab. Jedes einzelne Pixel, das gezeichnet werden muss, hat seinen Preis.
- Vermeiden Sie Formen mit langen, dünnen Zacken, sich selbst schneidenden Rändern oder zahlreichen feinen Details an den Rändern. Diese Formen werden langsamer dargestellt als Formen mit glatten Rändern.
- Vermeiden Sie zu große Anzeigeobjekte.
- Aktivieren Sie cacheAsBitMap und cacheAsBitmapMatrix für Anzeigeobjekte, deren Grafiken nicht oft aktualisiert werden.
- Vermeiden Sie die Verwendung der ActionScript-Zeichnungs-API (die Graphics-Klasse) beim Erstellen von Grafiken. Erstellen Sie diese Objekte nach Möglichkeit statisch beim Authoring.
- Skalieren Sie Bitmapbestände vor dem Importieren auf die endgültige Größe.

GPU-Rendering in mobilen AIR 2.0.3-Anwendungen

Das GPU-Rendering ist in mobilen AIR-Anwendungen, die mit dem Packager for iPhone erstellt werden, restriktiver. Die GPU wird nur für Bitmaps, durchgehende Formen und Anzeigeobjekte mit der `cacheAsBitmap`-Eigenschaft verwendet. Bei Objekten, für die `cacheAsBitmap` und `cacheAsBitmapMatrix` eingestellt sind, kann die GPU auch Objekte rendern, die gedreht oder skaliert werden. Die GPU wird in Reihe mit anderen Anzeigeobjekten verwendet, was im Allgemeinen zu einer schlechteren Renderleistung führt.

Tipps zum Optimieren der GPU-Renderleistung

Auch wenn GPU-Rendering die Leistung für SWF-Inhalte deutlich verbessern kann, spielt der Entwurf des Inhalts doch auch eine große Rolle. Denken Sie daran, dass Einstellungen, die in der Vergangenheit mit Softwarerendering gut funktioniert haben, mit GPU-Rendering nicht immer genauso gut funktionieren. Mit den folgenden Tipps können Sie mit GPU-Rendering eine gute Leistung erzielen, ohne dass es beim Softwarerendering zu Leistungseinbußen kommt.

Hinweis: *Mobile Geräte, die Hardwarerendering unterstützen, rufen SWF-Inhalte häufig über das Internet auf. Deshalb ist es am besten, diese Tipps beim Erstellen von SWF-Inhalten zu befolgen, um sicherzustellen, dass auf allen Bildschirmen die bestmöglichen Ergebnisse erzielt werden können.*

- Vermeiden Sie die Verwendung von `wmode=transparent` oder `wmode=opaque` in HTML-eingebetteten Parametern. Diese Modi können zu schlechteren Leistungen führen. Es kann dabei sowohl bei Software- als auch bei Hardwarerendering auch zu einem leichten Verlust bei der Audio-Video-Synchronizität kommen. Außerdem unterstützen viele Plattformen das GPU-Rendering nicht, wenn diese Modi verwendet werden, wodurch die Leistung deutlich verschlechtert wird.
- Verwenden Sie nur die normalen und Alpha-Mischmodi. Vermeiden Sie die Verwendung anderer Mischmodi, besonders des Ebenenmischmodus. Nicht alle Mischmodi können beim GPU-Rendering originalgetreu wiedergegeben werden.
- Wenn eine GPU Vektorgrafiken darstellt, werden sie in Netze aus kleinen Dreiecken zerlegt, bevor sie gezeichnet werden. Dieser Prozess wird Parkettierung genannt. Die Parkettierung ist mit kleinen Leistungseinbußen verbunden, die mit zunehmender Komplexität der Form größer werden. Um die Leistung so wenig wie möglich zu beeinträchtigen, vermeiden Sie Morph-Formen, die beim GPU-Rendering in jedem Bild parkettiert werden.
- Vermeiden Sie sich selbst schneidende Kurven, sehr dünne Kurvenformen (z. B. einen dünnen zunehmenden Mond) und komplizierte Details an den Rändern einer Form. Die Parkettierung dieser Formen in Dreiecke ist für die GPU aufwändig. Zur Veranschaulichung stellen Sie sich zwei Vektoren vor: ein Quadrat der Größe 500 × 500 und ein zunehmender Mond der Größe 100 × 10. Das große Quadrat kann von der GPU problemlos dargestellt werden, da es lediglich aus zwei Dreiecken besteht. Um die Kurve des zunehmenden Mondes zu beschreiben, sind jedoch zahlreiche Dreiecke erforderlich. Das Darstellen dieser Form ist deshalb komplizierter, obwohl weniger Pixel beteiligt sind.
- Vermeiden Sie große Änderungen beim Skalieren, da diese Änderungen ebenfalls dazu führen können, dass die GPU die Grafiken erneut parkettiert.
- Vermeiden Sie das Überzeichnen, wenn möglich. Überzeichnen ist das Überlagern mehrerer grafischer Elemente, sodass sie einander verdecken. Mit dem Softwarerenderer wird jedes Pixel nur einmal gezeichnet. Beim Softwarerendering gibt es in der Anwendung deshalb keine Leistungseinbußen, unabhängig davon, wie viele grafische Elemente einander an dieser Pixelposition überlagern. Der Hardwarerenderer dagegen zeichnet jedes Pixel für jedes Element, auch wenn dieser Bereich durch andere Elemente verdeckt wird. Wenn sich zwei Rechtecke überlappen, zeichnet der Hardwarerenderer den überlappten Bereich zweimal, während der Softwarerenderer diesen Bereich nur einmal zeichnet.

  Auf dem Desktop, wo der Softwarerenderer verwendet wird, bemerken Sie deshalb normalerweise keine Leistungsverschlechterung beim Überzeichnen. Bei Geräten, die GPU-Rendering verwenden, kann sich die Verwendung vieler überlappender Formen dagegen nachteilig auf die Leistung auswirken. Es hat sich bewährt, Objekte aus der Anzeigeliste zu entfernen anstatt sie zu verbergen.
- Verwenden Sie kein großes gefülltes Rechteck als Hintergrund. Legen Sie stattdessen die Hintergrundfarbe des Stage-Objekts fest.
- Vermeiden Sie möglichst den standardmäßigen Bitmapfüllmodus Bitmapwiederholung. Verwenden Sie stattdessen den Bitmapclampmodus, um eine bessere Leistung zu erzielen.

Asynchrone Operationen

Geben Sie der asynchronen Version von Operationen den Vorzug gegenüber synchronen Versionen, falls möglich.

Synchrone Operationen werden ausgeführt, sobald der Code dies befiehlt, und der Code wird erst dann fortgesetzt, wenn die Operationen abgeschlossen sind. Deshalb werden sie in der Anwendungscodephase der Bildschleife ausgeführt. Wenn eine synchrone Operation zu lange dauert, dehnt sie die Größe der Bildschleife aus, wodurch die Anzeige stehenbleiben („einfrieren“) oder stocken kann.

Wenn der Code eine asynchrone Operation ausführt, muss diese nicht notwendigerweise sofort ausgeführt werden. Ihr Code und weiterer Anwendungscode im aktuellen Ausführungs-Thread werden weiterhin ausgeführt. Die Laufzeit führt die Operation so bald wie möglich aus und versucht dabei, Probleme beim Rendern zu vermeiden. In einigen Fällen erfolgt die Ausführung im Hintergrund und nicht als Teil der Bildschleife der Laufzeitumgebung. Wenn die Operation abgeschlossen ist, setzt die Laufzeitumgebung ein Ereignis ab und Ihr Code kann einen Listener für dieses Ereignis verwenden, um weitere Aufgaben auszuführen.

Asynchrone Operationen sind geplant und werden aufgeteilt, um Renderingprobleme zu vermeiden. Folglich ist es mit asynchronen Operationen viel einfacher, eine reagierende Anwendung zu behalten. Weitere Informationen finden Sie unter „Wahrgenommene Leistung und tatsächliche Leistung“ auf Seite 3.

Die asynchrone Ausführung von Operationen erfordert jedoch einige Ressourcen. Die tatsächliche Ausführungszeit kann für asynchrone Operationen länger sein, besonders bei Operationen, die nur wenig Zeit erfordern.

In der Laufzeitumgebung sind viele Operationen von Natur aus synchron oder asynchron, sodass Sie nicht wählen können, wie sie ausgeführt werden. In Adobe AIR gibt es jedoch drei Arten von Operationen, für die Sie zwischen synchroner und asynchroner Ausführung wählen können:

- Operationen der File- und FileStream-Klasse

  Viele Operationen der File-Klasse können synchron oder asynchron ausgeführt werden. Die Methoden zum Kopieren oder Löschen von Dateien und Ordnern sowie zum Auflisten von Verzeichnisinhalten gibt es zum Beispiel jeweils in einer asynchronen Version. Bei diesen Methoden steht das Suffix „Async“ hinter dem Namen der asynchronen Version. Um zum Beispiel eine Datei asynchron zu löschen, rufen Sie die `File.deleteFileAsync()`-Methode anstatt der `File.deleteFile()`-Methode auf.

  Wenn Sie ein FileStream-Objekt zum Lesen aus einer Datei oder zum Schreiben in eine Datei verwenden, bestimmt die Art, wie das FileStream-Objekt geöffnet wird, ob die Operationen asynchron ausgeführt werden. Verwenden Sie die `FileStream.openAsync()`-Methode für asynchrone Operationen. Das Schreiben der Daten erfolgt asynchron. Daten werden in Blöcken geschrieben, sodass die Daten jeweils portionsweise verfügbar sind. Im Gegensatz dazu liest das FileStream-Objekt im synchronen Modus die gesamte Datei, bevor die Codeausführung fortgesetzt wird.

- Lokale SQL-Datenbankoperationen

  Beim Arbeiten mit einer lokalen SQL-Datenbank werden alle Operationen, die durch ein SQLConnection-Objekt ausgeführt werden, entweder im synchronen oder asynchronen Modus ausgeführt. Um festzulegen, dass Operationen asynchron ausgeführt werden, öffnen Sie die Verbindung zur Datenbank mit der `SQLConnection.openAsync()`-Methode statt mit der `SQLConnection.open()`-Methode. Wenn Datenbankoperationen asynchron ausgeführt werden, laufen sie im Hintergrund. Die Datenbankengine wird nicht in der Laufzeitbildschleife ausgeführt, sodass es durch Datenbankoperationen sehr wahrscheinlich nicht zu Renderingproblemen kommt.

  Informationen zu weiteren Strategien zur Leistungsverbesserung mit lokalen SQL-Datenbanken finden Sie unter „SQL-Datenbankleistung“ auf Seite 91.

- Eigenständige Pixel Bender-Shader

  Mit der ShaderJob-Klasse können Sie ein Bild oder einen Satz von Daten durch einen Pixel Bender-Shader ausführen und auf die unformatierten Ergebnisdaten zugreifen. Standardmäßig wird der Shader asynchron ausgeführt, wenn Sie die `ShaderJob.start()`-Methode aufrufen. Die Ausführung erfolgt im Hintergrund; die Laufzeitbildschleife wird nicht verwendet. Um die synchrone Ausführung des ShaderJob-Objekts zu erzwingen (was nicht empfohlen wird), übergeben Sie den Wert `true` an den ersten Parameter der `start()`-Methode.

Neben diesen integrierten Techniken zum asynchronen Ausführen von Code können Sie Ihren eigenen Code auch so strukturieren, dass er asynchron statt synchron ausgeführt wird. Wenn Sie Code schreiben, der eine potenziell lang dauernde Aufgabe ausführen soll, können Sie den Code so strukturieren, dass er in Teilen ausgeführt wird. Indem Sie den Code aufteilen, ermöglichen Sie der Laufzeitumgebung, die Rendervorgänge zwischen der Ausführung von Codeblöcken auszuführen, sodass es seltener zu Renderingproblemen kommt.

Nachstehend sind verschiedene Techniken zum Aufteilen des Codes aufgeführt. Der Grundgedanke hinter diesen Techniken ist es, den Code so zu schreiben, dass jeweils nur ein Teil der Aufgabe ausgeführt wird. Sie verfolgen, was der Code tut und wo er seine Arbeit unterbricht. Sie verwenden einen Mechanismus wie ein Timer-Objekt, um wiederholt zu prüfen, ob Aufgaben verbleiben, und um zusätzliche Aufgaben portionsweise auszuführen, bis die Arbeit abgeschlossen ist.

Es gibt einige bewährte Muster für die Strukturierung von Code, um Arbeit auf diese Weise aufzuteilen. In den folgenden Artikeln und Codebibliotheken werden diese Muster beschrieben und es wird Code bereitgestellt, der Ihnen bei der Implementierung der Muster in Ihre Anwendungen hilft:

- Asynchronous ActionScript Execution (Artikel von Trevor McCauley mit weiteren Hintergrundinformationen sowie verschiedenen Implementierungsbeispielen)
- Parsing & Rendering Lots of Data in Flash Player (Artikel von Jesse Warden mit Hintergrundinformationen und Beispielen für zwei Ansätze: „builder pattern“ und „green threads“)
- Green Threads (Artikel von Drew Cummins, in dem die „green threads“-Technik beschrieben wird, mit Beispielquellcode)
- greenthreads (Open-Source-Codebibliothek von Charlie Hubbard zur Implementierung von „green threads“ in ActionScript. Weitere Informationen finden Sie unter greenthreads Quick Start.)
- Threads in ActionScript 3 unter http://www.adobe.com/go/learn_fp_as3_threads_de (Artikel von Alex Harui mit einer Beispielimplementierung der „Pseudo-Threading“-Technik)

Transparente Fenster

Ziehen Sie bei AIR-Desktopanwendungen die Verwendung eines undurchsichtigen rechteckigen Anwendungsfensters anstelle eines durchsichtigen Fensters in Betracht.

Um ein undurchsichtiges Fenster zu verwenden, legen Sie für das Anfangsfenster der AIR-Desktopanwendung in der Anwendungsdeskriptor-XML-Datei den folgenden Wert fest:

```
<initialWindow>
    <transparent>false</transparent>
</initialWindow>
```

Für Fenster, die vom Anwendungscode erstellt werden, erstellen Sie ein NativeWindowInitOptions-Objekt mit der `transparent`-Eigenschaft auf `false` (Standardeinstellung). Übergeben Sie es an den NativeWindow-Konstruktor, während Sie das NativeWindow-Objekt erstellen:

```
// NativeWindow: flash.display.NativeWindow class

var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
initOptions.transparent = false;
var win:NativeWindow = new NativeWindow(initOptions);
```

Achten Sie bei einer Flex-Window-Komponente darauf, dass die transparent-Eigenschaft der Komponente auf den Standardwert „false“ gesetzt ist, bevor Sie die `open()`-Methode des Window-Objekts aufrufen.

```
// Flex window component: spark.components.Window class

var win:Window = new Window();
win.transparent = false;
win.open();
```

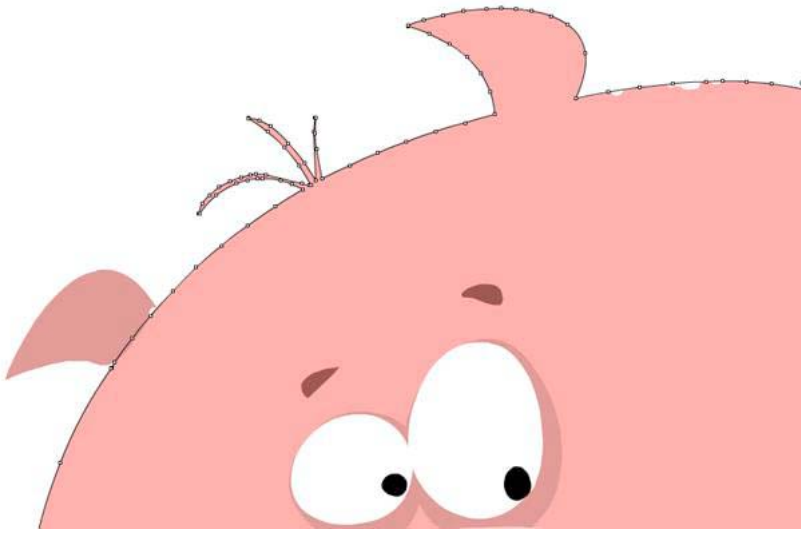
Ein transparentes Fenster zeigt potenziell einen Teil des Benutzerdesktops oder eines anderen Anwendungsfensters hinter dem Anwendungsfenster an. Deshalb braucht die Laufzeit mehr Ressourcen, um ein transparentes Fenster zu rendern. Bei einem rechteckigen, nicht transparenten Fenster ist das Rendering weniger aufwändig, unabhängig davon, ob das Fensterdesign des Betriebssystems oder ein benutzerdefiniertes Fensterdesign verwendet wird.

Verwenden Sie ein transparentes Fenster nur dann, wenn es wichtig ist, dass die Anzeige nicht rechteckig ist, oder wenn der Hintergrundinhalt im Anwendungsfenster sichtbar sein soll.

Glätten von Vektorformen

Durch das Glätten von Formen lässt sich die Leistung beim Rendern verbessern.

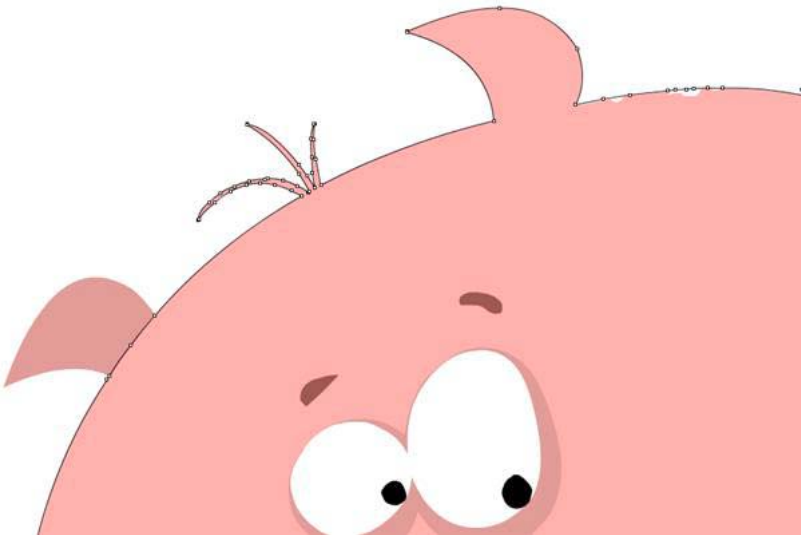
Im Gegensatz zu Bitmaps erfordert das Rendern von Vektorinhalten zahlreiche Berechnungen, besonders für Verläufe und komplexe Pfade, die zahlreiche Steuerpunkte enthalten. Designer und Entwickler sollten daher sicherstellen, dass Formen ausreichend optimiert sind. Die folgende Abbildung zeigt nicht vereinfachte Pfade mit zahlreichen Steuerpunkten:



Nicht optimierte Pfade

Mithilfe des Glättungswerkzeugs in Flash Professional können überflüssige Steuerpunkte entfernt werden. Ein ähnliches Werkzeug steht auch in Adobe® Illustrator® zur Verfügung. Die Gesamtanzahl der Punkte und Pfade ist im Bedienfeld mit den Dokumentinformationen ersichtlich.

Durch das Glätten können Sie überflüssige Steuerpunkte entfernen, die endgültige Größe der SWF-Datei verringern und die Leistung beim Rendern verbessern. Die folgende Abbildung zeigt dieselben Pfade nach dem Glätten:



Optimierte Pfade

Sofern Sie die Pfade nicht übermäßig vereinfachen, macht sich diese Optimierung visuell nicht bemerkbar. Die durchschnittliche Bildrate der fertigen Anwendung lässt sich durch die Vereinfachung komplexer Pfade jedoch erheblich verbessern.

Kapitel 6: Optimieren der Netzwerkinteraktion

Verbesserungen der Netzwerkinteraktion

In Flash Player 10.1 und AIR 2.5 wird eine neue Funktionsgruppe für die Netzwerkoptimierung auf allen Plattformen eingeführt, darunter Ringpufferung und intelligenter Suchlauf.

Ringpufferung

Beim Laden von Medieninhalten auf Mobilgeräten können Probleme auftreten, die bei Desktopcomputern sehr unwahrscheinlich sind. Beispielsweise können auf Mobilgeräten eher Situationen auftreten, in denen Arbeitsspeicher oder Festplattenkapazität nicht ausreichen. Beim Laden von Video wird die gesamte FLV-Datei (oder MP4-Datei) von den Desktopversionen von Flash Player 10.1 und AIR 2.5 auf die Festplatte heruntergeladen und im Cache zwischengespeichert. Dann wird das Video von dieser Cachedatei aus abgespielt. Es ist eher unwahrscheinlich, dass die Festplattenkapazität dazu nicht ausreicht, Sollte diese Situation eintreten, beendet die Desktop-Laufzeitumgebung die Wiedergabe des Videos.

Bei einem Mobilgerät kommt es eher zu Situationen, in denen die Festplattenkapazität erschöpft ist. Wenn auf dem Mobilgerät keine Festplattenkapazität mehr verfügbar ist, wird die Wiedergabe von der Laufzeitumgebung nicht angehalten, wie dies bei der Desktop-Laufzeitumgebung der Fall ist. Stattdessen beginnt die Laufzeitumgebung, die Cachedatei wiederzuverwenden, indem ab Anfang der Datei erneut in die Datei geschrieben wird. Der Benutzer kann das Video weiterhin ansehen, er kann jedoch nicht in dem Bereich des Videos suchen, der neu geschrieben wurde, mit Ausnahme des Dateianfangs. Die Ringpufferung wird nicht standardmäßig gestartet. Sie kann während der Wiedergabe und am Anfang der Wiedergabe gestartet werden, wenn die Festplattenkapazität oder der Arbeitsspeicher für den Film nicht ausreicht. Die Laufzeitumgebung benötigt für die Ringpufferung mindestens 4 MB RAM oder 20 MB Festplattenkapazität.

Hinweis: *Wenn auf dem Gerät nicht genügend Festplattenspeicher verfügbar ist, verhält sich die Mobilversion der Laufzeitumgebung wie die Desktopversion. Der Puffer im Arbeitsspeicher dient als Ausweichlösung, wenn ein Gerät keinen Festplattenspeicher hat oder wenn der Festplattenspeicher erschöpft ist. Bei der Kompilierung kann für die Cachedatei und den RAM-Puffer eine Größenbeschränkung festgelegt werden. Die Struktur einiger MP4-Dateien sieht vor, dass die ganze Datei heruntergeladen werden muss, bevor die Wiedergabe beginnen kann. Die Laufzeitumgebung erkennt diese Dateien und verhindert den Download, wenn nicht ausreichend Festplattenspeicher vorhanden ist und die MP4-Datei deshalb nicht abgespielt werden kann. Am besten ist es möglicherweise, solche Dateien überhaupt nicht herunterzuladen.*

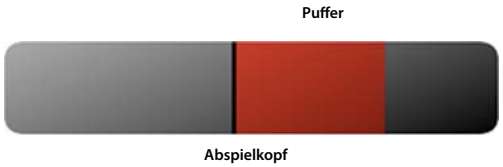
Entwickler müssen berücksichtigen, dass das Suchen nur innerhalb des im Cache zwischengespeicherten Streams funktioniert. `NetStream.seek()` schlägt manchmal fehl, wenn der Offset sich außerhalb des zulässigen Bereichs befindet. In diesem Fall wird ein `NetStream.Seek.InvalidTime`-Ereignis ausgelöst.

Intelligenter Suchlauf

Hinweis: *Für den intelligenten Suchlauf ist Adobe® Flash® Media Server 3.5.3 erforderlich.*

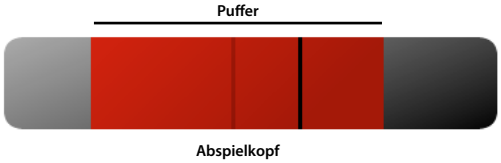
In Flash Player 10.1 und AIR 2.5 wird ein neues Verhalten eingeführt: intelligenter Suchlauf, wodurch das Benutzererlebnis beim Abspielen von Streaming-Video verbessert wird. Wenn der Benutzer einen Suchlauf zu einem Ziel innerhalb der Puffergrenzen ausführt, kann die Laufzeitumgebung den Puffer wiederverwenden, um einen sofortigen Suchlauf zu ermöglichen. In früheren Versionen der Laufzeitumgebung wurde der Puffer nicht wiederverwendet. Angenommen, der Benutzer spielte ein Video von einem Streaming-Server ab und die Pufferzeit war auf 20 Sekunden (`NetStream.bufferTime`) eingestellt. Wenn der Benutzer nun versuchte, 10 Sekunden vorwärts zu suchen, entfernte die Laufzeitumgebung alle Daten aus dem Puffer, anstatt die bereits geladenen 10 Sekunden erneut zu verwenden. Dieses Verhalten zwang die Laufzeitumgebung, viel häufiger neue Daten vom Server anzufordern, was bei langsameren Verbindungen zu einer mangelhaften Abspielleistung führte.

Die folgende Abbildung veranschaulicht das Verhalten des Puffers in früheren Versionen der Laufzeitumgebung. Die `bufferTime`-Eigenschaft definiert in Sekunden, wie viel Daten im Voraus geladen werden, damit der Puffer bei einem Verbindungsausfall verwendet werden kann und das Video nicht angehalten werden muss:

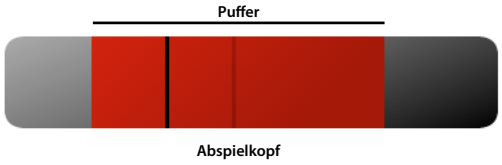


Pufferverhalten vor Einführung des intelligenten Suchlaufs

Mit der intelligenten Suchlauffunktion kann die Laufzeitumgebung jetzt den Puffer verwenden, um einen sofortigen Suchlauf vor oder zurück zu ermöglichen, wenn der Benutzer den Abspielkopf bewegt. Die folgende Abbildung veranschaulicht das neue Verhalten:



Vorwärts-Suchlauf mit der intelligenten Suchlauffunktion



Zurück-Suchlauf mit der intelligenten Suchlauffunktion

Beim intelligenten Suchlauf wird der Puffer wieder verwendet, wenn der Benutzer in Vorwärts- oder Rückwärtsrichtung sucht, was sich positiv auf die Geschwindigkeit und Qualität der Wiedergabe auswirkt. Dieses neue Verhalten bietet unter anderem den Vorteil, dass beim Veröffentlichen von Video Bandbreite eingespart wird. Wenn der Suchlauf die Puffergrenzen verlässt, zeigt die Laufzeitumgebung jedoch das Standardverhalten und fordert neue Daten vom Server an.

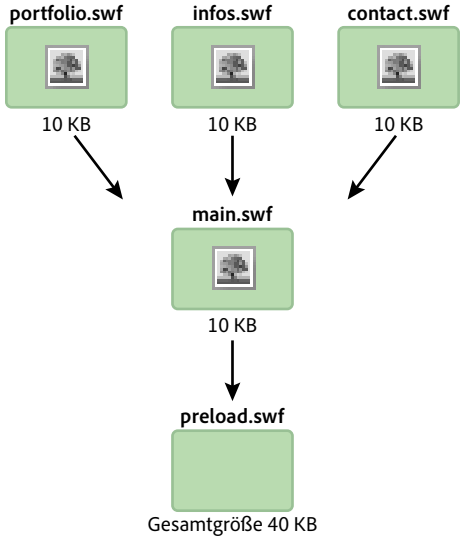
Hinweis: *Dieses Verhalten gilt nicht für den progressiven Download von Videos.*

Um den intelligenten Suchlauf zu verwenden, stellen Sie `NetStream.inBufferSeek` auf `true` ein.

Externe Inhalte

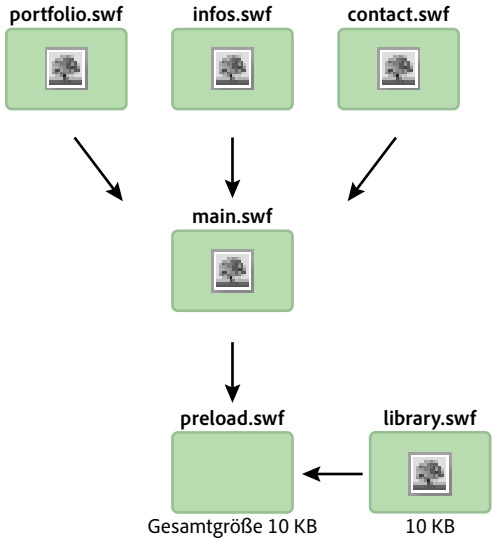
Unterteilen Sie Ihre Anwendung in mehrere SWF-Dateien.

Mobilgeräte haben möglicherweise nur eingeschränkten Zugang zum Netzwerk. Damit der Inhalt rasch geladen werden kann, sollten Sie Ihre Anwendung in mehrere SWF-Dateien unterteilen. Versuchen Sie, die Codelogik und Bestände in der ganzen Anwendung wiederzuverwenden. Nehmen wir als Beispiel eine Anwendung, die in mehrere SWF-Dateien unterteilt wurde, wie in der folgenden Abbildung gezeigt:



In mehrere SWF-Dateien unterteilte Anwendung

In diesem Beispiel enthält jede SWF-Datei eine eigene Kopie derselben Bitmap. Diese Doppelverwendung kann durch den Einsatz einer Runtime Shared Library vermieden werden, wie in der folgenden Abbildung gezeigt:



Einsatz einer Runtime Shared Library

Mit dieser Technik wird eine Runtime Shared Library geladen, die die Bitmap den anderen SWF-Dateien zur Verfügung stellt. Die ApplicationDomain-Klasse speichert alle geladenen Klassendefinitionen und stellt sie zur Laufzeit über die `getDefinition()`-Methode zur Verfügung.

Eine Runtime Shared Library kann auch die gesamte Codelogik enthalten. Die ganze Anwendung kann zur Laufzeit aktualisiert werden, ohne dass sie neu kompiliert werden muss. Der folgende Code lädt eine Runtime Shared Library und extrahiert zur Laufzeit die in der SWF-Datei enthaltene Definition. Diese Technik kann mit Schriftarten, Bitmaps, Sounds oder jeder ActionScript-Klasse verwendet werden:

```
// Create a Loader object
var loader:Loader = new Loader();

// Listen to the Event.COMPLETE event
loader.contentLoaderInfo.addEventListener(Event.COMPLETE, loadingComplete );

// Load the SWF file
loader.load(new URLRequest("library.swf") );
var classDefinition:String = "Logo";

function loadingComplete(e:Event ):void
{
    var objectLoaderInfo:LoaderInfo = LoaderInfo ( e.target );

    // Get a reference to the loaded SWF file application domain
    var appDomain:ApplicationDomain = objectLoaderInfo.applicationDomain;

    // Check whether the definition is available
    if ( appDomain.hasDefinition(classDefinition) )
    {
        // Extract definition
        var importLogo:Class = Class ( appDomain.getDefinition(classDefinition) );

        // Instantiate logo
        var instanceLogo:BitmapData = new importLogo(0,0);

        // Add it to the display list
        addChild ( new Bitmap ( instanceLogo ) );
    } else trace ("The class definition " +  classDefinition + " is not available.");
}
```

Das Abrufen der Definition kann vereinfacht werden, indem die Klassendefinitionen in der Anwendungsdomäne der geladenen SWF-Datei geladen werden.

```
// Create a Loader object
var loader:Loader = new Loader();

// Listen to the Event.COMPLETE event
loader.contentLoaderInfo.addEventListener ( Event.COMPLETE, loadingComplete );

// Load the SWF file
loader.load ( new URLRequest ("rsl.swf"), new LoaderContext ( false,
ApplicationDomain.currentDomain) );
var classDefinition:String = "Logo";

function loadingComplete ( e:Event ):void
{
    var objectLoaderInfo:LoaderInfo = LoaderInfo ( e.target );

    // Get a reference to the current SWF file application domain
    var appDomain:ApplicationDomain = ApplicationDomain.currentDomain;

    // Check whether the definition is available
    if (appDomain.hasDefinition( classDefinition ) )
    {
        // Extract definition
        var importLogo:Class = Class ( appDomain.getDefinition(classDefinition) );

        // Instantiate it
        var instanceLogo:BitmapData = new importLogo(0,0);

        // Add it to the display list
        addChild ( new Bitmap ( instanceLogo ) );
    } else trace ("The class definition " +  classDefinition + " is not available.");
}
```

Dann können die Klassen, die in der geladenen SWF-Datei zur Verfügung stehen, über einen Aufruf der `getDefinition()`-Methode in der aktuellen Anwendungsdomäne verwendet werden. Sie können auch durch Aufrufen der `getDefinitionByName()`-Methode auf die Klassen zugreifen. Bei dieser Technik wird Bandbreite eingespart, da Schriftarten und große Bestände nur einmal geladen werden. Bestände werden nie in andere SWF-Dateien exportiert. Es gilt nur eine Einschränkung, und zwar muss die Anwendung getestet und über die loader.swf-Datei ausgeführt werden. Diese Datei lädt zuerst die Bestände und dann die SWF-Dateien, aus denen die Anwendung sich zusammensetzt.

Eingabe-/Ausgabefehler

Implementieren Sie Ereignisprozeduren und Fehlermeldungen für E/A-Fehler.

Das Netzwerk ist bei Mobilgeräten oft weniger zuverlässig als bei einem Desktopcomputer, der über eine Internetverbindung mit hoher Geschwindigkeit verfügt. Der Zugriff auf externe Inhalte bei Mobilgeräten unterliegt zwei Einschränkungen: Verfügbarkeit und Geschwindigkeit. Deshalb sollten Sie Bestände verwenden, die möglichst wenig Ressourcen belegen, und Prozeduren für jedes IO_ERROR-Ereignis hinzufügen, damit die Benutzer Fehlermeldungen erhalten.

Angenommen, ein Benutzer hat Ihre Website von einem mobilen Gerät aus aufgerufen, aber die Netzwerkverbindung wird zwischen zwei U-Bahn-Haltestellen unterbrochen. Zu diesem Zeitpunkt wurde gerade ein dynamischer Bestand geladen. Da dieses Szenario auf einem Desktop praktisch nie auftritt, können Sie dort einen leeren Ereignis-Listener verwenden, um zu verhindern, dass ein Laufzeitfehler eingeblendet wird. Auf einem Mobilgerät ist ein leerer Ereignis-Listener dagegen unzureichend.

Der folgende Code reagiert nicht auf einen E/A-Fehler. Verwenden Sie den Code nicht in der gezeigten Form:

```
var loader:Loader = new Loader();
loader.contentLoaderInfo.addEventListener( Event.COMPLETE, onComplete );
addChild( loader );
loader.load( new URLRequest ("asset.swf" ) );

function onComplete( e:Event ):void
{
    var loader:Loader = e.currentTarget.loader;
    loader.x = ( stage.stageWidth - e.currentTarget.width ) >> 1;
    loader.y = ( stage.stageHeight - e.currentTarget.height ) >> 1;
}
```

Stattdessen sollte für eine derartige Fehlersituation eine Prozedur implementiert werden, die eine Fehlermeldung für den Benutzer einblendet. Mit dem folgenden Code wird angemessen auf den Fehler reagiert:

```
var loader:Loader = new Loader();
loader.contentLoaderInfo.addEventListener ( Event.COMPLETE, onComplete );
loader.contentLoaderInfo.addEventListener ( IOErrorEvent.IO_ERROR, onIOError );
addChild ( loader );
loader.load ( new URLRequest ("asset.swf" ) );

function onComplete ( e:Event ):void
{
      var loader:Loader = e.currentTarget.loader;
      loader.x = ( stage.stageWidth - e.currentTarget.width ) >> 1;
      loader.y = ( stage.stageHeight - e.currentTarget.height ) >> 1;
}

function onIOError ( e:IOErrorEvent ):void
{
      // Show a message explaining the situation and try to reload the asset.
      // If it fails again, ask the user to retry when the connection will be restored
}
```

Es empfiehlt sich, dem Benutzer die Möglichkeit zu geben, den Inhalt erneut zu laden. Dieses Verhalten kann über die `onIOError()`-Prozedur implementiert werden.

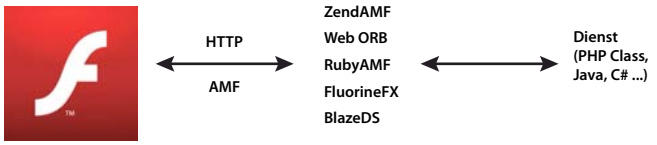
Flash Remoting

Verwenden Sie Flash Remoting und AMF für eine optimierte Client-/Server-Datenkommunikation.

Sie können XML zum Laden von Remote-Inhalt in SWF-Dateien verwenden. Bei XML handelt es sich jedoch um reinen Text, der von der Laufzeitumgebung geladen und analysiert wird. XML eignet sich am besten für Anwendungen, die nur eine eingeschränkte Inhaltsmenge laden. Bei der Entwicklung von Anwendungen, die große Mengen an Inhalt laden, sollten Sie die Flash Remoting-Technologie und das Action Message Format (AMF) in Betracht ziehen.

AMF ist ein Binärformat zum Austauschen von Daten zwischen einem Server und der Laufzeitumgebung. AMF reduziert die Datengröße und beschleunigt die Übertragung. Da AMF ein natives Format für die Laufzeitumgebung ist, wird beim Senden von AMF-Daten an die Laufzeitumgebung die arbeitsspeicherintensive Serialisierung und Entserialisierung auf der Clientseite vermieden. Diese Aufgaben werden vom Remoting-Gateway gehandhabt. Beim Senden eines ActionScript-Datentyps an einen Server übernimmt der Remoting-Gateway die Serialisierung am Server. Das Gateway sendet Ihnen auch den entsprechenden Datentyp. Dieser Datentyp ist eine Klasse, die auf dem Server erstellt wird und Methoden bereitstellt, die von der Laufzeitumgebung aufgerufen werden können. Zu den Flash Remoting-Gateways zählen ZendAMF, FluorineFX, WebORB und BlazeDS, ein offizieller Java Flash Remoting-Gateway auf Open-Source-Basis von Adobe.

Die folgende Abbildung veranschaulicht das Konzept von Flash Remoting:



Flash Remoting

Im folgenden Beispiel wird die NetConnection-Klasse verwendet, um eine Verbindung mit dem Flash Remoting-Gateway herzustellen:

```
// Create the NetConnection object
var connection:NetConnection = new NetConnection ();

// Connect to a Flash Remoting gateway
connection.connect ("http://www.yourserver.com/remotingservice/gateway.php");

// Asynchronous handlers for incoming data and errors
function success ( incomingData:* ):void
{
    trace( incomingData );
}

function error ( error:* ):void
{
    trace( "Error occured" );
}

// Create an object that handles the mapping to success and error handlers
var serverResult:Responder = new Responder (success, error);

// Call the remote method
connection.call ("org.yourserver.HelloWorld.sayHello", serverResult, "Hello there ?");
```

Eine Verbindung mit einem Remoting-Gateway lässt sich sehr einfach herstellen. Die Verwendung von Flash Remoting kann jedoch mit der RemoteObject-Klasse im Adobe® Flex® SDK noch weiter vereinfacht werden.

Hinweis: *Externe SWC-Dateien, beispielsweise aus dem Flex Framework, können innerhalb eines Projekts in Adobe® Flash® Professional verwendet werden. Bei Verwendung von SWC-Dateien können Sie die RemoteObject-Klasse und ihre abhängigen Elemente verwenden, ohne den Rest des Flex SDK nutzen zu müssen. Erfahrene Entwickler können sogar über die reine Socket-Klasse direkt mit einem Remoting-Gateway kommunizieren, sofern erforderlich.*

Unnötige Netzwerkoperationen

Speichern Sie Bestände nach dem Laden lokal im Cache, anstatt sie jedes Mal, wenn sie gebraucht werden, aus dem Netzwerk zu laden.

Wenn Ihre Anwendung Bestände wie Medien oder Daten lädt, speichern Sie diese Bestände im Cache, indem Sie sie auf dem lokalen Gerät speichern. Für Bestände, die sich selten ändern, können Sie den Cachespeicher ggf. in bestimmten Abständen aktualisieren. Zum Beispiel könnte Ihre Anwendung einmal am Tag überprüfen, ob eine neue Version einer Bilddatei verfügbar ist, oder alle zwei Stunden, ob neue Daten vorliegen.

Sie können Bestände auf verschiedene Arten im Cache speichern, je nach Typ und Charakter der Bestände:

- Medienbestände wie Bilder und Video: Speichern Sie die Dateien mithilfe der File- und FileStream-Klassen im Dateisystem.
- Einzelne Datenwerte oder kleine Datensätze: Speichern Sie die Werte mithilfe der SharedObject-Klasse als lokale gemeinsam genutzte Objekte.
- Größere Datensätze: Speichern Sie die Daten in einer lokalen Datenbank oder serialisieren Sie die Daten und speichern Sie sie in einer Datei.

Für das Zwischenspeichern von Datenwerten finden Sie unter open-source AS3CoreLib project eine ResourceCache-Klasse, die das Laden und Zwischenspeichern für Sie erledigt.

Kapitel 7: Arbeiten mit Medien

Video

Informationen zum Optimieren von Video auf mobilen Geräten finden Sie unter Optimize web content for mobile delivery auf der Website der Adobe Developer Connection.

Sehen Sie sich besonders die folgenden Abschnitte an:

- *Abspielen von Video auf mobilen Geräten*
- *Codebeispiele*

Diese Abschnitte enthalten Informationen zum Entwickeln von Videoplayern für mobile Geräte, zum Beispiel:

- Richtlinien zur Videokodierung
- Empfohlene Verfahren
- Erstellen eines Leistungsprofils für den Videoplayer
- Eine Beispielimplementierung eines Videoplayers

StageVideo

Verwenden Sie die StageVideo-Klasse, um beim Darstellen von Videos die Vorteile der Hardwarebeschleunigung zu nutzen.

Informationen zur Verwendung des StageVideo-Objekts finden Sie unter Verwenden der StageVideo-Klasse für die hardwarebeschleunigte Darstellung im ActionScript 3.0-Entwicklerhandbuch.

Audio

Seit Flash Player 9.0.115.0 und AIR 1.0 kann die Laufzeitumgebung AAC-Dateien abspielen (AAC Main, AAC LC und SBR). Eine einfache Optimierung lässt sich erzielen, indem AAC-Dateien anstelle von MP3-Dateien verwendet werden. Das AAC-Format bietet eine höhere Qualität und eine geringere Dateigröße als das MP3-Format bei einer gleichwertigen Bitrate. Je kleiner die Datei, desto weniger Bandbreite ist erforderlich – ein wichtiger Faktor bei Mobilgeräten, die nicht über schnelle Internetverbindungen verfügen.

Hardware-Audiodekodierung

Wie die Videodekodierung führt auch die Audiodekodierung zu einer hohen Beanspruchung der CPU. Eine Optimierung ist möglich, indem die verfügbare Hardware des Geräts genutzt wird. Flash Player 10.1 und AIR 2.5 können Hardware-Audiotreiber erkennen und verwenden, um die Leistung beim Dekodieren von AAC-Dateien (LC, HE/SBR-Profile) oder MP3-Dateien zu verbessern (PCM wird nicht unterstützt). Die CPU-Auslastung wird deutlich reduziert, wodurch weniger Akkustrom benötigt wird und die CPU für andere Vorgänge zur Verfügung steht.

Hinweis: *Bei Verwendung des AAC-Formats wird das AAC Main Profile nicht unterstützt, da die meisten Geräte keine Hardware-Unterstützung bieten.*

Die Hardware-Audiodekodierung ist für Anwender und Entwickler transparent. Wenn die Laufzeitumgebung mit der Wiedergabe von Audiostreams beginnt, wird zunächst – wie bei Video – die Hardware überprüft. Wenn ein Hardwaretreiber verfügbar ist und das Audioformat unterstützt wird, wird die Hardware-Audiodekodierung durchgeführt. Doch selbst wenn die Dekodierung des eingehenden AAC- oder MP3-Streams von der Hardware erledigt werden kann, ist die Hardware in manchen Fällen nicht in der Lage, alle Effekte zu verarbeiten. So kann die Hardware manchmal Audiomischungen oder Resampling nicht verarbeiten, wenn bestimmte Einschränkungen der Hardware dies verhindern.

Kapitel 8: SQL-Datenbankleistung

Anwendungsdesign für die Datenbankleistung

Ändern Sie die `text`-Eigenschaft eines SQLStatement-Objekts nicht, nachdem Sie es ausgeführt haben. Verwenden Sie stattdessen eine SQLStatement-Instanz für jede SQL-Anweisung und Anweisungsparameter, um verschiedene Werte bereitzustellen.

Bevor eine SQL-Anweisung ausgeführt wird, wird sie von der Laufzeitumgebung vorbereitet (kompiliert), um festzustellen, welche Schritte intern ausführt werden müssen, um die Anweisung auszuführen. Wenn Sie `SQLStatement.execute()` für eine SQLStatement-Instanz aufrufen, die zuvor noch nicht ausgeführt wurde, wird die Anweisung automatisch vorbereitet, bevor sie ausgeführt wird. Bei nachfolgenden Aufrufen der `execute()`-Methode ist die Anweisung dann bereits vorbereitet, sofern die `SQLStatement.text`-Eigenschaft nicht geändert wurde. Deshalb wird sie schneller ausgeführt.

Den größten Vorteil erzielen Sie, wenn Sie die Anweisung mit Anweisungsparametern anpassen. So können Anweisungen wiederverwendet werden, selbst wenn sich Werte in der Anweisung ändern. (Anweisungsparameter werden mit der assoziativen Array-Eigenschaft `SQLStatement.parameters` angegeben.) Anders als beim Ändern der `text`-Eigenschaft der SQLStatement-Instanz muss die Laufzeitumgebung die Anweisung nicht erneut vorbereiten, wenn Sie die Werte der Anweisungsparameter ändern.

Wenn Sie eine SQLStatement-Instanz wiederverwenden, muss Ihre Anwendung einen Verweis auf die SQLStatement-Instanz speichern, nachdem sie vorbereitet wurde. Deklarieren Sie dazu die Variable als Variable mit Klassenumfang anstatt mit Funktionsumfang. Eine gute Vorgehensweise, die SQLStatement in eine Variable mit Klassenumfang zu ändern, besteht darin, die Anwendung so zu strukturieren, dass eine SQL-Anweisung in eine einzelne Klasse gesetzt wird. Auch eine Gruppe von Anweisungen, die im Zusammenhang ausgeführt werden, lassen sich von einer einzelnen Klasse umhüllen. (Diese Technik wird als „Command Design Pattern“ bezeichnet.) Indem Sie die Instanzen als Mitgliedsvariablen der Klasse definieren, bleiben sie erhalten, solange die Instanz der „umhüllenden“ Klasse in der Anwendung vorhanden ist. Sie können im Mindestfall einfach eine Variable definieren, die die SQLStatement-Instanz außerhalb einer Funktion enthält, sodass die Instanz im Speicher verbleibt. Deklarieren Sie die SQLStatement-Instanz zum Beispiel als Mitgliedsvariable in einer ActionScript-Klasse oder als Nicht-Funktions-Variable in einer JavaScript-Datei. Sie können dann die Werte der Anweisungsparameter festlegen und die `execute()`-Methode aufrufen, wenn die Abfrage dann ausgeführt werden soll.

Verwenden Sie Datenbankindizes, um die Ausführungsgeschwindigkeit beim Vergleichen und Sortieren von Daten zu erhöhen.

Wenn Sie einen Index für eine Spalte erstellen, speichert die Datenbank eine Kopie der Daten in dieser Spalte. Die Kopie wird in numerisch oder alphabetisch sortierter Reihenfolge gehalten. Durch diese Sortierung kann die Datenbank schnell Werte abgleichen (zum Beispiel bei Verwendung des Gleichheitsoperators) und die Ergebnisdaten mithilfe der `ORDER BY`-Klausel sortieren.

Datenbankindizes werden fortlaufend auf dem neuesten Stand gehalten. Deshalb dauern Datenänderungen (INSERT oder UPDATE) für diese Tabelle etwas länger. Die Daten können möglicherweise jedoch erheblich schneller abgerufen werden. Wegen dieser Leistungseinschränkung sollten Sie nicht einfach jede Spalte jeder Tabelle mit einem Index versehen. Verwenden Sie stattdessen eine Strategie zur Definition Ihrer Indizes. Orientieren Sie sich an den folgenden Richtlinien, um Ihre Indexierungsstrategie zu planen:

- Indexieren Sie Spalten, die bei Tabellenzusammenführungen, in WHERE-Klauseln oder in ORDER BY-Klauseln verwendet werden.

- Wenn Spalten häufig zusammen verwendet werden, indexieren Sie sie mit einem gemeinsamen Index.
- Für Spalten mit Textdaten, die Sie alphabetisch sortiert abrufen, legen Sie die COLLATE NOCASE-Sortierreihenfolge für den Index fest.


Ziehen Sie das Vorkompilieren von SQL-Anweisungen während Leerlaufzeiten der Anwendung in Betracht.

Wenn eine SQL-Anweisung zum ersten Mal ausgeführt wird, ist dies langsamer, da der SQL-Text von der Datenbankengine vorbereitet (kompiliert) wird. Da die Vorbereitung und Ausführung einer Anweisung ressourcenintensiv sein können, bietet es sich möglicherweise an, die anfänglichen Daten im Voraus zu laden und dann andere Anweisungen im Hintergrund auszuführen:

1. Laden Sie die Daten, die die Anwendung zuerst benötigt..
2. Nachdem die ersten Startoperationen der Anwendung abgeschlossen sind oder wenn die Anwendung gerade nicht beschäftigt ist, führen Sie andere Anweisungen aus.

Angenommen, Ihre Anwendung greift zum Beispiel überhaupt nicht auf die Datenbank zu, um den Anfangsbildschirm anzuzeigen. Warten Sie in diesem Fall, bis dieser Bildschirm angezeigt wird, bevor Sie die Datenbankverbindung öffnen. Erstellen Sie dann die SQLStatement-Instanzen und führen Sie so viel Code wie möglich aus.

Es kann aber auch sein, dass Ihre Anwendung beim Starten sofort einige Daten anzeigt, zum Beispiel das Ergebnis einer bestimmten Abfrage. Führen Sie in diesem Fall die SQLStatement-Instanz für diese Abfrage aus. Nachdem die ersten Daten geladen und angezeigt werden, erstellen Sie SQLStatement-Instanzen für anderen Datenbankoperationen und führen Sie nach Möglichkeit andere Anweisungen aus, die später benötigt werden.

In der Praxis ist die zusätzliche Zeit für das Vorbereiten der Anweisung nur einmal erforderlich. Sie hat vermutlich keine großen Auswirkungen auf die Gesamtleistung.


Gruppieren Sie mehrere SQL-Datenänderungsoperationen in einer Transaktion.

Angenommen, Sie führen zahlreiche SQL-Anweisungen aus, die das Hinzufügen oder Ändern von Daten beinhalten (`INSERT`- oder `UPDATE`-Anweisungen). Sie können die Leistung deutlich verbessern, indem Sie alle Anweisungen in einer expliziten Transaktion ausführen. Wenn Sie eine Transaktion nicht explizit beginnen, wird jede Anweisung in ihrer eigenen automatisch erstellten Transaktion ausgeführt. Nachdem jede Transaktion (Anweisung) abgeschlossen ist, schreibt die Laufzeitumgebung die resultierenden Daten in die Datenbankdatei auf der Festplatte.

Überlegen Sie, was passiert, wenn Sie dagegen explizit eine Transaktion erstellen und die Anweisungen im Kontext der Transaktion ausführen. Die Laufzeitumgebung nimmt alle Änderungen im Speicher vor und schreibt dann alle Änderungen an der Datenbankdatei zur gleichen Zeit, wenn die Transaktion ausgeführt wird. Das Schreiben der Daten auf die Festplatte ist normalerweise der zeitaufwändigste Teil der Operation. Dementsprechend können Sie die Leistung erheblich verbessern, wenn es nur einen Schreibvorgang gibt anstatt einen pro SQL-Anweisung.


Verarbeiten Sie große `SELECT`-Abfrageergebnisse in Portionen, indem Sie die `execute()`-Methode (mit dem `prefetch`-Parameter) und die `next()`-Methode der SQLStatement-Klasse verwenden.

Angenommen, Sie führen eine AQL-Anweisung aus, die einen großen Ergebnissatz abruft. Die Anwendung verarbeitet dann jede Datenzeile in einer Schleife. Zum Beispiel formatiert sie die Daten oder erstellt Objekte daraus. Das Verarbeiten der Daten kann viel Zeit in Anspruch nehmen, was zu Renderingproblemen wie einem nicht reagierenden Bildschirm führen kann. Wie unter „Asynchrone Operationen“ auf Seite 77 beschrieben, besteht eine Lösung darin, die Arbeit in Abschnitte aufzuteilen. Mit der SQL-Datenbank-API ist die Aufteilung der Datenverarbeitung ein einfacher Vorgang.

Die `execute()`-Methode der SQLStatement-Klasse hat einen optionalen `prefetch`-Parameter (dies ist der erste Parameter). Wenn Sie einen Wert dafür angeben, legt dieser die maximale Anzahl von Ergebniszeilen fest, die die Datenbank zurückgibt, wenn die Ausführung abgeschlossen ist:

```
dbStatement.addEventListener(SQLEvent.RESULT, resultHandler);
dbStatement.execute(100); // 100 rows maximum returned in the first set
```

Nachdem der erste Satz von Ergebnisdaten zurückgegeben wurde, können Sie die `next()`-Methode aufrufen, um die Ausführung der Anweisung fortzusetzen und einen weiteren Satz mit Ergebniszeilen abzurufen. Wie die `execute()`-Methode akzeptiert auch die `next()`-Methode einen `prefetch`-Parameter, um die maximale Anzahl der zurückzugebenden Ergebniszeilen festzulegen:

```
// This method is called when the execute() or next() method completes
function resultHandler(event:SQLEvent):void
{
    var result:SQLResult = dbStatement.getResult();
    if (result != null)
    {
        var numRows:int = result.data.length;
        for (var i:int = 0; i < numRows; i++)
        {
            // Process the result data
        }

        if (!result.complete)
        {
            dbStatement.next(100);
        }
    }
}
```

Sie können die `next()`-Methode wiederholt aufrufen, bis alle Daten geladen wurden. Wie im vorherigen Beispiel gezeigt, können Sie bestimmen, wann alle Daten geladen wurden. Überprüfen Sie die `complete`-Eigenschaft des SQLResult-Objekts, das jedes Mal erstellt wird, wenn die `execute()`- oder `next()`-Methode abgeschlossen ist.

Hinweis: *Verwenden Sie den `prefetch`-Parameter und die `next()`-Methode, um die Verarbeitung der Ergebnisdaten aufzuteilen. Verwenden Sie diesen Parameter und diese Methode nicht, um die Ergebnisse einer Abfrage auf einen Teil der Ergebnismenge einzuschränken. Wenn Sie nur eine Untermenge der Zeilen in der Ergebnismenge einer Anweisung abrufen möchten, verwenden Sie die `LIMIT`-Klausel der `SELECT`-Anweisung. Bei einem großen Ergebnissatz können Sie immer noch den `prefetch`-Parameter und die `next()`-Methode verwenden, um die Verarbeitung der Ergebnisse aufzuteilen.*


Verwenden Sie ggf. mehrere asynchrone SQLConnection-Objekte mit einer einzelnen Datenbank, um mehrere Anweisungen gleichzeitig auszuführen.

Wenn ein SQLConnection-Objekt mithilfe der `openAsync()`-Methode mit einer Datenbank verbunden ist, wird sie im Hintergrund ausgeführt anstatt im Hauptausführungs-Thread der Laufzeitumgebung. Des Weiteren wird jede SQLConnection in ihrem eigenen Hintergrund-Thread ausgeführt. Durch Verwendung mehrerer SQLConnection-Objekte können Sie effektiv mehrere SQL-Anweisungen gleichzeitig ausführen.

Dieser Ansatz hat jedoch auch potenzielle Nachteile. Der wichtigste ist, dass jedes zusätzliche SQLStatement-Objekt zusätzlichen Arbeitsspeicher benötigt. Außerdem bedeutet die gleichzeitige Ausführung auch Mehrarbeit für den Prozessor, besonders bei Computern mit nur einer CPU oder einem CPU-Kern. Aufgrund dieser Nachteile wird dieser Ansatz für mobile Geräte nicht empfohlen.

Ein weiterer Nachteil ist, dass der potenzielle Vorteil aus der Wiederverwendung von SQLStatement-Objekten verloren geht, wenn ein SQLStatement-Objekt mit einem einzelnen SQLConnection-Objekt verknüpft ist. Deshalb kann das SQLStatement-Objekt nicht wiederverwendet werden, wenn es einem bereits verwendeten SQLConnection-Objekt zugewiesen ist.

Wenn Sie mehrere SQLConnection-Objekte verwenden, die mit einer einzelnen Datenbank verbunden sind, denken Sie daran, dass jedes Objekt seine Anweisung in einer eigenen Transaktion ausführt. Berücksichtigen Sie diese separaten Transaktionen in jedem Code, der Daten ändert (beispielsweise durch Hinzufügen, Ändern oder Löschen).

Paul Robertson hat eine Open-Source-Codebibliothek erstellt, die Sie beim Integrieren der Vorteile der Verwendung von mehreren SQLConnection-Objekten unterstützt, während die möglichen Nachteile minimiert werden. Die Bibliothek verwendet einen Pool von SQLConnection-Objekten und verwaltet die damit verknüpften SQLStatement-Objekte. Auf diese Weise wird sichergestellt, dass SQLStatement-Objekte wiederverwendet werden und mehrere SQLConnection-Objekte verfügbar sind, um mehrere Anweisungen gleichzeitig auszuführen. Wenn Sie weitere Informationen wünschen oder die Bibliothek herunterladen möchten, besuchen Sie http://probertson.com/projects/air-sqlite/.

Optimierung der Datenbankdatei

Vermeiden Sie Änderungen am Datenbankschema.

Vermeiden Sie nach Möglichkeit Änderungen am Schema (an der Tabellenstruktur) einer Datenbank, nachdem Sie den Datenbanktabellen Daten hinzugefügt haben. Normalerweise ist eine Datenbankdatei mit den Tabellendefinitionen am Anfang der Datei strukturiert. Wenn Sie eine Verbindung zu einer Datenbank herstellen, lädt die Laufzeitumgebung diese Definitionen. Wenn Sie den Datenbanktabellen Daten hinzufügen, werden die Daten der Datei hinter den Tabellendefinitionsdaten hinzugefügt. Wenn Sie jedoch Schemaänderungen vornehmen, werden die neuen Tabellendefinitionsdaten mit den Tabellendaten in der Datenbankdatei kombiniert. Wenn Sie beispielsweise einer Tabelle eine Spalte hinzufügen oder eine neue Tabelle hinzufügen, kann dies dazu führen, dass die Datentypen kombiniert werden. Wenn die Tabellendefinitionsdaten sich nicht am Anfang der Datenbankdatei befinden, dauert das Öffnen einer Verbindung mit der Datenbank länger. Das Öffnen der Verbindung dauert länger, da die Laufzeit mehr Zeit braucht, um die Tabellendefinitionsdaten aus den verschiedenen Teilen der Datei zu lesen.

Verwenden Sie die `SQLConnection.compact()`-Methode, um eine Datenbank nach Schemaänderungen zu optimieren.

Wenn Schemaänderungen unumgänglich sind, können Sie die `SQLConnection.compact()`-Methode aufrufen, nachdem Sie die Änderungen vorgenommen haben. Diese Operation strukturiert die Datenbankdatei neu, sodass die Tabellendefinitionsdaten sich zusammen am Anfang der Datei befinden. Die `compact()`-Operation kann jedoch einige Zeit in Anspruch nehmen, besonders, wenn die Datenbankdatei an Größe zunimmt.

Unnötige Datenbankverarbeitung zur Laufzeit

Verwenden Sie einen vollständig qualifizierten Tabellennamen (einschließlich Datenbankname) in Ihrer SQL-Anweisung.

Geben Sie den Datenbanknamen immer zusammen mit den einzelnen Tabellennamen in einer Anweisung an. (Verwenden Sie „main“, wenn es sich um die Hauptdatenbank handelt.) Der folgende Code schließt zum Beispiel einen expliziten Datenbanknamen `main` ein:

```
SELECT employeeId
FROM main.employees
```

Wenn Sie den Datenbanknamen angeben, muss die Laufzeitumgebung nicht in allen verbundenen Datenbanken nach der richtigen Tabelle suchen. So verhindern Sie außerdem, dass die Laufzeitumgebung die falsche Datenbank wählt. Halten Sie sich auch dann an diese Regel, wenn eine SQLConnection-Instanz nur mit einer einzelnen Datenbank verbunden ist. Die SQLConnection-Instanz ist im Hintergrund auch mit einer temporären Datenbank verbunden, auf die mit SQL-Anweisungen zugegriffen werden kann.

Verwenden Sie explizite Spaltennamen in den SQL-Anweisungen `INSERT` und `SELECT`.

Das folgende Beispiel veranschaulicht die Verwendung expliziter Spaltennamen:

```
INSERT INTO main.employees (firstName, lastName, salary)
VALUES ("Bob", "Jones", 2000)

SELECT employeeId, lastName, firstName, salary
FROM main.employees
```

Vergleichen Sie die vorstehenden Beispiele mit den folgenden. Vermeiden Sie diesen Codestil:

```
-- bad because column names aren't specified
INSERT INTO main.employees
VALUES ("Bob", "Jones", 2000)

-- bad because it uses a wildcard
SELECT *
FROM main.employees
```

Ohne die Angabe expliziter Spaltennamen muss die Laufzeitumgebung zusätzliche Arbeit leisten, um die Spaltennamen herauszufinden. Wenn eine `SELECT`-Anweisung ein Platzhalterzeichen anstelle von expliziten Spalten verwendet, ruft die Laufzeit zusätzliche Daten ab. Die zusätzlichen Daten erfordern zusätzliche Verarbeitung und erstellen zusätzliche Objektinstanzen, die nicht benötigt werden.

Vermeiden Sie, dieselbe Tabelle mehrmals in einer Anweisung zu verwenden, falls Sie die Tabelle nicht mit sich selbst vergleichen.

Wenn SQL-Anweisungen groß werden, kann es passieren, dass Sie eine Datenbanktabelle versehentlich mehrmals mit der Abfrage verbinden. Häufig lässt sich dasselbe Ergebnis erzielen, indem die Tabelle nur einmal verwendet wird. Das mehrmalige Verbinden einer Tabelle tritt wahrscheinlich dann auf, wenn Sie eine oder mehrere Ansichten in einer Abfrage verwenden. Dies ist beispielsweise der Fall, wenn Sie nicht nur eine Tabelle mit der Abfrage verbinden, sondern auch eine Ansicht, die Daten aus dieser Tabelle enthält. Durch diese beiden Vorgänge entsteht mehr als eine Verbindung.

Effiziente SQL-Syntax

Verwenden Sie `JOIN` (in der `FROM`-Klausel), um eine Tabelle in eine Abfrage einzubeziehen, anstatt einer Unterabfrage in der `WHERE`-Klausel. Dieser Tipp gilt auch, wenn Sie die Daten einer Tabelle nur zum Filtern, nicht für den Ergebnissatz benötigen.

Das Verbinden mehrerer Tabellen in der `FROM`-Klausel bringt eine bessere Leistung als die Verwendung einer Unterabfrage in einer `WHERE`-Klausel.

Vermeiden Sie SQL-Anweisungen, die Indizes nicht nutzen können. Diese Anweisungen umfassen die Verwendung von Aggregatfunktionen in einer Unterabfrage, eine `UNION`-Anweisung in einer Unterabfrage oder eine `ORDER BY`-Klausel in einer `UNION`-Anweisung.

Ein Index kann die Verarbeitung einer `SELECT`-Abfrage erheblich beschleunigen. Eine bestimmte SQL-Syntax verhindert jedoch, dass die Datenbank Indizes verwendet, und zwingt sie, die tatsächlichen Daten für Such- oder Sortieroperationen zu verwenden.

Vermeiden Sie ggf. den `LIKE`-Operator, besonders mit einem Platzhalterzeichen am Anfang wie in `LIKE('%XXXX%')`.

Da die `LIKE`-Operation die Verwendung von Platzhaltersuchen unterstützt, ist die Leistung hierbei langsamer als bei Verwendung von Vergleichen auf genaue Übereinstimmung. Insbesondere dann, wenn Sie den Suchstring mit einem Platzhalterzeichen beginnen, kann die Datenbank für die Suche überhaupt keine Indizes verwenden. Stattdessen muss die Datenbank den ganzen Text in jeder Tabellenzeile durchsuchen.

Vermeiden Sie eventuell den `IN`-Operator. Wenn die möglichen Werte im Voraus bekannt sind, kann die `IN`-Operation zur schnelleren Ausführung mit `AND` oder `OR` geschrieben werden.

Die zweite der beiden folgenden Anweisungen wird schneller ausgeführt. Dies liegt daran, dass sie einfache Gleichheitsausdrücke zusammen mit `OR` verwendet, anstatt die `IN()` oder `NOT IN()`-Anweisungen zu verwenden:

```
-- Slower
SELECT lastName, firstName, salary
FROM main.employees
WHERE salary IN (2000, 2500)

-- Faster
SELECT lastName, firstName, salary
FROM main.employees
WHERE salary = 2000
    OR salary = 2500
```

Ziehen Sie alternative Formen einer SQL-Anweisung in Betracht, um die Leistung zu verbessern.

Wie in Beispielen weiter oben gezeigt, kann auch die Art, wie eine SQL-Anweisung geschrieben ist, Auswirkungen auf die Datenbankleistung haben. Häufig gibt es mehrere Möglichkeiten, mit einer SQL-`SELECT`-Anweisung einen bestimmten Ergebnissatz abzurufen. In einigen Fällen ist ein bestimmter Ansatz deutlich schneller als ein anderer. Neben den vorstehenden Vorschlägen können Sie in dedizierten Ressourcen für die SQL-Sprache mehr über verschiedene SQL-Anweisungen und ihre Leistung erfahren.

Leistung von SQL-Anweisungen

Vergleichen Sie alternative SQL-Anweisungen direkt, um festzustellen, welche schneller ist.

Am besten vergleichen Sie die Leistung verschiedener Versionen einer SQL-Anweisung, indem Sie sie direkt mit Ihrer Datenbank und den Daten testen.

Die folgenden Entwicklungstools geben Ausführungszeiten beim Ausführen von SQL-Anweisungen an. Vergleichen Sie damit die Geschwindigkeit alternativer Versionen einer Anwendung:

- Run! (AIR SQL-Abfrage- und Authoringtool von Paul Robertson)
- Lita (SQLite-Administration-Tool von David Deraedt)

Kapitel 9: Vergleichswerte und Bereitstellung

Vergleichswerte

Es stehen verschiedene Tools zur Verfügung, um Anwendungen anhand von Vergleichswerten zu bewerten. Sie können die Stats- und die PerformanceTest-Klasse verwenden, die von Mitgliedern der Flash-Community entwickelt wurden. Außerdem können Sie den Profiler in Adobe® Flash® Builder™ und das FlexPMD-Tool verwenden.

Die Stats-Klasse

Für das Profiling Ihres Code zur Laufzeit mit der Veröffentlichungsversion der Laufzeitumgebung, ohne externes Werkzeug, können Sie die Stats-Klasse verwenden, die von mr. doob aus der Flash-Community entwickelt wurde. Sie können die Stats-Klasse von der folgenden Adresse herunterladen: https://github.com/mrdoob/Hi-ReS-Stats.

Mithilfe der Stats-Klasse können folgende Informationen ermittelt werden:

- Pro Sekunde dargestellte Bilder (je höher die Zahl, desto besser).
- Darstellungszeit eines Bildes in Millisekunden (je kleiner die Zahl, desto besser).
- Die vom Code belegte Arbeitsspeichermenge. Wenn dieser Wert bei jedem Bild zunimmt, liegt in Ihrer Anwendung möglicherweise ein Arbeitsspeicherleck vor. Die Ursache dieses potenziellen Lecks muss unbedingt untersucht werden.
- Die maximal von der Anwendung belegte Arbeitsspeichermenge.

Nach dem Herunterladen kann die Stats-Klasse zusammen mit dem folgenden Kompaktcode verwendet werden:

```
import net.hires.debug.*;
addChild( new Stats() );
```

Durch eine bedingte Kompilierung in Adobe® Flash® Professional oder Flash Builder kann das Stats-Objekt aktiviert werden:

```
CONFIG::DEBUG
{
    import net.hires.debug.*;
    addChild( new Stats() );
}
```

Sie können die Kompilierung des Stats-Objekts aktivieren oder deaktivieren, indem Sie den Wert der `DEBUG`-Konstante wechseln. Dasselbe Verfahren kann verwendet werden, um Codelogik zu ersetzen, die in der Anwendung nicht kompiliert werden soll.

Die PerformanceTest-Klasse

Für das Profiling der Ausführung von ActionScript-Code hat Grant Skinner ein Tool entwickelt, das in einen Testarbeitsablauf integriert werden kann. Sie übergeben eine benutzerdefinierte Klasse an die PerformanceTest-Klasse, die Ihren Code verschiedenen Tests unterzieht. Mithilfe der PerformanceTest-Klasse können Sie ganz einfach Vergleichswerte für verschiedene Strategien ermitteln. Die PerformanceTest-Klasse kann von folgender Website heruntergeladen werden: http://www.gskinner.com/blog/archives/2009/04/as3_performance.html.

Flash Builder-Profiler

Zum Lieferumfang von Flash Builder gehört ein Profiler, mit dem Sie Ihren Code mit hoher Detailgenauigkeit Vergleichswerten gegenüberstellen können.

Hinweis: *Verwenden Sie die Debugger-Version von Flash Player für den Zugriff auf den Profiler (andernfalls wird eine Fehlermeldung eingeblendet).*

Der Profiler kann auch mit Inhalten verwendet werden, die in Adobe Flash Professional produziert wurden. Dazu laden Sie die kompilierte SWF-Datei aus einem ActionScript- oder Flex-Projekt in Flash Builder und führen dann den Profiler für diese Datei aus. Weitere Informationen zum Profiler finden Sie unter „Profiling von Flex-Anwendungen“ in Verwenden von Flash Builder 4.

FlexPMD

Adobe Technical Services hat das Tool FlexPMD veröffentlicht, mit dem Sie die Qualität von ActionScript 3.0-Code überprüfen können. FlexPMD ist ein ActionScript-Tool, das JavaPMD ähnelt. FlexPMD verbessert die Codequalität durch Überprüfung eines ActionScript 3.0- oder Flex-Quellverzeichnisses. Es erkennt mangelhaft geschriebenen Code, wie beispielsweise nicht verwendeten, komplexen oder sehr langen Code sowie eine falsche Verwendung des Lebenszyklus von Flex-Komponenten.

FlexPMD ist ein Open-Source-Projekt von Adobe. Es ist unter der folgenden Adresse erhältlich: http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD. Unter der folgenden Adresse ist auch ein Eclipse-Plug-In verfügbar: http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD+Eclipse+plugin.

Mithilfe von FlexPDM können Sie Code ganz einfach überprüfen und sicherstellen, dass er optimal geschrieben ist. Der größte Vorteil von FlexPMD liegt in seiner Erweiterbarkeit. Entwickler können eigene Regeln zur Codeprüfung erstellen. Beispielsweise können Sie Regeln erstellen, die eine übermäßige Verwendung von Filtern oder andere unerwünschte Codestrukturen erkennen.

Bereitstellung

Beim Exportieren der endgültigen Version Ihrer Anwendung in Flash Builder müssen Sie darauf achten, dass Sie die Release-Version exportieren. Beim Exportieren der Release-Version werden die in der SWF-Datei enthaltenen Debugging-Informationen entfernt. Nachdem die Debugging-Informationen entfernt wurden, ist die SWF-Datei kleiner und die Anwendung kann schneller ausgeführt werden.

Zum Exportieren der Release-Version Ihres Projekts verwenden Sie das Projekt-Bedienfeld in Flash Builder und die Option zum Exportieren des Releasebuild.

Hinweis: *Bei der Kompilierung Ihres Projekts in Flash Professional können Sie nicht zwischen der Release-Version und der Debugging-Version wählen. Bei der kompilierten SWF-Datei handelt es sich standardmäßig um eine Release-Version.*