De prestaties voor het ADOBE® FLASH®-platform optimaliseren

Juridische kennisgeving

Laatst bijgewerkt 22/5/2012

Inhoud

Hoofdstuk 1: Inleiding

Adobe® AIR®- en Adobe® Flash®-toepassingen kunnen op vele platforms worden uitgevoerd, waaronder desktopcomputers, mobiele apparaten, tablets en televisies. In dit document worden aan de hand van code- en praktijkvoorbeelden aanbevolen procedures beschreven voor ontwikkelaars die deze toepassingen implementeren. De volgende onderwerpen worden besproken:

- Geheugen besparen
- CPU-gebruik minimaliseren
- Prestaties van ActionScript 3.0 verbeteren
- Renderingsnelheid vergroten
- Netwerkinteractie optimaliseren
- Werken met audio en video
- Prestaties van SQL-database verbeteren
- Benchmarking en implementatie van toepassingen

De meeste van deze optimalisaties zijn van toepassing op toepassingen op alle apparaten, zowel voor de AIR- als de Flash Player-runtime. Aanvullingen en uitzonderingen voor specifieke apparaten komen ook aan de orde.

Sommige van deze optimalisaties richten zich op de mogelijkheden die in Flash Player 10.1 en AIR 2.5 werden geïntroduceerd. De meeste van deze optimalisaties zijn echter ook van toepassing op eerdere versies van AIR en Flash Player.

Basisprincipes voor het uitvoeren van code in de runtime

Om te begrijpen hoe de prestaties van een toepassing verbeterd kunnen worden, moet u begrijpen hoe code in de runtime van het Flash-platform wordt uitgevoerd. Tijdens het uitvoeren van code wordt een lusbewerking gemaakt, waarin bepaalde acties in afzonderlijke "frames" worden uitgevoerd. Een frame is in dit geval gewoon een periode die wordt bepaald door de framesnelheid die voor de toepassing is opgegeven. Hoeveel tijd er voor elk frame wordt uitgetrokken is volledig afhankelijk van de framesnelheid. Als u bijvoorbeeld een framesnelheid van 30 frames per seconde opgeeft, probeert de runtime elk frame een dertigste seconde te laten duren.

De aanvankelijke framesnelheid van de toepassing wordt tijdens het ontwerpen opgegeven. U kunt de framesnelheid opgeven met instellingen in Adobe® Flash® Builder™ of Flash Professional. U kunt de aanvankelijke framesnelheid ook in code opgeven. In een toepassing met alleen ActionScript stelt u de framesnelheid in met de metatag `[SWF(frameRate="24")]` in de hoofddocumentklasse. In MXML stelt u het kenmerk `frameRate` in de tag Application of WindowedApplication in.

Elke framelusbewerking bestaat uit twee fasen en is onderverdeeld in drie onderdelen: gebeurtenissen, de gebeurtenis `enterFrame` en rendering.

De eerste fase bevat twee onderdelen (gebeurtenissen en de gebeurtenis `enterFrame`), die er allebei toe kunnen leiden dat uw code wordt aangeroepen. In het eerste onderdeel van de eerste fase arriveren runtimegebeurtenissen en worden deze verzonden. Deze gebeurtenissen kunnen een weerspiegeling zijn van de voltooiing of voortgang van asynchrone bewerkingen, zoals een antwoord op het laden van gegevens via een netwerk of gebeurtenissen naar aanleiding van invoer van de gebruiker. Wanneer gebeurtenissen worden verzonden, voert de runtime·de code uit in listeners die door u zijn geregistreerd. Als er geen gebeurtenissen plaatsvinden, wacht de runtime totdat deze fase kan worden voltooid, zonder acties uit te voeren. De framesnelheid wordt nooit door de runtime verhoogd als er geen activiteit plaatsvindt. Als er gebeurtenissen plaatsvinden tijdens de andere onderdelen van de uitvoeringscyclus, worden deze gebeurtenissen in een wachtrij geplaatst en in het volgende frame verzonden.

Het tweede onderdeel van de eerste fase is de gebeurtenis `enterFrame`. Deze gebeurtenis wijkt af van andere gebeurtenissen, omdat deze altijd slechts één keer per frame wordt verzonden.

Wanneer alle gebeurtenissen zijn verzonden, begint de renderingfase van de framelus. Op dat punt wordt de status van alle zichtbare elementen op het scherm berekend en worden deze op het scherm getekend. Vervolgens herhaalt het proces zichzelf, zoals een hardloper op een atletiekbaan.

Opmerking: *Voor gebeurtenissen die een eigenschap `updateAfterEvent` hebben, kan het zijn dat wordt afgedwongen dat de rendering meteen plaatsvindt en er dus niet wordt gewacht tot de renderingfase. Vermijd echter het gebruik van `updateAfterEvent` als dit leidt tot prestatieproblemen.*

Het zou het eenvoudigst zijn als de twee fasen in de framelusbewerking even lang duren. Dat zou betekenen dat gedurende de eerste helft van de framelusbewerking gebeurtenishandlers en toepassingscode worden uitgevoerd, en gedurende de tweede helft rendering plaatsvindt. In werkelijkheid is dit meestal niet zo. Soms neemt de toepassingscode meer dan de helft van de beschikbare tijd van het frame in beslag, meer dan er voor is toegewezen, waardoor de beschikbare tijd voor rendering afneemt. In andere gevallen, vooral bij complexe visuele inhoud zoals filters en overvloeiing, duurt de rendering meer dan de helft van de frametijd. De werkelijke tijd die door de fasen in beslag wordt genomen is dus flexibel. Daarom wordt de framelusbewerking ook wel de "elastic racetrack" (elastisch circuit) genoemd.

Als de gecombineerde bewerkingen van de framelusbewerking (uitvoering van code en rendering) te lang duren, kan de runtime de framesnelheid niet handhaven. Het frame wordt groter en duurt langer dan de toegewezen tijd, waardoor er een vertraging optreedt voordat het volgende frame wordt gestart. Als een framelusbewerking bijvoorbeeld langer dan een dertigste seconde duurt, kan de runtime het scherm niet met een frequentie van 30 frames per seconde bijwerken. Wanneer de framesnelheid afneemt, wordt de gebruikerservaring slechter. In het beste geval wordt een animatie schokkerig afgespeeld. In het slechtste geval loopt de toepassing vast en wordt een leeg scherm weergegeven.

In de volgende bronnen vindt u meer informatie over het uitvoeren van runtimecode en rendering in het Flash-platform:

- Flash Player Mental Model - The Elastic Racetrack (artikel van Ted Patrick)
- Asynchronous ActionScript Execution (artikel van Trevor McCauley)
- Adobe AIR optimaliseren voor het uitvoeren van code, geheugen en rendering op http://www.adobe.com/go/learn_fp_air_perf_tv_nl (video van de presentatie van Sean Christmann op het MAX-congres)

Waargenomen prestaties versus werkelijke prestaties

Of de prestaties van uw toepassing wel of niet goed zijn wordt uiteindelijk door de gebruikers beoordeeld. Ontwikkelaars kunnen de prestaties van een toepassing uitdrukken in de hoeveelheid tijd die nodig is voor de uitvoering van bepaalde bewerkingen, of het aantal instanties dat van objecten wordt gemaakt. Deze statistieken zijn echter niet van belang voor de eindgebruikers. Soms hanteren·gebruikers andere criteria voor prestaties. Bijvoorbeeld: werkt de toepassing snel en naadloos en reageert deze snel op invoer? Heeft de toepassing negatieve gevolgen voor de prestaties van het systeem? Door de volgende vragen te beantwoorden kunt u de waargenomen·prestaties testen:

- Zijn animaties vloeiend of schokkerig?
- Ziet video-inhoud er vloeiend of schokkerig uit?
- Worden geluidsfragmenten zonder onderbrekingen afgespeeld?
- Hapert het scherm of wordt er een leeg scherm weergegeven tijdens lange bewerkingen?
- Wordt tijdens het typen·de tekst meteen op het scherm weergegeven of loopt deze achter?
- Gebeurt er na een muisklik meteen iets, of is er sprake van een vertraging?
- Maakt de ventilator van de CPU meer geluid wanneer de toepassing wordt uitgevoerd?
- Raakt de accu van een laptop of mobiel apparaat sneller leeg wanneer de toepassing hierop wordt uitgevoerd?
- Reageren andere toepassingen slecht wanneer de toepassing wordt uitgevoerd?

Het verschil tussen waargenomen prestaties en werkelijke prestaties is belangrijk. Het streven naar de beste waargenomen prestaties komt niet altijd overeen met het streven naar de absoluut snelste prestaties. Zorg ervoor dat de toepassing nooit zoveel code uitvoert dat de runtime·niet langer in staat is om het scherm bij te werken en invoer van de gebruiker te verzamelen. In sommige gevallen bereikt u deze balans door een programmataak te splitsen, zodat het scherm tussendoor kan worden bijgewerkt. (Zie “Renderprestaties” op pagina 49 voor meer informatie.)

De hier beschreven tips en technieken streven naar een verbetering van zowel de daadwerkelijke uitvoering van code als de manier waarop gebruikers de prestaties ervaren.

Het doel van optimalisatie

Sommige prestatieverbeteringen leveren geen merkbare verbetering voor gebruikers op. Het is belangrijk dat u zich bij het optimaliseren van de prestaties richt op gebieden die problematisch zijn voor uw specifieke toepassing. Sommige optimalisaties horen bij het goed ontwikkelen van toepassingen en kunnen altijd worden doorgevoerd. Voor andere optimalisaties geldt dat het nut ervan afhankelijk is van de behoeften van de toepassing en het·verwachte gebruikerspubliek. Een toepassing presteert bijvoorbeeld altijd beter als deze geen animaties, video of grafische filters en effecten bevat. Maar een belangrijke reden waarom u het Flash-platform gebruikt zijn natuurlijk de media- en grafische functies waarmee u uitgebreide·en expressieve toepassingen kunt maken. Bepaal in hoeverre een uitgebreide toepassing overeenkomt met de prestatiekenmerken van de computers en apparaten waarop de toepassing zal worden uitgevoerd.

Het is raadzaam om niet te vroeg te beginnen met optimaliseren. Voor sommige optimalisaties wordt code gebruikt die moeilijk leesbaar of minder flexibel is. Dergelijke code is na optimalisatie moeilijker te beheren. Voor deze·optimalisaties geldt dat u beter kunt wachten totdat u kunt bepalen of een bepaald gedeelte van de code slecht presteert voordat u ervoor kiest deze te optimaliseren.

Bij het verbeteren van prestaties moet u soms concessies doen. Onder ideale omstandigheden zou het natuurlijk zo zijn dat een vermindering van de hoeveelheid geheugen die een toepassing gebruikt ertoe leidt dat de toepassing sneller taken uitvoert. Dit ideale type verbetering is echter niet altijd mogelijk. Als een toepassing bijvoorbeeld tijdens een bewerking vastloopt, moet u dit vaak oplossen door taken over verschillende frames te verdelen. Als het werk wordt opgesplitst, duurt het meestal langer om het proces te voltooien. Het kan echter zijn dat de gebruiker deze extra tijd niet opmerkt als de toepassing blijft reageren op invoer en niet vastloopt.

Om te weten wat u moet optimaliseren en of optimalisaties nut hebben, moet u de prestaties testen. In "Benchmarking en implementatie" op pagina 97 worden verschillende technieken en tips voor het testen·van prestaties beschreven.

Raadpleeg de volgende bronnen voor meer informatie over hoe u kunt bepalen·welke onderdelen van een toepassing geschikt zijn voor optimalisatie:

- Prestaties van toepassingen afstemmen voor AIR op http://www.adobe.com/go/learn_fp_goldman_tv_nl(video van de presentatie van Oliver Goldman op het MAX-congres)
- Prestaties van Adobe AIR-toepassingen afstemmen http://www.adobe.com/go/learn_fp_air_perf_devnet_nl(Adobe Developer Connection-artikel van Oliver Goldman, gebaseerd op zijn presentatie)

Hoofdstuk 2: Geheugen besparen

Het gebruik van zo weinig mogelijk geheugen is altijd belangrijk bij de ontwikkeling van toepassingen, zelfs voor bureaubladtoepassingen. Op mobiele apparatuur krijgt geheugenverbruik echter de prioriteit en daarom is het raadzaam de hoeveelheid geheugen die uw toepassing verbruikt te beperken.

Weergaveobjecten

Kies een toepasselijk weergaveobject.

ActionScript 3.0 bevat een groot aantal weergaveobjecten. Een van de meest eenvoudige optimalisatietips voor het beperken van geheugengebruik is het gebruik van het juiste weergaveobject. Gebruik de Shape-objecten voor eenvoudige niet-interactieve vormen. Gebruik de Sprite-objecten voor interactieve objecten waarvoor geen tijdlijn nodig is. Gebruik MovieClip-objecten voor animatie met een tijdlijn. Kies altijd het meest efficiënte objecttype voor uw toepassing.

In de volgende code wordt het geheugengebruik voor verschillende objecten weergegeven:

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

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

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

De `getSize()`-methode geeft weer hoeveel bytes aan geheugen een object verbruikt. U ziet dat het gebruik van meerdere MovieClip-objecten in plaats van eenvoudige Shape-objecten geheugen verspilt als de functies van een MovieClip-object overbodig zijn.

Primitieve typen

Gebruik de `getSize()`-methode om code te definiëren en het meest efficiënte object voor de taak te bepalen.

Alle primitieve typen, behalve String, gebruiken 4 - 8 bytes geheugen. U kunt geheugen niet optimaliseren door een specifiek primitief type te gebruiken.

```
// 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
```

Als aan een nummer dat een 64-bits waarde vertegenwoordigt geen waarde wordt toegewezen, wijst AVM (ActionScript Virtual Machine) er acht bytes aan toe. Alle andere primitieve typen worden in 4 bytes opgeslagen.

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

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

Het gedrag verschilt per String-type. De toegewezen opslaghoeveelheid wordt gebaseerd op de lengte van de String.

```
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
```

Gebruik de `getSize()`-methode om code te definiëren en het meest efficiënte object voor de taak te bepalen.

Objecten opnieuw gebruiken


Gebruik objecten indien mogelijk opnieuw, in plaats van ze opnieuw te maken.

Een andere eenvoudige manier om geheugen te optimaliseren, is het hergebruiken van objecten en ze alleen opnieuw te maken als dit onvermijdelijk is. Gebruik in een loop bijvoorbeeld de volgende code niet:

```
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);
}
```

Het opnieuw maken van het Rectangle-object in elke loopherhaling verbruikt meer geheugen en neemt meer tijd, omdat in elke herhaling een nieuw object wordt gemaakt. Gebruik de volgende aanpak:

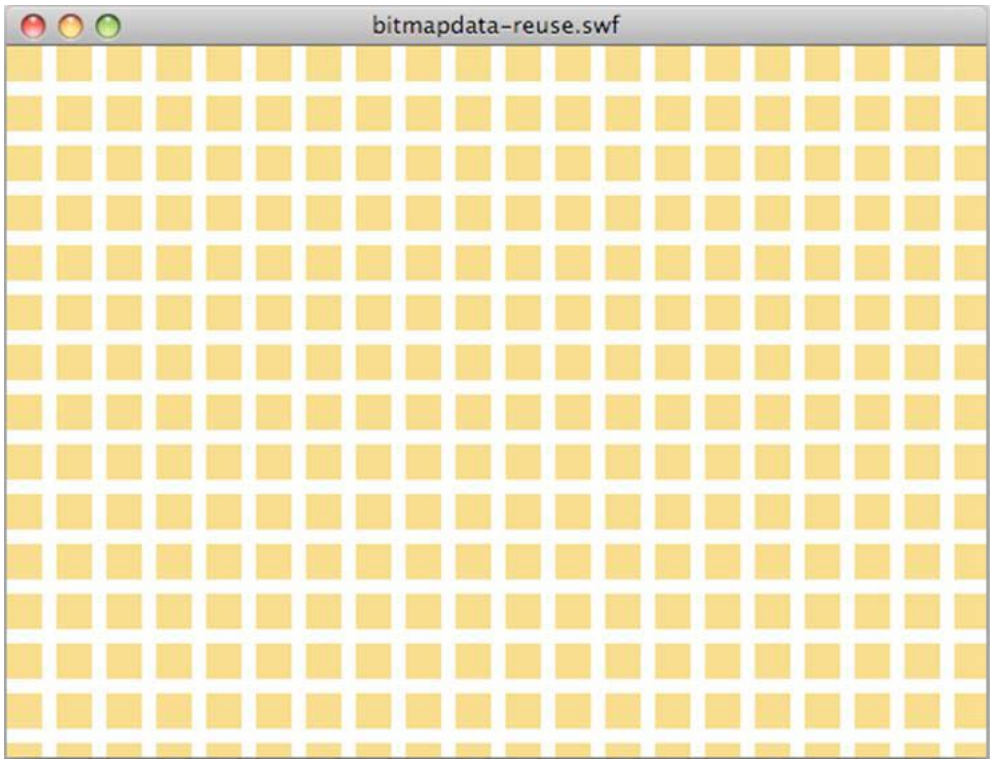
```
const MAX_NUM:int = 18;
const COLOR:uint = 0xCCCCCC;

// 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);
}
```

Het vorige voorbeeld gebruikt een object met een relatief klein geheugengebruik. In het volgende voorbeeld wordt weergegeven hoe u meer geheugen kunt besparen door een BitmapData-object opnieuw te gebruiken. De volgende code voor het maken van een tegeleffect verspilt geheugen:

```
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);
}
```

Opmerking: *Wanneer u positieve waarden gebruikt, is het veel sneller om de afgeronde waarde naar int te sturen dan om de* `Math.floor()`*-methode te gebruiken.*

In de volgende afbeelding ziet u het resultaat van het tegeleffect voor een bitmap:



Resultaat van het tegeleffect voor een bitmap

Een geoptimaliseerde versie maakt één BitmapData-instantie, waarnaar door meerdere Bitmap-instanties verwezen wordt en die hetzelfde resultaat heeft:

```
// 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);
}
```

Deze aanpak bespaart ongeveer 700 kB geheugen; een belangrijke besparing op een traditioneel mobiel apparaat. Elke bitmapcontainer kan met behulp van de Bitmap-eigenschappen worden aangepast, zonder dat de oorspronkelijke BitmapData-instantie wordt gewijzigd.

```
// 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();
}
```

In de volgende afbeelding ziet u het resultaat van de bitmaptransformaties:

Resultaat van de bitmaptransformaties

Meer Help-onderwerpen

"Het in cache plaatsen van bitmaps" op pagina 55

Objectpooling

Gebruik indien mogelijk objectpooling.

Een andere belangrijke optimalisatie is objectpooling, waarbij objecten worden hergebruikt. U maakt een bepaald aantal objecten tijdens de initialisatie van uw toepassing en slaat deze op in een pool, zoals een Array- of Vector-object. Wanneer u een object niet meer gebruikt, deactiveert u dit, zodat het geen CPU-bronnen meer verbruikt, en verwijdert u alle wederzijdse referenties. U stelt de verwijzingen echter niet in op `null`, anders zouden ze in aanmerking komen voor afvalophaling. U plaatst het object gewoon terug in de pool en haalt het op wanneer u een nieuw object nodig hebt.

Hergebruik van objecten betekent dat u objecten niet zo vaak hoeft te instantiëren, hetgeen een besparing oplevert. Het beperkt ook de kans dat de afvalophaling wordt uitgevoerd, die uw toepassing kan vertragen. De volgende code illustreert de techniek voor objectpooling:

```
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;
        }
    }
}
```

De SpritePool-klasse maakt een pool met nieuwe objecten tijdens de initialisatie van de toepassing. De `getSprite()`-methode retourneert instanties van deze objecten en ze worden door de `disposeSprite()`-methode vrijgegeven. Met de code kan de pool groeien, wanneer die volledig is gebruikt. U kunt ook een pool maken met een vaste omvang, waaraan nieuwe objecten niet worden toegewezen als de pool uitgeput is. Probeer indien mogelijk het maken van nieuwe objecten in loops te vermijden. Zie voor meer informatie “Geheugen vrijmaken” op pagina 12. De volgende code gebruikt de SpritePool-klasse om nieuwe instanties op te halen:

```
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 );
    }
}
```

De volgende code verwijdert alle weergaveobjecten uit de weergavelijst wanneer er met de muis wordt geklikt en gebruikt ze later opnieuw voor een andere taak.

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

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

Opmerking: *De poolvector verwijst altijd naar de Sprite-objecten. Als u het object volledig uit het geheugen wilt verwijderen, hebt u een* `dispose()`*-methode nodig voor de SpritePool-klasse, die alle overgebleven referenties verwijdert.*

Geheugen vrijmaken

Verwijder alle referenties naar objecten om ervoor te zorgen, dat de afvalophaling wordt gestart.

U kunt de afvalophaling niet rechtstreeks starten in de releaseversie van Flash Player. Verwijder alle referenties naar het object om ervoor te zorgen dat er een afvalophaling wordt gestart wanneer u een object verwijdert. Vergeet niet dat de oude `delete`-operator die in ActionScript 1.0 en 2.0 wordt gebruikt anders functioneert in ActionScript 3.0. Deze kan alleen worden gebruikt voor verwijdering van de dynamische eigenschappen van een dynamisch object.

Opmerking: *U kunt afvalophaling rechtstreeks starten in Adobe® AIR® en in de foutopsporingsversie van Flash Player.*

De volgende code stelt een Sprite-referentie bijvoorbeeld in op `null`.

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

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

Onthoud, dat wanneer een object op `null` wordt ingesteld, deze niet noodzakelijk uit het geheugen wordt verwijderd. Soms wordt de afvalophaler niet uitgevoerd, bijvoorbeeld als het beschikbare geheugen als niet laag genoeg wordt beschouwd. Afvalophaling is onvoorspelbaar. Afvalophaling wordt geactiveerd door geheugentoewijzing en niet door het verwijderen van objecten. Wanneer de afvalophaling wordt uitgevoerd, vindt deze afbeeldingen van objecten die nog niet zijn opgehaald. Het detecteert inactieve objecten in de afbeeldingen, doordat deze objecten zoekt die naar elkaar verwijzen, maar die niet meer worden gebruikt door de toepassing. Inactieve objecten die op deze manier worden gedetecteerd worden verwijderd.

In grote toepassingen kan dit proces CPU-intensief zijn en kan dit de prestatie beïnvloeden en een merkbare vertraging van de toepassing veroorzaken. Probeer afvalophaling te beperken, door objecten zoveel mogelijk opnieuw te gebruiken. Stel referenties zoveel mogelijk in op null, zodat de afvalophaler minder lang bezig is met het zoeken naar objecten. Beschouw de afvalophaling als een verzekering en beheer de levensduur van objecten indien mogelijk per geval.

Opmerking: *Het instellen van een referentie naar een weergaveobject op null garandeert niet dat het object wordt vastgezet. Het object blijft CPU-cycli consumeren totdat de afvalophaling is voltooid. Zorg ervoor dat u het object op de juiste manier deactiveert voordat u de referentie naar het object instelt op* `null`.

De afvalophaler kan gestart worden met behulp van de `System.gc()`-methode, die beschikbaar is in Adobe AIR en in de foutopsporingsversie van Flash Player. Met de profiler die bij Adobe® Flash® Builder wordt geleverd, kunt u de afvalophaling handmatig opstarten. Als u de afvalophaler start, kunt u zien hoe uw toepassing reageert en of objecten correct uit het geheugen kunnen worden verwijderd.

Opmerking: *Als een object als een gebeurtenislistener werd gebruikt, kan een ander object ernaar verwijzen. Als dat het geval is, verwijdert u gebeurtenislisteners die de methode* `removeEventListener()` *gebruiken voordat u de referenties instelt op* `null`.

Gelukkig kan het door bitmaps gebruikte geheugen onmiddellijk worden gereduceerd. De BitmapData-klasse bevat bijvoorbeeld een `dispose()`-methode. Het volgende voorbeeld maakt een BitmapData-instantie van 1.8 MB. Het huidige geheugen in gebruik groeit tot 1,8 MB en de eigenschap `System.totalMemory` retourneert een kleinere waarde:

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

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

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

Vervolgens wordt de BitmapData handmatig uit het geheugen verwijderd en wordt het geheugengebruik opnieuw gecontroleerd:

```
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
```

Hoewel de `dispose()`-methode de pixels uit het geheugen verwijdert, moet de referentie nog steeds op `null` worden ingesteld om deze volledig te verwijderen. Roep altijd de `dispose()`-methode op en stel de referentie in op `null`, wanneer u geen BitmapData-object meer nodig hebt, zodat er direct geheugen vrijkomt.

Opmerking: *Flash Player 10.1 en AIR 1.5.2 introduceren een nieuwe methode, `disposeXML()`, in de System-klasse. Met deze methode maakt u een XML-object dat meteen beschikbaar is voor afvalophaling door de XML-structuur door te geven als een parameter.*

Meer Help-onderwerpen

"Objecten vastzetten en vrijgeven" op pagina 28

Bitmaps gebruiken

U kunt veel geheugen besparen door vectors te gebruiken in plaats van bitmaps. Het gebruik van vectors, vooral in grote aantallen, vergroot echter aanzienlijk de behoefte aan CPU- of GPU-bronnen. Het gebruik van bitmaps is een goede manier om rendering te optimaliseren, omdat de runtime minder verwerkingsbronnen nodig heeft om pixels op het scherm te tekenen dan om vectorinhoud te renderen.

Meer Help-onderwerpen

"Het handmatig in cache plaatsen van bitmaps" op pagina 63

Het downsampelen van bitmaps

Om beter gebruik te maken van het geheugen, worden 32-bits ondoorzichtige afbeeldingen verkleind tot 16-bits afbeeldingen, wanneer Flash Player een 16-bits scherm detecteert. Dit downsampelen verbruikt de helft van de geheugenmiddelen en rendeert afbeeldingen sneller. Deze functie is alleen beschikbaar in Flash Player 10.1 voor Windows Mobile.

Opmerking: *Voor Flash Player 10.1 werden alle gemaakte pixels in het geheugen als 32-bits (4 bytes) opgeslagen. Een eenvoudig logo van 300 x 300 pixels verbruikte 350 kb geheugen (300*300*4/1024). Met dit nieuwe gedrag verbruikt hetzelfde ondoorzichtige logo slechts 175 kb. Als het logo transparant is, wordt het niet gedownsampeld tot 16 bits en behoudt het dezelfde grootte in het geheugen. Deze functie is alleen van toepassing op ingesloten bitmaps of bij uitvoering geladen afbeeldingen (.png, .gif. jpg).*

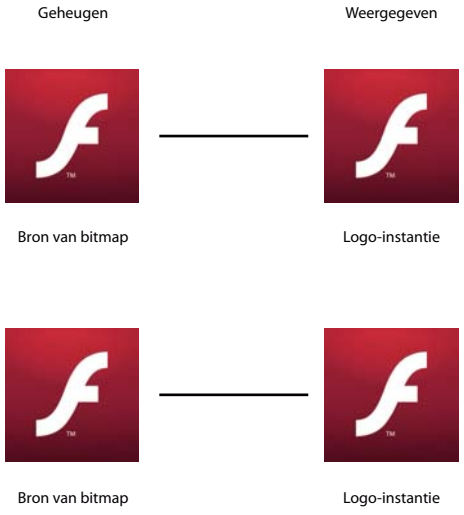
Op mobiele apparaten kan het moeilijk zijn om het verschil te zien of een afbeelding in 16 bits of 32 bits wordt gerenderd. Er is geen merkbaar verschil bij een eenvoudige afbeelding die slechts een paar kleuren bevat. Zelfs bij een complexere afbeelding is het moeilijk om de verschillen te vinden. Er kan echter een kleurdegradatie voorkomen, wanneer u op een afbeelding inzoomt en een 16 bits-verloop kan er minder vloeiend uitzien dan de 32 bits-versie.

Een enkele BitmapData-referentie

Het is belangrijk het gebruik van de BitmapData-klasse te optimaliseren door instanties zoveel mogelijk opnieuw te gebruiken. In Flash Player 10.1 en AIR 2.5 wordt een nieuwe functie geïntroduceerd voor alle platforms: de enkele BitmapData-referentie. Wanneer u BitmapData-instanties maakt van een ingesloten afbeelding, wordt één versie van de bitmap gebruikt voor alle BitmapData-instanties. Als een bitmap later wordt aangepast, krijgt deze een eigen unieke bitmap in het geheugen. De ingesloten afbeelding kan afkomstig zijn uit de bibliotheek of uit een [Embed]-tag.

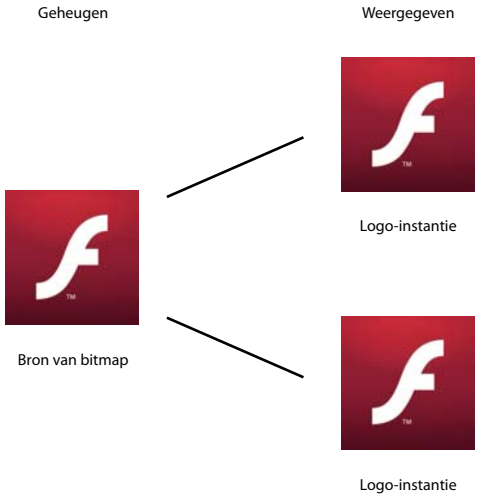
Opmerking: *De bestaande inhoud profiteert ook van deze nieuwe functie, omdat Flash Player 10.1 en AIR 2.5 deze bitmaps automatisch opnieuw gebruiken.*

Wanneer u een ingesloten afbeelding instantieert, wordt een bijbehorende bitmap gemaakt in het geheugen. Vóór Flash Player 10.1 en AIR 2.5 kreeg elke instantie een aparte bitmap in het geheugen, zoals wordt weergegeven in het volgende diagram:



Bitmaps in het geheugen vóór Flash Player 10.1 en AIR 2.5

Wanneer er in Flash Player 10.1 en AIR 2.5 meerdere instanties van dezelfde afbeelding worden gemaakt, wordt er een enkele versie van de bitmap gebruikt voor alle BitmapData-instanties. het volgende diagram illustreert dit concept:

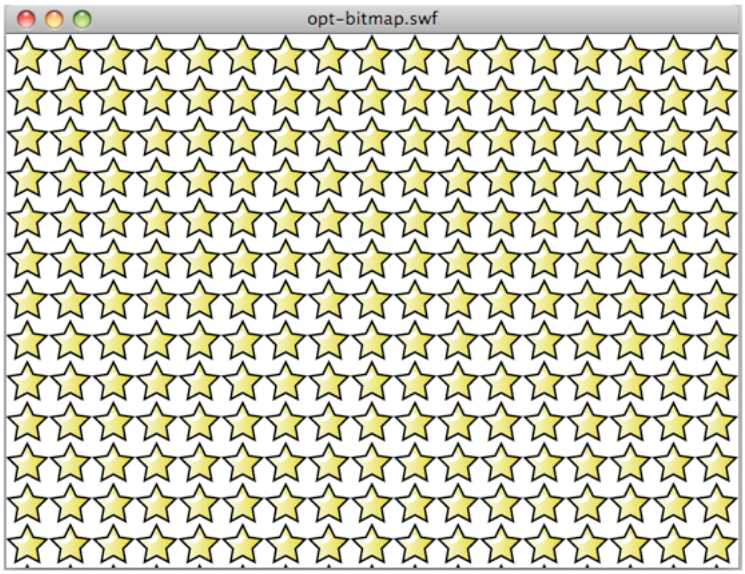


Bitmaps in het geheugen van Flash Player 10.1 en AIR 2.5

Deze aanpak vermindert aanzienlijk de hoeveelheid geheugen die een toepassing met veel bitmaps gebruikt. De volgende code maakt meerdere instanties van een `Star`-symbool:

```
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)
    }
}
```

In de volgende afbeelding ziet u het resultaat van de code:



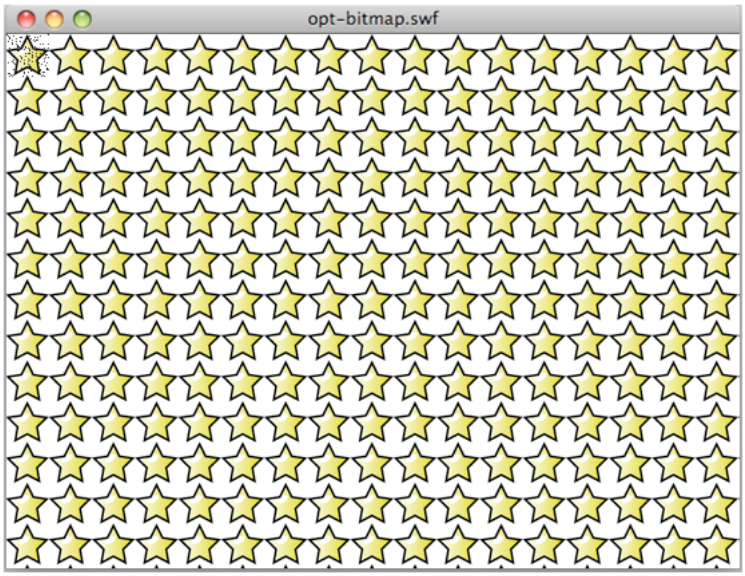
Resultaat van code om meerdere instanties van een symbool te maken.

Met Flash Player 10 gebruikt de bovenstaande animatie bijvoorbeeld ongeveer 1008 kB geheugen. Met Flash Player 10.1 gebruikt de animatie op een desktopcomputer en op een mobiel apparaat slechts 4 kB.

De volgende code past één BitmapData-instantie aan:

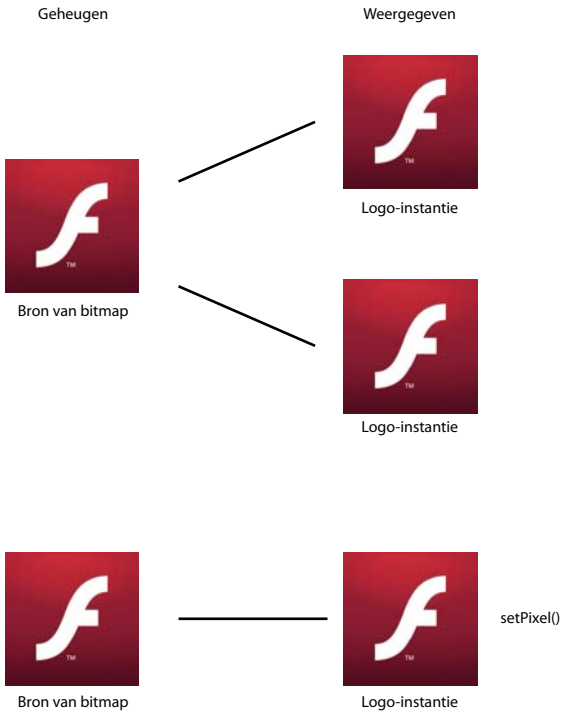
```
const MAX_NUM:int = 18;

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);
```

In de volgende afbeelding ziet u het resultaat na het aanpassen van één `Star`-instantie:



Resultaat na het aanpassen van één instantie.

Intern wijst de runtime automatisch een bitmap aan het geheugen toe of maakt een bitmap in het geheugen om de pixelwijzigingen te verwerken. Wanneer een methode van de BitmapData-klasse wordt aangeroepen die tot pixelwijzigingen leidt, wordt er een nieuwe instantie gemaakt in het geheugen en worden er geen andere instanties bijgewerkt. De onderstaande afbeelding illustreert het concept:



Resultaat in het geheugen van het aanpassen van één bitmap.

Als er één ster wordt gewijzigd, wordt er een nieuwe kopie in het geheugen gemaakt. De animatie die daarvan het resultaat is, gebruikt ongeveer 8 kB geheugen met Flash Player 10.1 en AIR 2.5.

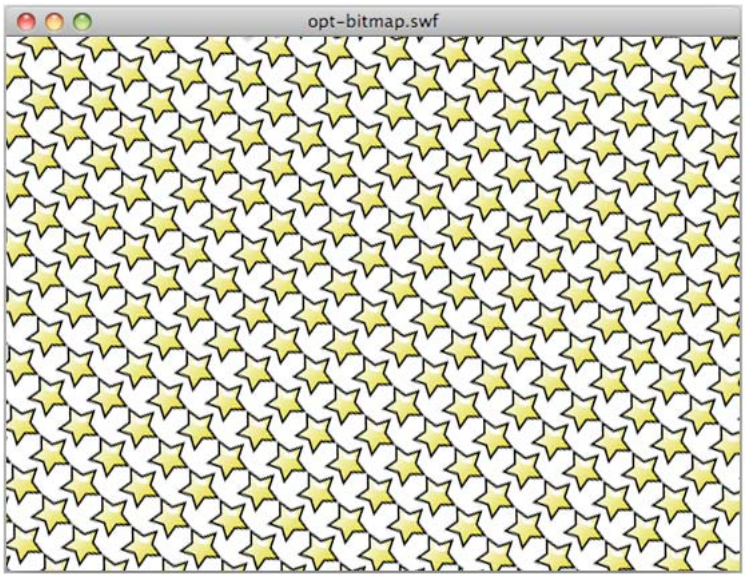
In het vorige voorbeeld is elke bitmap afzonderlijk beschikbaar voor transformatie. De `beginBitmapFill()`-methode is het geschiktst als u alleen een naast-elkaareffect wilt bereiken.

```
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);
```

Deze aanpak produceert hetzelfde resultaat met slechts één BitmapData-instantie. Als u sterren voortdurend wilt roteren in plaats van elke sterinstantie afzonderlijk te openen, gebruikt u een Matrix-object dat op elk frame wordt geroteerd. Geef dit Matrix- object door aan de methode `beginBitmapFill()`:

```
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);
}
```

Met deze techniek hebt u geen ActionScript-loop nodig om het effect te bereiken. De runtime doet alles intern. In de volgende afbeelding ziet u het resultaat van de transformatie van de sterren:



Resultaat van het draaien van sterren.

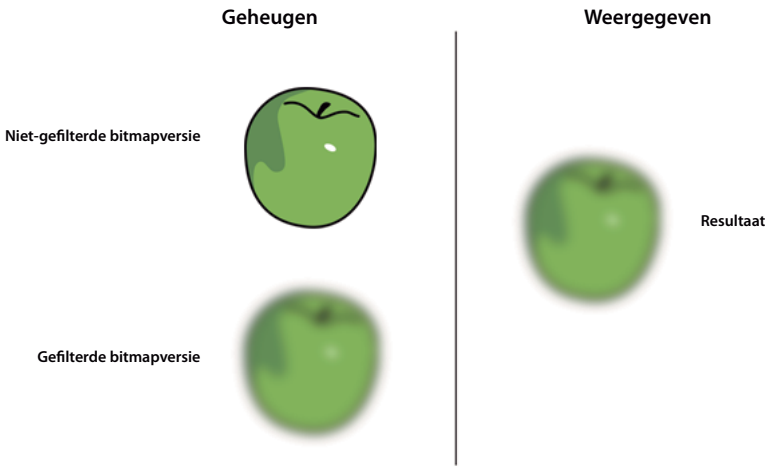
Met deze aanpak wordt tijdens het bijwerken van de oorspronkelijke bron, het BitmapData-object, zijn gebruik op een andere plek in het werkgebied automatisch bijgewerkt. Dit is een krachtige techniek. Deze aanpak staat echter niet toe dat de schaal van elke ster afzonderlijk wordt aangepast, zoals in het vorige voorbeeld.

Opmerking: *Wanneer u meerdere instanties van dezelfde afbeelding gebruikt, hangt het tekenen af van het feit of er een klasse aan de oorspronkelijke bitmap in het geheugen is gekoppeld. Als er geen klasse aan de bitmap is gekoppeld, worden afbeeldingen getekend als Shape-objecten met bitmapinvullingen.*

Filters en dynamische bitmapontladingen

Vermijd het gebruik van filters, inclusief filters die verwerkt worden met Pixel Bender.

Minimaliseer het gebruik van effecten zoals filters, inclusief filters die in mobiele apparaten worden verwerkt via Pixel Bender. Wanneer er een filter op een weergaveobject wordt toegepast, maakt de runtime twee bitmaps in het geheugen. Deze bitmaps hebben elk de grootte van het weergaveobject. De eerste bitmap wordt gemaakt als een gerasterde versie van het weergaveobject, dat zelf wordt gebruikt om een tweede bitmap met toegepaste filter te maken:



Twee bitmaps in het geheugen als er een filter wordt toegepast.

Wanneer u een van de eigenschappen van een filter aanpast, worden beide bitmaps in het geheugen bijgewerkt, zodat de resulterende bitmap ontstaat. Dit proces omvat CPU-verwerking en de twee bitmaps kunnen een grote hoeveelheid geheugen gebruiken.

In Flash Player 10.1 en AIR 2.5 wordt een nieuwe manier van filteren geïntroduceerd op alle platforms. Als de filter niet binnen 30 seconden wordt aangepast of als deze verborgen of offscreen is, komt het geheugen dat door de ongefilterde bitmap werd gebruikt vrij.

Deze functie bespaart de helft van het geheugen dat door een filter op alle platforms wordt gebruikt. Bekijk eens een tekstobject waarop een vervagingsfilter is toegepast. De tekst wordt in dit geval gebruikt voor eenvoudige decoratie en wordt niet aangepast. Na 30 seconden, komt de ongefilterde bitmap in he geheugen vrij. U verkrijgt hetzelfde effect als de tekst gedurende 30 seconden verborgen of offscreen is. Wanneer een van de filtereigenschappen wordt aangepast, wordt de ongefilterde bitmap in het geheugen opnieuw gemaakt. Deze functie wordt ook wel dynamisch bitmapontladen genoemd. Zelfs met deze optimalisaties moet u oppassen met het gebruik van filters; deze vereisen nog steeds uitgebreide CPU- of GPU-verwerking wanneer ze worden aangepast.

U kunt het beste bitmaps gebruiken die met een ontwerpprogramma zoals Adobe® Photoshop® zijn gemaakt om waar mogelijk filters te emuleren. Vermijd het gebruik van dynamische bitmaps die bij uitvoering zijn gemaakt in ActionScript. Het gebruik van extern ontworpen bitmaps helpt Flash Player om de CPU- of GPU-belasting te verminderen, vooral als de filtereigenschappen niet veranderen. Maak indien mogelijk alle vereiste bitmapeffecten in een ontwerpprogramma. U kunt de bitmap dan weergeven in de runtime zonder deze verder te hoeven bewerken en dat kan veel sneller gaan.

Direct mipmappen

Gebruik mipmapping als u grote afbeeldingen moet schalen.

Een andere nieuwe functie die beschikbaar is in Flash Player 10.1 en AIR 2.5 op alle platforms, heeft betrekking op mipmapping. In Flash Player 9 en AIR 1.0 werd een mipmappingfunctie geïntroduceerd waarmee de kwaliteit en prestaties van verkleinde bitmaps werd verbeterd.

Opmerking: *De mipmapfunctie is alleen van toepassing op dynamisch geladen afbeeldingen of ingesloten bitmaps. Mipmapping is niet alleen van toepassing op weergaveobjecten die zijn gefilterd of in cache zijn opgeslagen. Mipmapping kan alleen worden verwerkt als de bitmap een even breedte en hoogte heeft. Wanneer een breedte of hoogte oneven is, wordt de mipmapping stopgezet. Een afbeelding van 250 x 250 kan bijvoorbeeld maximaal worden gemipmapt tot 125 x 125. In dat geval is minstens één van de afmetingen een oneven getal. Bitmaps met afmetingen die getallen in kwadraat zijn leveren de beste resultaten op, zoals 256 x 256, 512 x 512, 1024 x 1024, enz.*

Stel bijvoorbeeld dat er een afbeelding van 1024 x 1024 geladen is en dat een ontwikkelaar de afbeelding wil schalen om een miniatuur in een galerij te maken. De mipmappingfunctie rendert de afbeelding correct, als deze geschaald is met de gedownsampelde tussenversies van de bitmap als texturen. In vorige versies van de runtime werden verkleinde tussenversies van de bitmap in het geheugen gemaakt. Als er een afbeelding van 1024 x 1024 werd geladen als een afbeelding van 64 x 64 werd weergegeven, maakten oudere versies van de runtime elke halve bitmap. In dit geval zouden er bijvoorbeeld 512 x 512, 256 x 256, 128 x 128 en 64 x 64 bitmaps worden gemaakt.

Flash Player 10.1 en AIR 2.5 ondersteunen nu mipmapping rechtstreeks vanuit de oorspronkelijke bron naar de vereiste doelomvang. In het vorige voorbeeld zouden alleen de 4 MB (1024 x 1024) originele bitmap en de 16 KB (64 x 64) gemipmapte bitmap worden gemaakt.

De mipmappinglogica werkt ook met de ontlaadfunctie voor dynamische bitmaps: Als er alleen een 64 x 64 bitmap wordt gebruikt, wordt de 4 MB oorspronkelijke bitmap uit het geheugen vrijgegeven. Als de mipmap opnieuw moet worden gemaakt, wordt het origineel opnieuw geladen. Als andere gemipmapte bitmaps van verschillende grootten vereist zijn, wordt de mipmapketen van bitmaps gebruikt om de bitmap te maken. Als er bijvoorbeeld een 1:8-bitmap moet worden gemaakt, worden de 1:4-, 1:2- en 1:1-bitmaps gecontroleerd om te bepalen welke eerst in het geheugen wordt geladen. Als er geen andere versies zijn gevonden, wordt de originele 1:1-bitmap vanaf de bron geladen en gebruikt.

De JPEG-decompressor kan binnen het eigen formaat mipmapping uitvoeren. Met dit directe mipmappen kan een grote bitmap direct worden gedecromprimeerd tot een mipmapindeling, zonder dat de volledige ongecomprimeerde afbeelding wordt geladen. Het generen van de mipmap is aanmerkelijk sneller en het geheugen dat door grote bitmaps wordt gebruikt, wordt niet eerst toegewezen en vervolgens vrijgegeven. De kwaliteit van de JPEG-afbeelding is vergelijkbaar met de algemene mipmappingtechniek.

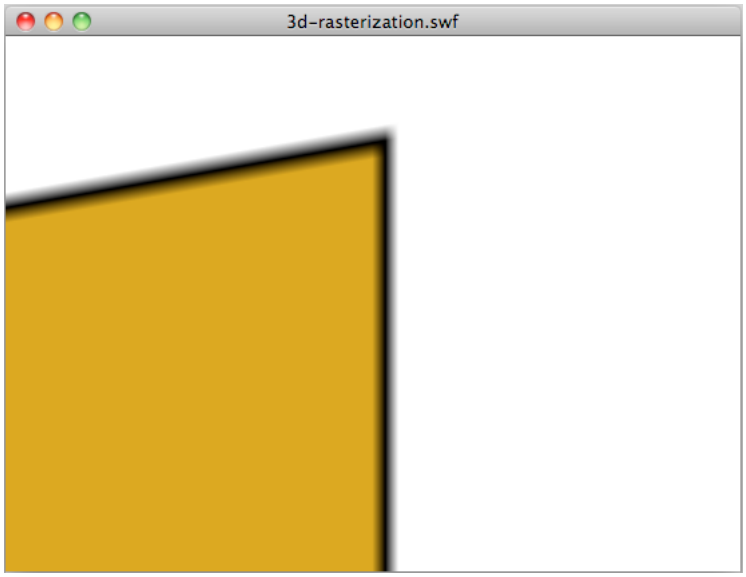
Opmerking: *Maak spaarzaam gebruik van mipmapping. Mipmapping verbetert dan wel de kwaliteit van in schaal verkleinde bitmaps, maar heeft gevolgen voor de bandbreedte, het geheugen en de snelheid. Soms kunt u beter een bitmapversie importeren die in een extern programma is geschaald. Begin niet met grote bitmaps als u van plan bent deze te verkleinen.*

3D-effecten gebruiken

U kunt overwegen handmatig 3D-effecten tot stand te brengen.

In Flash Player 10 en AIR 1.5 werd een 3D-engine geïntroduceerd waarmee u perspectieftransformatie op weergaveobjecten kunt toepassen. U kunt deze transformaties toepassen met behulp van de eigenschappen `rotationX` en `rotationY` of met de `drawTriangles()`-methode van de Graphics-klasse. U kunt dan ook diepte toepassen met de `z`-eigenschap. Onthoud, dat elk met een perspectief getransformeerd weergaveobject als een bitmap gerasterd is en daarom meer geheugen vereist.

In de volgende afbeelding ziet u de anti-aliasing die door de rastering tijdens de perspectieftransformatie wordt gemaakt.



Anti-aliasing door perspectieftransformatie.

Antialiasing treedt op wanneer vectorinhoud dynamisch wordt gerasterd als een bitmap. Deze anti-aliasing doet zich voor wanneer u 3D-effecten gebruikt in de desktopversie van AIR en Flash Player en in AIR 2.0.1 en AIR 2.5 voor mobiele apparaten. Anti-aliasing wordt echter niet toegepast op Flash Player voor mobiele apparaten.

Als u handmatig een 3D-effect kunt maken zonder op een eigen API te vertrouwen, kan dit het geheugengebruik verminderen. De nieuwe 3D-functies die in Flash Player 10 en AIR 1.5 werden geïntroduceerd, maken textuurmapping eenvoudiger, dankzij methoden zoals `drawTriangles()` die textuurmapping zelf verwerkt.

Als ontwikkelaar bepaalt u of het 3D-effect dat u wilt maken een betere prestatie levert als deze via de native API of handmatig wordt verwerkt. Houd rekening met de uitvoerings- en renderingsprestatie en het geheugengebruik van ActionScript.

In mobiele toepassingen van AIR 2.0.1 en AIR 2.5, waarin u de toepassingseigenschap `renderMode` op `GPU` instelt, voert de GPU de 3D-transformaties uit. Als de eigenschap `renderMode` is ingesteld op `CPU`, voert de CPU en niet de GPU de 3D-transformaties uit. In Flash Player 10.1-toepassingen voert de CPU de 3D-transformaties uit.

Wanneer de CPU de 3D-transformaties uitvoert, moet u er rekening mee houden dat voor het toepassen van 3D-transformatie op een weergaveobjectect, twee bitmaps in het geheugen zijn vereist. Eén bitmap is bedoeld voor de bronbitmap en een tweede bitmap voor de versie met het gewijzigde perspectief. 3D-transformaties werken in dit opzicht net als filters. Maak daarom spaarzaam gebruik van 3D-eigenschappen wanneer de CPU de 3D-transformaties uitvoert.

Tekstobjecten en geheugen

Gebruik de Adobe® Flash®-tekstengine voor tekst die alleen kan worden gelezen; gebruik TextField-objecten voor invoertekst.

In Flash Player 10 en AIR 1.5 werd een krachtige, nieuwe tekstengine geïntroduceerd, de Adobe Flash-tekstengine (FTE), die minder systeemgeheugen gebruikt. De FTE is echter een API van laag niveau waarvoor een extra aanvullende ActionScript 3.0-laag is vereist die in het flash.text.engine-pakket is opgenomen.

Voor alleen-lezen tekst kunt u beter de Flash Text Engine gebruiken, die minder geheugen gebruikt en betere rendering biedt. Voor invoertekst kunt u beter TextField-objecten gebruiken, omdat deze minder ActionScript-code gebruiken om typisch gedrag te maken, zoals invoerbeheer en tekstomloop.

Meer Help-onderwerpen

"Tekstobjecten renderen" op pagina 69

Gebeurtenismodel en callbacks

U kunt eenvoudige callbacks gebruiken in plaats van het gebeurtenismodel.

Het ActionScript 3.0-gebeurtenismodel is gebaseerd op het verzenden van objecten. Het gebeurtenismodel is objectgericht en geoptimaliseerd voor hergebruik van code. De methode `dispatchEvent()` herhaalt de lijst met listeners en roept de gebeurtenishandlermethode aan voor elk geregistreerd object. Een van de nadelen van het gebeurtenismodel is echter dat u vele objecten moet maken tijdens de levensduur van uw toepassing.

Stel u voor dat u een gebeurtenis moet verzenden vanaf de tijdlijn om het einde van een animatiereeks aan te geven. U kunt dit bereiken door een gebeurtenis te verzenden vanuit een specifiek frame in de tijdlijn, zoals in het volgende codevoorbeeld wordt geïllustreerd:

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

De Document-klasse kan met behulp van de volgende coderegel naar deze gebeurtenis luisteren:

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

Deze benadering is in principe correct, maar het gebruik van het interne gebeurtenismodel verloopt wellicht trager en verbruikt wellicht meer geheugen dan een traditionele callbackfunctie. Gebeurtenisobjecten moeten namelijk in het geheugen worden gemaakt en toegewezen en dat leidt tot tragere prestaties. Als u bijvoorbeeld naar de gebeurtenis Event.ENTER_FRAME luistert, wordt er op elk frame voor de gebeurtenishandler een nieuw gebeurtenisobject gemaakt. Vooral weergaveobjecten zijn aanzienlijk trager vanwege de vastleg- en terugkoppelingsfasen die lang kunnen duren in geval van een complexe weergavelijst.

Hoofdstuk 3: CPU-gebruik minimaliseren

Een ander belangrijk focusgebied voor optimalisatie is het CPU-gebruik. Optimalisatie van CPU-verwerking verbetert de prestaties en dat betekent in geval van mobiele apparatuur dat de batterij langer meegaat.

Flash Player 10.1-uitbreidingen voor CPU-gebruik

Flash Player 10.1 introduceert twee nieuwe functies die de CPU-verwerking helpen opslaan. Met de functies kan SWF-inhoud worden gepauzeerd en voortgezet wanneer de inhoud offscreen gaat en wordt het aantal instanties van Flash Player op een pagina beperkt.

Pauzeren, vertragen en hervatten

Opmerking: *De functie voor pauzeren, vertragen en hervatten is niet van toepassing op Adobe® AIR®-toepassingen.*

Om het CPU- en batterijgebruik te optimaliseren, introduceert Flash Player 10.1 een nieuwe functie voor inactieve instanties. U kunt het CPU-verbruik zo beperken door het SWF-bestand te pauzeren en te hervatten wanneer de content op het scherm verschijnt en ervan verdwijnt. Met deze functie, geeft Flash Player zoveel mogelijk geheugen vrij door objecten te verwijderen die opnieuw kunnen worden gemaakt wanneer de content verder wordt afgespeeld. Content wordt beschouwd als offscreen, wanneer de volledige content offscreen is.

De SWF-content bevindt zich in twee scenario's offscreen:

- De gebruiker verschuift de pagina en verplaatst de SWF-content offscreen.

  In dat geval wordt het afspelen van eventuele audio of video voortgezet, maar de rendering wordt beëindigd. Als er geen audio of video wordt afgespeeld, stelt u de HTML-parameter `hasPriority` in op true om ervoor te zorgen dat het afspelen of het uitvoeren van ActionScript niet wordt onderbroken. Houd er echter rekening mee dat de rendering van SWF-content wordt gepauzeerd wanneer de content verborgen is of niet op het scherm wordt weergegeven, ongeacht de waarde van de HTML-parameter `hasPriority`.

- Er wordt een tabblad geopend in de browser, waardoor de SWF-content naar de achtergrond wordt verplaatst.

  In dat geval wordt de SWF-inhoud *vertraagd* tot tussen 2 en 8 fps, ongeacht de waarde van de HTML-parameter `hasPriority`. Het afspelen van audio en video wordt beëindigd en er wordt geen inhoud gerenderd totdat de SWF-inhoud weer zichtbaar wordt.

Als Flash Player 11.2 en later via een desktopbrowser op Windows en Mac wordt uitgevoerd, kunt u de ThrottleEvent in uw toepassing gebruiken. Flash Player verzendt een ThrottleEvent wanneer Flash Player wordt gepauzeerd, wordt vertraagd of wanneer het afspelen wordt hervat.

ThrottleEvent is een uitzendgebeurtenis, wat betekent dat deze wordt verzonden door alle objecten EventDispatcher met een listener die is ingesteld voor deze gebeurtenis. Zie De klasseDisplayObject voor informatie over uitzendgebeurtenissen.

Instantiebeheer

Opmerking: *De functie voor instantiebeheer is niet van toepassing op Adobe® AIR®-toepassingen.*

Gebruik de HTML-parameter `hasPriority` om het laden van de offscreen SWF-bestanden te vertragen.

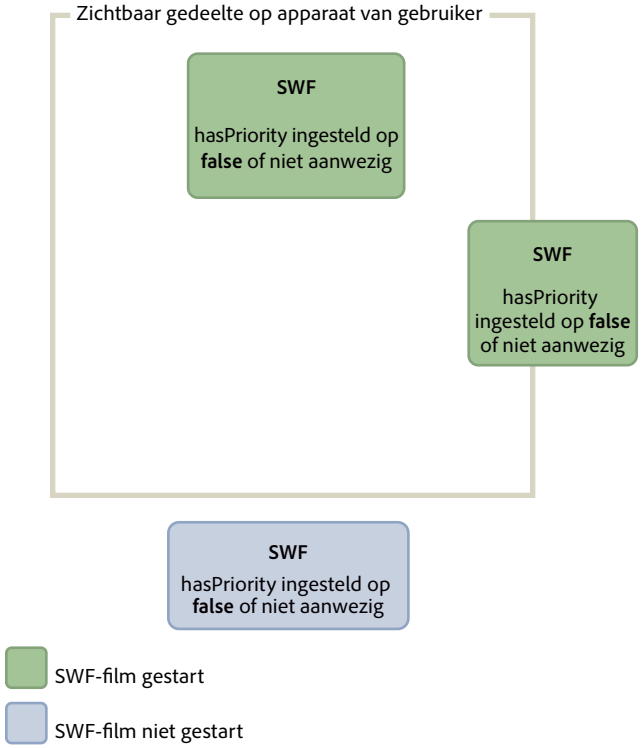
Flash Player 10.1 beschikt over de nieuwe HTML-parameter `hasPriority`:

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

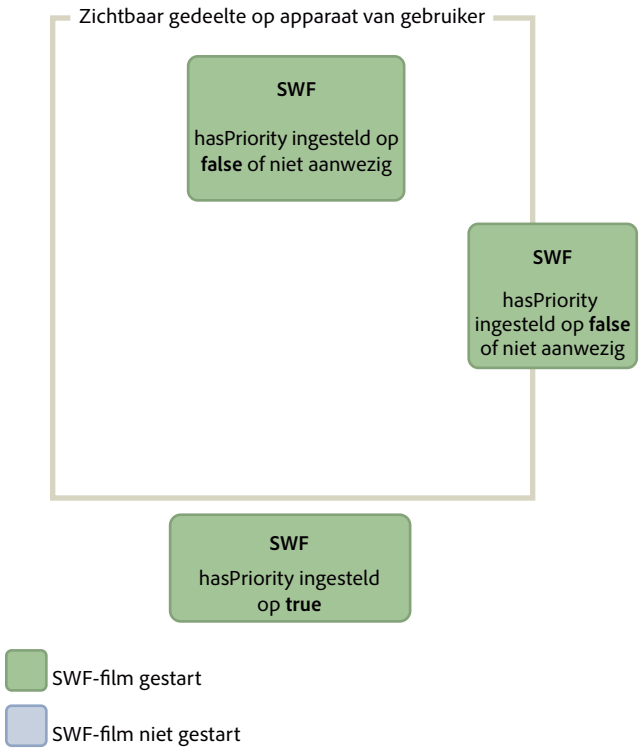
Hiermee beperkt u het aantal Flash Player-instanties dat op een pagina wordt gestart. Op deze manier verlaagt u het CPU- en batterijverbruik. U kunt hiermee een specifieke prioriteit aan SWF-content toewijzen, zodat content prioriteit krijgt over andere content op een pagina. Een eenvoudig voorbeeld: een gebruiker bladert door een website en de indexpagina bevat drie verschillende SWF-bestanden. Een van deze bestanden is zichtbaar, het andere is gedeeltelijk zichtbaar op het scherm en de laatste bevindt zich offscreen, waardoor u moet verschuiven. De eerste twee animaties zijn normaal gestart, maar de laatste is uitgesteld totdat deze zichtbaar wordt. Dit scenario is het standaardgedrag als de `hasPriority`-parameter niet aanwezig is of is ingesteld op `false`. Om ervoor te zorgen dat er een SWF-bestand wordt gestart, zelfs als dit zich offscreen bevindt, stelt u de `hasPriority`-parameter in op `true`. Ongeacht de waarde van de parameter `hasPriority` wordt de rendering van SWF-bestanden die de gebruiker niet kan zien altijd gepauzeerd.

Opmerking: *Als de beschikbare CPU-middelen op een laag niveau staan, worden Flash Player-instanties niet meer automatisch gestart, zelfs niet als de `hasPriority`-parameter op `true` is ingesteld. Als er met JavaScript nieuwe instanties worden gemaakt nadat de pagina geladen is, zullen die instanties de `hasPriority`-markering negeren. Er wordt 1x1- of 0x0-pixelcontent gestart, waardoor de SWF-hulpbestanden niet worden uitgesteld als de webmaster geen `hasPriority`-markering heeft ingevoegd. U kunt SWF-bestanden echter nog steeds starten door erop te klikken. Dit gedrag wordt "klikken om af te spelen" genoemd.*

In het volgende diagram ziet u wat er gebeurt als u de parameter `hasPriority` instelt op verschillende waarden:



Effecten van verschillende waarden voor de parameter hasPriority



Effecten van verschillende waarden voor de parameter hasPriority

Slaapmodus

In Flash Player 10.1 en AIR 2.5 wordt een nieuwe functie op mobiele apparaten geïntroduceerd, waarmee wordt bespaard op CPU-verwerking en zodat de batterij langer meegaat. Deze functie heeft betrekking op de achtergrondverlichting waarover veel mobiele apparaten beschikken. Als een gebruiker bijvoorbeeld tijdens het gebruik van een mobiele toepassing wordt onderbroken en het apparaat niet meer gebruikt, detecteert de runtime wanneer de achtergrondverlichting overgaat naar de slaapmodus. De framesnelheid wordt vervolgens verlaagt tot 4 frames per seconde (fps) en de rendering wordt gepauzeerd. Voor AIR-toepassingen wordt de slaapmodus ook geactiveerd wanneer de toepassing naar de achtergrond wordt verplaatst.

ActionScript-code wordt verder uitgevoerd in slaapmodus, zoals het instellen van de `Stage.frameRate`-eigenschap op 4 fps. Maar de renderingsstap wordt overgeslagen, zodat de gebruiker niet ziet dat de speler ingesteld is op 4 fps. Er wordt gekozen voor een framesnelheid van 4 fps, in plaats van nul, omdat hierdoor alle verbindingen open kunnen blijven (NetStream, Socket, and NetConnection). Als er op nul overgeschakeld zou worden, zouden open verbindingen onderbroken worden. Er werd gekozen voor een vernieuwingsfrequentie van 250 ms (4 fps), omdat veel producenten van mobiele apparaten deze framesnelheid als hun vernieuwingsfrequentie gebruiken. Als u deze waarde gebruikt, blijft de framesnelheid van de runtime dezelfde als die van het apparaat.

Opmerking: *Wanneer de slaapmodus voor de runtime is geactiveerd, retourneert de eigenschap* `Stage.frameRate` *de framesnelheid van het oorspronkelijke SWF-bestand in plaats van 4 fps.*

Wanneer de achtergrondverlichting weer wordt ingeschakeld, wordt er doorgegaan met de rendering. De framesnelheid retourneert de oorspronkelijke waarde. Bijvoorbeeld: een mediaspelertoepassing waarmee een gebruiker muziek afspeelt. Als het scherm overschakelt naar de slaapmodus, reageert de runtime op basis van het type inhoud dat wordt afgespeeld. Hier volgt een lijst met situaties en het bijhorende runtimegedrag.

- De achtergrondverlichting schakelt over naar de slaapmodus en er wordt niet-A/V-inhoud afgespeeld. De rendering wordt gepauzeerd en de framesnelheid wordt ingesteld op 4 fps.
- De achtergrondverlichting schakelt over naar de slaapmodus en er wordt A/V-inhoud afgespeeld. De runtime bepaalt dat de achtergrondverlichting altijd aan is, zodat de gebruiker door kan gaan.
- De achtergrondverlichting schakelt over van de slaapmodus naar de aan-modus. De runtime stelt de framesnelheid in op de instelling voor de framesnelheid van het oorspronkelijke SWF-bestand en gaat door met de rendering.
- Flash Player is gepauzeerd terwijl er A/V-inhoud wordt afgespeeld. Flash Player herstelt het standaardsysteemgedrag van de achtergrondverlichting, omdat er geen A/V meer wordt afgespeeld.
- Het mobiele apparaat ontvangt een telefoongesprek terwijl er A/V-inhoud wordt afgespeeld. De rendering wordt gepauzeerd en de framesnelheid wordt ingesteld op 4 fps.
- De slaapmodus van de achtergrondverlichting op een mobiel apparaat wordt uitgeschakeld. De runtime gedraagt zich normaal.

Wanneer de achtergrondverlichting overgaat in de slaapmodus, wordt de rendering gepauzeerd en de framesnelheid vertraagt. Deze functie bespaart op CPU-verwerking, maar u kunt er niet op rekenen dat deze een echte pauze maakt, zoals in een gametoepassing.

Opmerking: *Er wordt geen ActionScript-gebeurtenis verzonden wanneer de runtime schakelt tussen de actieve modus en de slaapmodus.*

Objecten vastzetten en vrijgeven

U kunt objecten op de juiste wijze vastzetten en vrijgeven met gebruik van de gebeurtenissen `REMOVED_FROM_STAGE` *en* `ADDED_TO_STAGE`*.*

Om uw code te optimaliseren, moet u uw objecten altijd vastzetten en vrijgeven. Vastzetten en vrijgeven is belangrijk voor alle objecten, maar vooral voor weergaveobjecten. Zelfs als weergaveobjecten niet meer in de weergavelijst staan en wachten op de afvalophaling, kunnen deze nog steeds CPU-intensieve code gebruiken. Ze kunnen bijvoorbeeld nog steeds Event.ENTER_FRAME gebruiken. Het is dus van essentieel belang objecten op de juiste wijze vast te zetten en vrij te geven met de gebeurtenissen `Event.REMOVED_FROM_STAGE` en `Event.ADDED_TO_STAGE`. In het volgende voorbeeld wordt in het werkgebied een filmclip afgespeeld die reageert op het toetsenbord:

```
// 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;
    }
}
```

Filmclip die interactief werkt met het toetsenbord

Wanneer op de knop Verwijderen wordt geklikt, wordt de filmclip uit de weergavelijst verwijderd.

```
// 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);
}
```

Zelfs als de filmclip uit de weergavelijst wordt verwijderd, verstuurt deze nog steeds de `Event.ENTER_FRAME`-gebeurtenis. De filmclip is nog steeds actief, maar wordt niet gerenderd. U verwerkt deze situatie op de juiste wijze door naar de juiste gebeurtenissen te luisteren en gebeurtenislisteners te verwijderen om te voorkomen dat CPU-intensieve code wordt uitgevoerd:

```
// 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();
}
```

Wanneer op de knop Tonen wordt geklikt, wordt de filmclip weer gestart, luistert deze weer naar de gebeurtenissen `Event.ENTER_FRAME` en bestuurt het toetsenbord de filmclip op de juiste wijze.

Opmerking: *Het verwijderen van een weergaveobject uit de weergavelijst en het instellen van de referentie van dat object op* `null` *nadat het object is verwijderd, garandeert niet dat het object wordt vastgezet. Als de afvalophaling niet actief is, blijft het object geheugen en CPU-verwerking gebruiken, zelfs als het object niet meer wordt weergegeven. Om ervoor te zorgen dat het object zo weinig mogelijk CPU-verwerking gebruikt, moet u controleren of het volledig is vastgezet wanneer het uit de weergavelijst is verwijderd.*

Bij Flash Player 10 en AIR 1.5 doet zich nu ook het volgende gedrag voor. Als de afspeelkop een leeg frame ontdekt, wordt het weergaveobject automatisch stilgezet, zelfs als u geen gedrag voor het stilzetten hebt geïmplementeerd.

Het concept van stilzetten is ook belangrijk als u externe inhoud laadt met behulp van de Loader-klasse. Wanneer u in Flash Player 9 en AIR 1.0 de Loader-klasse gebruikt, moet u inhoud handmatig stilzetten door te luisteren naar de `Event.UNLOAD`-gebeurtenis die door het LoaderInfo-object wordt verstuurd. Elk object moet handmatig worden stilgezet en dit is geen onbelangrijke taak. In Flash Player 10 en AIR 1.5 werd een belangrijke nieuwe methode voor de Loader-klasse geïntroduceerd met de naam `unloadAndStop()`. Met deze methode kunt u een SWF-bestand ontladen, automatisch elk object in het geladen SWF-bestand vastzetten en de afvalophaling starten.

In de volgende code wordt het SWF-bestand geladen en ontladen met behulp van de `unload()`-methode, die meer verwerking en handmatige vastzetting vereist.

```
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();
}
```

We raden u aan om de `unloadAndStop()`-methode te gebruiken, die de vastzetting intern behandelt en het afvalophalingsproces start.

```
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();
}
```

Als de `unloadAndStop()`-methode wordt aangeroepen, vinden de volgende acties plaats:

- Geluid wordt gestopt.
- Listeners die zijn geregistreerd voor de hoofdtijdlijn van het SWF-bestand worden verwijderd.
- Timerobjecten worden stopgezet.
- Hardware randapparatuur (zoals camera's en microfoons) worden vrijgegeven.
- Elke filmclip wordt gestopt.
- Het versturen van `Event.ENTER_FRAME`, `Event.FRAME_CONSTRUCTED`, `Event.EXIT_FRAME`, `Event.ACTIVATE` en `Event.DEACTIVATE` wordt gestopt.

Gebeurtenissen activeren en deactiveren

Gebruik de gebeurtenissen `Event.ACTIVATE` en `Event.DEACTIVATE` om inactiviteiten op de achtergrond te detecteren en optimaliseer vervolgens uw toepassing.

Met de twee gebeurtenissen `Event.ACTIVATE` en `Event.DEACTIVATE` kunt u uw toepassing nauwkeurig instellen, zodat deze zo min mogelijk CPU-cycli gebruikt. Met deze gebeurtenissen kunt u detecteren wanneer de runtime de focus krijgt of verliest. De code kan als gevolg hiervan worden geoptimaliseerd om te reageren op contextwijzigingen. De volgende code luistert naar beide gebeurtenissen en verandert dynamisch de framesnelheid in nul wanneer de toepassing de focus verliest. De animatie kan bijvoorbeeld de focus verliezen wanneer de gebruiker overschakelt naar een ander tabblad of de toepassing naar de achtergrond verplaatst:

```
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;
}
```

Wanneer de toepassing opnieuw de focus krijgt, wordt de framesnelheid weer ingesteld op de oorspronkelijke waarde. In plaats van de framesnelheid dynamisch te wijzigen, kunt u de toepassing ook op andere manieren optimaliseren, bijvoorbeeld door het stilzetten en het opnieuw activeren van objecten.

Met het activeren en deactiveren van gebeurtenissen kunt u een mechanisme implementeren dat vergelijkbaar is met de functie voor pauzeren en voortzetten die soms op mobiele apparaten en netbooks zijn te vinden.

Meer Help-onderwerpen

"Framesnelheid van een toepassing" op pagina 54

"Objecten vastzetten en vrijgeven" op pagina 28

Muisinteracties

U kunt proberen muisinteracties waar mogelijk uit te schakelen.

Wanneer u interactieve objecten gebruikt, zoals een MovieClip- of Sprite-object, voert de runtime de interne code uit om muisinteracties te detecteren en te verwerken. Detectie van muisinteractie kan veel van de CPU vergen als vele interactieve objecten op het scherm verschijnen, vooral als ze elkaar overlappen. U kunt dit gemakkelijk voorkomen door muisinteractie uit te schakelen voor objecten waarvoor muisinteractie overbodig is. De volgende code illustreert het gebruik van de eigenschappen `mouseEnabled` en `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;
```

U kunt indien mogelijk de muisinteracties ook uitschakelen, zodat uw toepassing minder van de CPU vergt en de batterij op het mobiele apparaat dus langer meegaat.

Timers versus ENTER_FRAME-gebeurtenissen.

Kies timers of `ENTER_FRAME`-gebeurtenissen, afhankelijk van het feit of de content geanimeerd is.

Gebruik voor niet-geanimeerde content die gedurende een lange periode wordt uitgevoerd liever timers dan `Event.ENTER_FRAME`-gebeurtenissen.

U kunt in ActionScript 3.0 op twee verschillende manieren een functie bij een bepaald interval aanroepen. U kunt bijvoorbeeld de gebeurtenis `Event.ENTER_FRAME` gebruiken die wordt verzonden door weergaveobjecten (DisplayObject). Maar u kunt ook een timer gebruiken. ActionScript-ontwikkelaars kiezen vaak voor de `ENTER_FRAME`-aanpak. De gebeurtenis `ENTER_FRAME` wordt op elk frame verzonden. Dat betekent dat het interval waarbij de functie wordt aangeroepen gerelateerd is aan de huidige framesnelheid. De framesnelheid is toegankelijk via de eigenschap `Stage.frameRate`. In sommige gevallen is het echter beter een timer te gebruiken dan de `ENTER_FRAME`-gebeurtenis. Als u bijvoorbeeld geen animatie gebruikt, maar de code bij een specifiek interval wilt aanroepen, kunt u beter een timer gebruiken.

Een timer kan net zo functioneren als een `ENTER_FRAME`-gebeurtenis, maar een gebeurtenis kan worden verzonden zonder te zijn gekoppeld aan de framesnelheid. En dat kan tot aanzienlijke optimalisatie leiden. Laten we een videospelertoepassing als voorbeeld gebruiken. In dit geval hoeft u geen hoge framesnelheid te gebruiken, omdat alleen de besturingselementen van de toepassing zich verplaatsen.

Opmerking: *De framesnelheid beïnvloedt de video niet, omdat de video niet ingesloten is in de tijdlijn. De video wordt namelijk dynamisch geladen via progressief downloaden of streaming.*

In dit voorbeeld is een lage framesnelheid ingesteld, namelijk 10 fps. De timer werkt de besturingselementen één keer per seconde bij. Deze hogere bijwerksnelheid wordt mogelijk gemaakt door de methode `updateAfterEvent()` die beschikbaar is op het TimerEvent-object. Iedere keer als de timer een gebeurtenis verzendt, forceert deze methode schermvernieuwing, als dat nodig is. De volgende code illustreert dit:

```
// 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();
}
```

De framesnelheid wordt niet gewijzigd wanneer de methode `updateAfterEvent()` wordt aangeroepen. De methode dwingt de runtime alleen om de inhoud op het scherm bij te werken. De snelheid van de tijdlijn blijft 10 fps. Vergeet niet dat timers en `ENTER_FRAME`-gebeurtenissen niet volledig nauwkeurig zijn op minder geavanceerde apparaten en dat gebeurtenishandlerfuncties met code veel verwerking vereisen. Net als de framesnelheid van het SWF-bestand kan de bijwerkframesnelheid van de timer in bepaalde situaties variëren.

Gebruik in uw toepassing zo weinig mogelijk objecten Timer en geregistreerde handlers `enterFrame`.

Tijdens elk frame wordt een gebeurtenis `enterFrame` naar elk weergaveobject in de weergavelijst verzonden. U kunt listeners voor de gebeurtenis `enterFrame` met meerdere weergaveobjecten registreren, maar dat betekent dat er in elk frame meer code wordt uitgevoerd. Het is beter om één gecentraliseerde `enterFrame`-handler te gebruiken waarmee alle code voor elk frame wordt uitgevoerd. Als u deze code centraliseert, is het eenvoudiger om alle code die vaak wordt uitgevoerd te beheren.

Als u objecten Timer gebruikt, is er sprake van overhead als gevolg van het maken en verzenden van gebeurtenissen vanaf meerdere objecten Timer. Als u verschillende bewerkingen met verschillende intervallen moet triggeren, kunt u dit ook op de volgende manieren doen:

- Gebruik een zo klein mogelijk aantal objecten Timer en groepbewerkingen, bepaald door het aantal keer dat deze voorkomen.

  Gebruik bijvoorbeeld één Timer voor frequente bewerkingen, die elke 100 milliseconden wordt getriggerd. Gebruik een andere Timer voor minder frequente of achtergrondbewerkingen, die elke 2000 milliseconden wordt getriggerd.

- Gebruik één object Timer en zorg ervoor dat de frequentie waarmee bewerkingen worden getriggerd een meervoud is van het interval van de eigenschap `delay` van het object Timer.

  Stel dat u bewerkingen hebt die elke 100 milliseconde moeten worden uitgevoerd en bewerkingen die elke 200 milliseconde moeten worden uitgevoerd. In dat geval gebruikt u één object Timer, waarvoor bij `delay` een waarde van 100 milliseconde is ingesteld. In de gebeurtenishandler `timer` voegt u een voorwaardelijke instructie toe waarmee de 200-milliseconde-bewerkingen om en om worden uitgevoerd. Deze techniek wordt in het volgende voorbeeld getoond:

```
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;
}
```

Onderbreek objecten Timer onderbreken wanneer deze niet worden gebruikt.

Als de gebeurtenishandler `timer` van een object Timer alleen onder bepaalde omstandigheden bewerkingen uitvoert, roept u de methode `stop()` van het object Timer aan wanneer aan geen van de voorwaarden wordt voldaan.

Zorg ervoor dat in de gebeurtenis `enterFrame` of Timer-handlers zo weinig mogelijk wijzigingen in de weergave van het object plaatsvinden die tot gevolg hebben dat het scherm opnieuw getekend moet worden.

In elk frame wordt in de renderingfase het gedeelte van het werkgebied dat tijdens dat frame is gewijzigd opnieuw getekend. Als het opnieuw te tekenen gebied groot is, of klein is maar een groot aantal of complexe weergaveobjecten bevat, heeft de runtime meer tijd nodig voor rendering. Met de functie Opnieuw te tekenen gebieden weergeven in de foutopsporingsversie van Flash Player of AIR kunt u testen hoeveel tijd er nodig is voor het opnieuw teken.

Raadpleeg het volgende artikel voor meer informatie over het verbeteren van de prestaties voor herhaalde acties:

- Writing well-behaved, efficient, AIR applications (artikel en voorbeeld van Arno Gourdol)

Meer Help-onderwerpen

"Gedragingen isoleren" op pagina 66

Tweeningsyndroom

U kunt CPU-vermogen besparen door het gebruik van tweening te beperken. Zo bespaart u CPU-verwerking, geheugen en de batterij.

Ontwerpers en ontwikkelaars die desktopcontent voor Flash produceren, gebruiken vaak bewegingstweens in hun toepassingen. Beperk het gebruik van bewegingstweens tot een minimum wanneer u content produceert voor mobiele apparatuur met lage specificaties. De content wordt dan sneller uitgevoerd op dergelijke apparatuur.

Hoofdstuk 4: Prestatie van ActionScript 3.0

Vectorklasse versus Arrayklasse

Gebruik indien mogelijk de Vectorklasse in plaats van de Arrayklasse.

Met de Vectorklasse is snellere lees- en schrijftoegang mogelijk dan met de Arrayklasse.

Een eenvoudige standaard omschrijft de voordelen van de Vectorklasse in vergelijking met de Arrayklasse. In de volgende code wordt een standaard voor de Arrayklasse weergegeven.

```
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
```

In de volgende code wordt een standaard voor de Vectorklasse weergegeven.

```
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
```

Het voorbeeld kan verder worden geoptimaliseerd door een specifieke lengte aan de vector toe te wijzen en deze lengte vast te leggen.

```
// 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
```

Als de grootte van de vector niet van tevoren is opgegeven, neemt de grootte toe wanneer de vector over onvoldoende ruimte beschikt. Steeds wanneer de grootte van de vector toeneemt, wordt een nieuwe geheugenblok toegewezen. De huidige inhoud van de vector wordt naar het nieuwe geheugenblok gekopieerd. De extra geheugentoewijzing en het kopiëren van de gegevens heeft negatieve gevolgen voor de prestaties. Bovenstaande code is geoptimaliseerd doordat de aanvankelijke grootte van de vector is opgegeven. De code is echter niet geoptimaliseerd vanuit het oogpunt van onderhoudsgemak. Als u dat ook wilt verbeteren, slaat u de opnieuw gebruikte waarde op in een constante waarde:

```
// 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
```

Probeer, voor zover mogelijk, de vectorobject-API's te gebruiken, aangezien die sneller uitgevoerd worden.

Teken-API

Gebruik de teken-API voor een snellere uitvoering van de code.

Flash Player 10 en AIR 1.5 bieden een nieuwe teken-API, waardoor u betere prestaties van de code krijgt. Deze nieuwe API biedt geen verbetering van de renderingsprestatie, maar kan het aantal regels van de code die u moet schrijven sterk verminderen. Minder coderegels leiden tot een betere prestatie van ActionScript tijdens de uitvoering.

De nieuwe teken-API bevat de volgende methoden:

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

Opmerking: *Deze discussie richt zich niet op de `drawTriangles()`-methode, die gerelateerd is aan 3D. Deze methode kan echter de prestatie van ActionScript verbeteren, omdat deze de toewijzing van de oorspronkelijke textuur behandelt.*

De volgende code roept expliciet de toepasselijke methode op voor elke getekende regel.

```
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 );
```

De volgende code wordt sneller dan het vorige voorbeeld uitgevoerd, omdat deze minder coderegels uitvoert. Hoe complexer het pad, hoe groter de prestatietoename met de `drawPath()`-methode.

```
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 );
```

De `drawGraphicsData()`-methode biedt vergelijkbare prestatieverbeteringen.

Gebeurtenissen vastleggen en terugkoppelen

U kunt gebeurtenissen vastleggen en terugkoppelen om gebeurtenishandlers tot een minimum te beperken.

In het gebeurtenismodel in ActionScript 3.0 werden de concepten gebeurtenissen vastleggen en gebeurtenissen terugkoppelen geïntroduceerd. U kunt gebeurtenissen terugkoppelen om u te helpen de uitvoeringstijd van ActionScript-code te optimaliseren. U kunt een gebeurtenishandler voor één object registreren in plaats van voor meerdere objecten en zo de prestaties verbeteren.

Denk bijvoorbeeld aan een game waarin gebruikers zo snel mogelijk op appels moeten klikken om deze te vernietigen. De game verwijdert elke appel van het scherm wanneer de speler erop klikt en voegt punten toe aan de score van de gebruiker. U zou de volgende code kunnen schrijven om de door elke appel verzonden `MouseEvent.CLICK`-gebeurtenis te signaleren:

```
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 );
}
```

De code roept de methode `addEventListener()` aan voor elke appel-instantie. Bovendien wordt de `removeEventListener()`-methode gebruikt om elke listener te verwijderen wanneer op een appel wordt geklikt. Het gebeurtenismodel in ActionScript 3.0 verschaft echter een vastleg- en terugkoppelingsfase voor bepaalde gebeurtenissen, zodat u ernaar kunt luisteren vanuit een bovenliggend InteractiveObject. Het resultaat is dat u de zojuist vermelde code kunt optimaliseren en het aantal aanroepen naar de methoden `addEventListener()` en `removeEventListener()` tot een minimum kunt beperken. In de volgende code wordt de vastlegfase gebruikt om naar de gebeurtenissen van het bovenliggende object te luisteren:

```
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 );
}
```

De code is eenvoudiger en aanzienlijk geoptimaliseerd, met slechts één aanroep naar de methode `addEventListener()` in de bovenliggende container. Listeners worden niet meer geregistreerd bij de appel-instanties en hoeven dus niet meer te worden verwijderd wanneer de gebruiker op een appel klikt. De `onAppleClick()`-handler kan verder worden geoptimaliseerd door het doorgeven van de gebeurtenis te stoppen, zodat de gebeurtenis zich niet voortzet:

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

De terugkoppelingsfase kan ook worden gebruikt om de gebeurtenis af te vangen, door `false` als de derde parameter door te geven aan de methode `addEventListener()`:

```
// 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 standaardwaarde voor de parameter van de vastlegfase is `false`, zodat u deze kunt weglaten:

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

Werken met pixels

Verfpixels die de `setVector()`-methode gebruiken.

Wanneer u pixels verft, kunnen er eenvoudige optimalisaties worden uitgevoerd met behulp van methoden in de BitmapData-klasse. U kunt de methode `setVector()` gebruiken om snel pixels te tekenen:

```
// 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 );
```

Wanneer u langzame methoden gebruikt, zoals `setPixel()` of `setPixel32()`, gebruikt u de methoden `lock()` en `unlock()` voor een snellere uitvoering. In de volgende code worden de methoden `lock()` en `unlock()` gebruikt om de prestatie te verbeteren.

```
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
```

De `lock()`-methode van de BitmapData-klasse vergrendelt een afbeelding en voorkomt dat objecten die ernaar verwijzen worden bijgewerkt als het BitmapData-object wijzigt. Als een Bitmap-object bijvoorbeeld verwijst naar een BitmapData-object, kunt u het BitmapData-object vergrendelen, wijzigen en ontgrendelen. Het Bitmap-object wordt niet gewijzigd, voordat het BitmapData-object wordt ontgrendeld. Gebruik deze methode met de methode `unlock()` voor en na verscheidene aanroepen van de methode `setPixel()` of `setPixel32()` om de prestaties te verbeteren. Als u `lock()` en `unlock()` oproept, wordt het scherm niet onnodig bijgewerkt.

Opmerking: *Deze techniek niet altijd tot betere prestaties bij het verwerken van pixels op een bitmap die niet op de weergavelijst staat (dubbele buffering). Als een bitmapobject niet naar de bitmapbuffer verwijst, verbeteren de prestaties niet wanneer u `lock()` en `unlock()` gebruikt. Flash Player detecteert dat er niet naar de buffer wordt verwezen en de bitmap niet op het scherm wordt gerenderd.*

Methoden die pixelherhalingen uitvoeren, zoals `getPixel()`, `getPixel32()`, `setPixel()` en `setPixel32()`, werken wellicht vrij langzaam, vooral op mobiele apparatuur. Gebruik, indien mogelijk, methoden die alle pixels in één oproep opvragen. Voor het lezen van pixels gebruikt u de `getVector()`-methode, die sneller is dan de `getPixels()`-methode. Gebruik waar mogelijk API's die gebruikmaken van Vector-objecten; deze zijn sneller.

Reguliere expressies

Gebruik methoden van de klasse String, zoals `indexOf()`, `substr()` of `substring()`, in plaats van reguliere expressies voor het zoeken naar en extraheren van tekenreeksen.

Bepaalde bewerkingen die kunnen worden uitgevoerd met een reguliere expressie kunnen ook worden uitgevoerd met methoden van de klasse String. Als u bijvoorbeeld wilt achterhalen of een tekenreeks een andere tekenreeks bevat, kunt u de methode `String.indexOf()` of een reguliere expressie gebruiken. Als er echter een methode van de klasse String beschikbaar is, wordt deze sneller uitgevoerd dan de vergelijkbare reguliere expressie, en hoeft er bovendien geen extra object te worden gemaakt.

Gebruik in een reguliere expressie een niet-vastleggende groep ("`(?:xxxx)`") in plaats van een groep ("`(xxxx)`") om elementen te groeperen zonder de inhoud van de groep in het resultaat te isoleren.

In reguliere expressies van gemiddelde complexiteit worden delen van de expressie vaak gegroepeerd. In het volgende reguliere expressiepatroon wordt bijvoorbeeld met haakjes een groep rond de tekst "ab" gemaakt. De hoeveelheidsbepaler "+" is van toepassing op de groep en niet op één teken:

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

De inhoud van elke groep wordt standaard "vastgelegd". De inhoud van elke groep kan in het patroon worden geplaatst als onderdeel van het resultaat van het uitvoeren van de reguliere expressie. Het vastleggen van deze groepsresultaten duurt langer en vereist meer geheugen, omdat er objecten worden gemaakt waarin de groepsresultaten worden geplaatst. U kunt ook de syntaxis voor een niet-vastleggende groep gebruiken. Dit doet u door een vraagtaken en dubbele punt te plaatsen na het haakje openen. Deze syntaxis geeft aan dat de tekens zich als een groep gedragen, maar niet voor het resultaat worden vastgelegd:

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

De syntaxis voor een niet-vastleggende groep is sneller en gebruikt minder geheugen dan de standaardsyntaxis voor groepen.

Gebruik een ander·patroon voor een reguliere expressie als deze slechte prestaties oplevert.

Soms kan er meer dan één regulier expressiepatroon worden gebruikt om een tekstpatroon te identificeren. Om verschillende redenen worden bepaalde patronen sneller uitgevoerd dan andere. Als u vaststelt dat een reguliere expressie ervoor zorgt dat de code langzamer wordt uitgevoerd dan nodig is, kunt u alternatieve reguliere-expressiepatronen overwegen die tot hetzelfde resultaat leiden. Test deze alternatieve patronen om te bepalen welke het snelste is.

Overige optimalisaties

Gebruik voor een TextField-object liever de methode `appendText()` in plaats van de `+=`-operator.

Wanneer u de `text`-eigenschap van de TextField-klasse gebruikt, gebruikt u de `appendText()`-methode in plaats van de `+=`-operator. Als u de `appendText()`-methode gebruikt, levert dit prestatieverbeteringen op.

De volgende code gebruikt bijvoorbeeld de `+=`-operator en de loop duurt 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
```

In het volgende voorbeeld, wordt de `+=`-operator vervangen met behulp van de `appendText()`-methode.

```
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
```

Het voltooien van de code duurt nu 847 ms.


Werk tekstvelden indien mogelijk buiten loops bij.

Deze code kan zelfs verder worden geoptimaliseerd met behulp van een eenvoudige techniek. Het bijwerken van een tekstveld in elke loop neemt veel interne verwerking in beslag. De uitvoeringstijd van een code wordt sterk verminderd als u een tekenreeks samenvoegt en deze aan een tekstveld buiten de loop toewijst. Het voltooien van de code duurt nu 2 ms.

```
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
```

Wanneer u met HTML-tekst werkt, is de eerste aanpak zo langzaam, dat deze in sommige gevallen een `Timeout`-uitzondering veroorzaakt in Flash Player: Er kan bijvoorbeeld een uitzondering optreden als de onderliggende hardware te traag is.

Opmerking: *Deze uitzondering treedt niet op in Adobe® AIR®.*

```
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 );
```

Als u de waarde aan een tekenreeks buiten de loop toewijst, wordt de code in slechts 29 ms voltooid.

```
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
```

Opmerking: *In Flash Player 10.1 en AIR 2.5 is de klasse String verbeterd, zodat strings minder geheugen gebruiken.*

Gebruik zo weinig mogelijk de vierkante-hakenoperator.

Het gebruik van deze operator kan de prestatie vertragen. U kunt gebruik ervan voorkomen, door uw referentie in een lokale variabele op te slaan. Het volgende codevoorbeeld geeft het inefficiënte gebruik van de vierkante-hakenoperator weer.

```
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
```

De volgende geoptimaliseerde versie beperkt het gebruik van deze operator:

```
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
```

Integreer indien mogelijk uw code om het aantal functieaanroepen in uw code te beperken.

Het aanroepen van functies kan duur zijn. Probeer het aantal functieaanroepen te beperken door inlinecode toe te passen. Door uw code inline te plaatsen, kunt u uw prestatie verder optimaliseren. Onthoud hierbij echter, dat door het gebruik van inlinecode uw code moeilijker herbruikbaar wordt en dat hierdoor de grootte van uw SWF-bestand groter wordt. Sommige functie-aanroepen, zoals de Math-klassemethoden, kunnen eenvoudig inline worden geplaatst. De volgende code gebruikt de `Math.abs()`-methode voor het berekenen van absolute waarden.

```
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
```

De berekening die door `Math.abs()` wordt uitgevoerd, kan handmatig worden uitgevoerd en inline worden geplaatst.

```
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
```

Als u de functieaanroep inline plaatst, resulteert dit in code die tot vier keer zo snel werkt. Deze aanpak is in veel situaties bruikbaar, maar houd rekening met het effect op de herbruikbaarheid en het onderhoudsgemak.

Opmerking: *De omvang van de code heeft grote invloed op de algemene prestaties van de speler. Als de toepassing grote hoeveelheden ActionScript-code omvat, besteedt de virtuele computer veel tijd aan het controleren van de code en aan JIT-compilaties. Eigenschap-lookups worden wellicht trager uitgevoerd, vanwege diepgaande, overgeërfde hiërarchieën en aangezien interne cachegeheugens vaker vastlopen. Gebruik het Adobe® Flex®-framework, de TLF-frameworkbibliotheek of een andere uitgebreide externe ActionScript-bibliotheek.*


Vermijd het evalueren van instructies in loops.

Een andere vorm van optimalisatie kan worden bereikt door een instructie niet te evalueren in een loop. De volgende code wordt over een array herhaald, maar is niet geoptimaliseerd, omdat de arraylengte voor elke herhaling wordt geëvalueerd:

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

U kunt de waarde beter opslaan en opnieuw gebruiken:

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

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


Gebruik omgekeerde volgorde voor while-loops.

Een while-loop in omgekeerde volgorde is sneller dan een voorwaartse loop:

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

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

Deze tips vormen een aantal manieren om ActionScript te optimaliseren en laten zien hoe één coderegel de prestaties en het geheugen kan beïnvloeden. Er zijn vele andere vormen van ActionScript-optimalisatie mogelijk. Klik op de volgende koppeling voor meer informatie: http://www.rozengain.com/blog/2007/05/01/some-actionscript-30-optimizations/.

Hoofdstuk 5: Renderprestaties

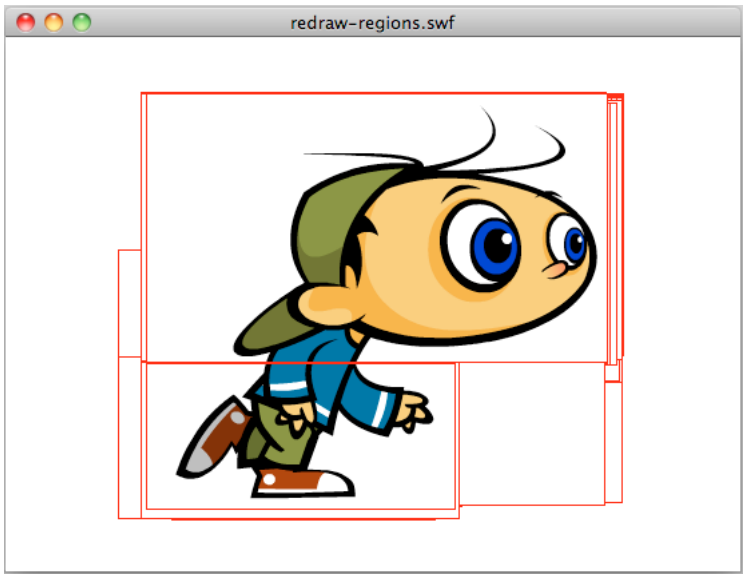
Gebieden opnieuw tekenen

Gebruik altijd de optie voor het opnieuw tekenen van gebieden wanneer u een project ontwikkelt.

Voor een betere rendering is het belangrijk dat u de optie voor het opnieuw tekenen van gebieden gebruikt wanneer u een project ontwikkelt. Met deze optie kunt u de gebieden zien die door Flash Player worden gerenderd en verwerkt. U kunt deze optie inschakelen door Opnieuw te tekenen gebieden weergeven in te schakelen in het contextmenu van de foutopsporingsversies van Flash Player.

Opmerking: *De optie Opnieuw te tekenen gebieden weergeven is niet beschikbaar in Adobe AIR of in de releaseversie van Flash Player. (In Adobe AIR is het contexmenu alleen beschikbaar in desktoptoepassingen, maar de toepassing heeft geen ingebouwde items of standaarditems, zoals Opnieuw te tekenen gebieden weergeven.)*

De onderstaande afbeelding illustreert deze optie met een eenvoudige MovieClip-animatie op de tijdlijn:



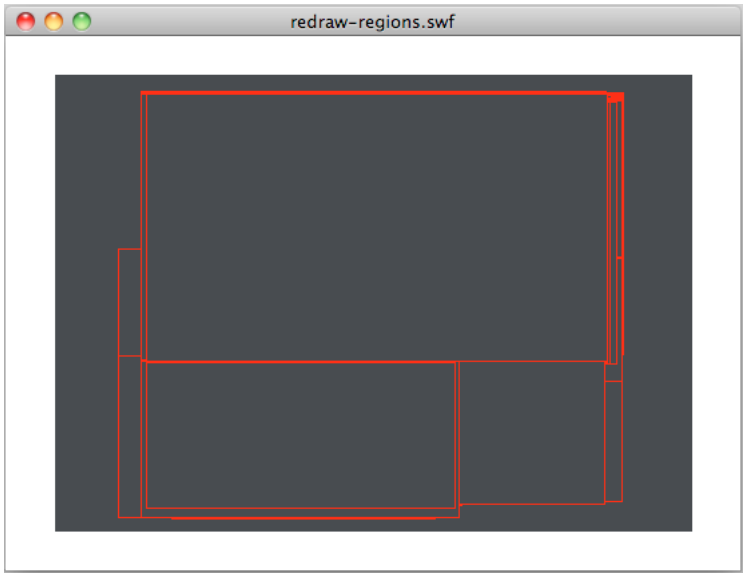
De optie voor het opnieuw tekenen van gebieden is ingeschakeld

U kunt deze optie ook programmatisch inschakelen met de methode `flash.profiler.showRedrawRegions()`:

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

In Adobe AIR-toepassingen kan alleen met deze methode de optie voor het opnieuw tekenen van gebieden worden ingeschakeld.

Gebruik de optie voor het opnieuw tekenen van gebieden om na te gaan welke optimalisatiemogelijkheden er beschikbaar zijn. Houd er rekening mee dat sommige weergaveobjecten die niet worden getoond, toch CPU-cycli verbruiken omdat ze wel worden gerenderd. Dit wordt in de volgende afbeelding geïllustreerd. De animatie van een rennend figuur wordt bedekt door een zwarte vectorvorm. De afbeelding laat zien dat het weergaveobject niet uit de weergavelijst is verwijderd en dus nog steeds wordt gerenderd. Hiermee worden CPU-cycli verspild:



Opnieuw getekende gebieden

Als u de prestaties wilt verbeteren, stelt u de eigenschap `visible` van de verborgen, rennende figuur in op `false` of verwijdert u de figuur uit de weergavelijst. U moet ook de bijbehorende tijdlijn stoppen. Deze stap zorgt ervoor dat het weergaveobject wordt stilgezet en slechts een minimum aan CPU-vermogen verbruikt.

Gebruik de optie voor het opnieuw tekenen van gebieden gedurende de gehele ontwikkelingscyclus. Als u deze optie gebruikt, wordt u aan het einde van het project niet onaangenaam verrast doordat gebieden opnieuw getekend worden zonder dat dat nodig is.

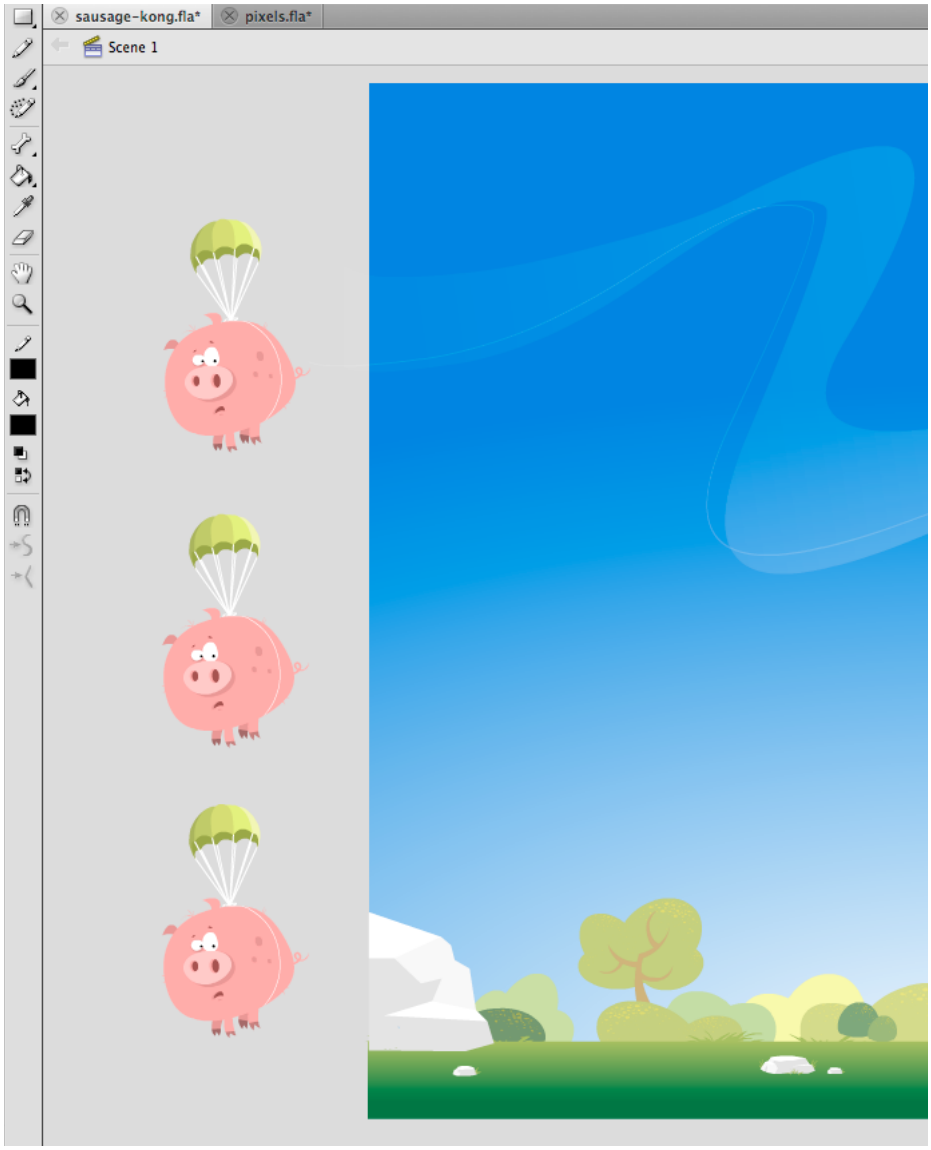
Meer Help-onderwerpen

"Objecten vastzetten en vrijgeven" op pagina 28

Inhoud buiten het werkgebied

Plaats geen inhoud buiten het werkgebied. Plaats objecten indien nodig gewoon in de weergavelijst.

Plaats indien mogelijk geen grafische inhoud buiten het werkgebied. Ontwerpers en ontwikkelaars plaatsen elementen gewoonlijk buiten het werkgebied om ze opnieuw te kunnen gebruiken tijdens de levensduur van de toepassing. De volgende afbeelding laat deze veelgebruikte techniek zien:



Inhoud buiten het werkgebied

Zelfs als de elementen buiten het werkgebied niet op het scherm worden weergegeven en niet zijn gerenderd, komen ze nog steeds voor in de weergavelijst. De runtime gaat door met het uitvoeren van interne tests op deze elementen om ervoor te zorgen dat ze zich nog steeds buiten het werkgebied bevinden en de gebruiker niet met ze communiceert. Plaats dus indien mogelijk geen objecten buiten het werkgebied en verwijder ze uit de weergavelijst.

Filmkwaliteit

Gebruik de geschikte Stage-kwaliteitsinstelling om de rendering te verbeteren.

Als u inhoud voor mobiele apparaten ontwikkelt met kleine schermen, zoals telefoons, is de afbeeldingskwaliteit minder belangrijk dan wanneer u bureaubladtoepassingen ontwikkelt. Als u de Stage-kwaliteit instelt op de geschikte instelling, kan dit de renderprestaties verbeteren.

De volgende instellingen zijn beschikbaar voor de Stage-kwaliteit:

- `StageQuality.LOW`: geeft voorrang aan de afspeelsnelheid boven het uiterlijk en past geen anti-aliasing toe. Deze instelling wordt niet ondersteund in Adobe AIR voor desktop of tv.
- `StageQuality.MEDIUM`: past enige anti-aliasing toe, maar maakt geschaalde bitmaps niet vloeiender. Deze instelling is de standaardwaarde voor AIR op mobiele apparaten, maar wordt niet ondersteund in AIR for TV of desktop.
- `StageQuality.HIGH`: (standaard op de desktop) geeft voorrang aan weergave boven de afspeelsnelheid en gebruikt altijd anti-aliasing. Als het SWF-bestand geen animatie bevat, worden geschaalde bitmaps vloeiend gemaakt. Als het SWF-bestand wel animatie bevat, worden bitmaps niet vloeiend gemaakt.
- `StageQuality.BEST`: biedt de beste weergavekwaliteit en houdt geen rekening met de afspeelsnelheid. Op alle uitvoer wordt anti-aliasing toegepast en geschaalde bitmaps worden altijd vloeiend gemaakt.

Het gebruik van `StageQuality.MEDIUM` biedt vaak voldoende kwaliteit voor toepassingen op mobiele apparaten en in sommige gevallen biedt ook `StageQuality.LOW` voldoende kwaliteit. Vanaf Flash Player 8 kan anti-aliased tekst accuraat worden gerenderd, zelfs als de Stage-kwaliteit is ingesteld op `LOW`.

Opmerking: *Op sommige mobiele apparaten wordt, zelfs als de kwaliteit is ingesteld op `HIGH`, de instelling `MEDIUM` gebruikt voor betere prestaties in Flash Player-toepassingen. Vaak geeft het gebruik van de kwaliteit HIGH echter geen groot verschil omdat mobiele schermen gewoonlijk een hogere dpi hebben. (De dpi kan variëren, afhankelijk van het apparaat.)*

In de volgende afbeelding is de filmkwaliteit ingesteld op `MEDIUM` en is de tekstrendering ingesteld op Anti-Alias for Animation:

Stage-kwaliteit is MEDIUM (normaal) en tekstrendering is ingesteld op Anti-Alias for Animation

De instelling van de Stage-kwaliteit heeft invloed op de tekstkwaliteit omdat de geschikte instelling voor tekstrendering niet wordt gebruikt.

De runtime staat u toe het renderen van tekst in te stellen op Anti-alias voor leesbaarheid. Deze instelling garandeert een perfecte kwaliteit van tekst (met anti-aliasing), ongeacht de Stage-kwaliteit die u gebruikt:

Stage-kwaliteit is LOW (laag) en tekstrendering is ingesteld op Anti-aliasing voor leesbaarheid.

Dezelfde renderingkwaliteit kan worden bekomen door de tekstrendering in te stellen op Bitmap Text (geen antialias).

Stage-kwaliteit is LOW (laag) en tekstrendering is ingesteld op Bitmap Text (geen antialias)

De twee laatste voorbeelden tonen dat u hoogwaardige tekst kunt verkrijgen, ongeacht de instelling voor Stage-kwaliteit. Deze functie is sinds Flash Player 8 beschikbaar en kan op mobiele apparatuur worden gebruikt. Vergeet niet dat Flash Player 10.1 op bepaalde apparaten automatisch overschakelt op `StageQuality.MEDIUM` om de prestaties te verbeteren.

Alfa-overvloeiing

Vermijd waar mogelijk het gebruik van de Alpha-eigenschap.

Vermijd ook het gebruik van effecten die alfa-overvloeiing vereisen, zoals vervagingseffecten, wanneer u de `alpha`-eigenschap gebruikt. Wanneer een weergaveobject alfa-overvloeiing gebruikt, moet de runtime de kleurwaarden van elk gestapelde weergaveobject en de achtergrondkleur combineren om de uiteindelijke kleur te bepalen. Zo kan alfa-overvloeiing meer vergen van de processor dan het tekenen van een dekkende kleur. Dit extra processorgebruik kan de prestaties op trage apparaten verminderen. Vermijd waar mogelijk het gebruik van de `Alpha`-eigenschap.

Meer Help-onderwerpen

"Het in cache plaatsen van bitmaps" op pagina 55

"Tekstobjecten renderen" op pagina 69

Framesnelheid van een toepassing

Gebruik in principe de laagst mogelijke framesnelheid voor betere prestaties..

De framesnelheid van een toepassing bepaalt hoeveel tijd er wordt uitgetrokken voor elke cyclus van toepassingscode en rendering, zoals beschreven in "Basisprincipes voor het uitvoeren van code in de runtime" op pagina 1. Bij een hogere framesnelheid worden animaties vloeiender afgespeeld. Maar als er geen sprake is van animatie of andere visuele veranderingen, is een hoge framesnelheid vaak niet nodig. Een hogere framesnelheid vergt meer CPU-cycli en stroom van de batterij dan een lagere snelheid.

Hieronder volgt een aantal algemene richtlijnen voor een geschikte standaardframesnelheid voor uw toepassing:

- Als u het Flex-framework gebruikt, laat u de standaardwaarde voor de aanvankelijke framesnelheid ongewijzigd.
- Als de toepassing animaties bevat, stelt u de framesnelheid in op ten minste 20 frames per seconde. Hoger dan 30 frames per seconde is vaak niet nodig.
- Als de toepassing geen animaties bevat, is een framesnelheid van 12 frames per seconde meestal voldoende.

De laagst mogelijke framesnelheid kan variëren afhankelijk van de huidige activiteit van de toepassing. Zie de volgende tip in "De framesnelheid van de toepassing dynamisch wijzigen" voor meer informatie.

Gebruik een lage framesnelheid als video de enige dynamische inhoud in de toepassing is.

Video-inhoud die wordt geladen wordt in de oorspronkelijke framesnelheid afgespeeld, onafhankelijk van de framesnelheid van de toepassing. Als de toepassing geen animaties of andere snel veranderende visuele inhoud bevat, leidt het gebruik van een lage framesnelheid niet tot een minder prettige gebruikerservaring.

De framesnelheid van de toepassing dynamisch wijzigen.

De aanvankelijke framesnelheid van de toepassing stelt u·in de project- of compilerinstellingen in, maar de framesnelheid hoeft niet altijd deze waarde te hebben. U kunt de framesnelheid wijzigen door de eigenschap `Stage.frameRate` (of de eigenschap `WindowedApplication.frameRate` in Flex) in te stellen.

Pas de framesnelheid aan bij de huidige behoeften van de toepassing. Verlaag de framesnelheid bijvoorbeeld gedurende een periode waarin de toepassing geen animaties uitvoert. Wanneer een animatieovergang gaat beginnen, verhoogt u de framesnelheid. Als de toepassing op de achtergrond wordt uitgevoerd (en geen focus heeft), kunt u de framesnelheid zelfs nog verder verlagen. De gebruiker heeft zijn aandacht waarschijnlijk bij een andere toepassing of taak.

Hieronder volgt een aantal richtlijnen die u kunt gebruiken als uitgangspunt voor het bepalen van de juiste framesnelheid voor verschillende soorten·activiteiten:

- Als u het Flex-framework gebruikt, laat u de standaardwaarde voor de aanvankelijke framesnelheid ongewijzigd.
- Tijdens het afspelen van animaties stelt u de framesnelheid in op ten minste 20 frames per seconde. Hoger dan 30 frames per seconde is vaak niet nodig.
- Wanneer er geen animatie wordt afgespeeld, is een framesnelheid van 12 frames per seconde meestal voldoende.

- Video die wordt·geladen wordt op de oorspronkelijke framesnelheid afgespeeld, onafhankelijk van de framesnelheid van de toepassing. Als video de enige bewegende inhoud in de toepassing is, is een framesnelheid van 12 frames per seconde meestal voldoende.
- Wanneer de toepassing geen invoerfocus heeft, is een framesnelheid van 5 frames per seconde meestal voldoende.
- Wanneer een AIR-toepassing niet zichtbaar is, is een framesnelheid van 2 frames per seconde of minder meestal voldoende. Deze richtlijn is bijvoorbeeld van toepassing als een toepassing wordt geminimaliseerd. De richtlijn is ook van toepassing op desktopapparaten als de eigenschap `visible` van het eigen venster is ingesteld op `false`.

Voor toepassingen die in Flex zijn ontwikkeld geldt dat de klasse spark.components. WindowedApplication ingebouwde ondersteuning biedt voor het dynamisch wijzigen van de framesnelheid van de toepassing. De eigenschap `backgroundFrameRate` definieert de framesnelheid van de toepassing wanneer de toepassing niet actief is. De standaardwaarde is 1, waarmee de framesnelheid van een toepassing die met het Spark-framework is ontwikkeld wordt ingesteld op 1 frame per seconde. U kunt de achtergrondframesnelheid wijzigen door de eigenschap `backgroundFrameRate` in te stellen. U kunt een andere waarde voor deze eigenschap instellen of de waarde -1 instellen om het automatisch wijzigen van de framesnelheid uit te schakelen.

Raadpleeg de volgende artikelen voor meer informatie over het dynamisch wijzigen van de framesnelheid van een toepassing:

- Reducing CPU usage in Adobe AIR (Adobe Developer Center-artikel en voorbeeldcode van Jonnie Hallman)
- Writing well-behaved, efficient, AIR applications (artikel en voorbeeld van Arno Gourdol)

Grant Skinner heeft een klasse ontwikkeld voor het vertragen van de framesnelheid. Deze klasse kunt u in toepassingen gebruiken om de framesnelheid automatisch te verlagen wanneer deze op de achtergrond wordt uitgevoerd. Zie het artikel Idle CPU Usage in Adobe AIR and Flash Player van Grant op http://gskinner.com/blog/archives/2009/05/idle_cpu_usage.html voor meer informatie en om de broncode voor de klasse FramerateThrottler te downloaden.

Adaptieve framesnelheid

Als u een SWF-bestand compileert, stelt u een bepaalde framesnelheid in voor de film. In een beperkte omgeving met een lage processorsnelheid kan de framesnelheid soms verlagen tijdens het afspelen. Om een acceptabele framesnelheid voor de gebruiker te behouden slaat de runtime de rendering van sommige frames over. Het overslaan van de rendering van sommige frames zorgt ervoor dat de framesnelheid niet terugloopt tot een onaanvaardbare waarde.

Opmerking: *In dit geval slaat de runtime geen frames over, maar alleen de rendering van inhoud van frames. Er wordt nog steeds code uitgevoerd en de weergavelijst wordt bijgewerkt, maar de updates worden niet weergegeven op het scherm. Er bestaat geen manier om een fps-drempelwaarde op te geven die aangeeft hoeveel frames er worden overgeslagen als de runtime de framesnelheid niet stabiel kan houden.*

Het in cache plaatsen van bitmaps

Gebruik, waar mogelijk, de functie voor het in cache plaatsen van bitmaps voor complexe vectorcontent.

U kunt voor een goede optimalisering zorgen door de functie voor het in de cache plaatsen van bitmaps te gebruiken. Met deze functie wordt een vectorobject in de cache geplaatst, als bitmap intern gerenderd en wordt deze bitmap voor rendering gebruikt. Het resultaat kan een enorme verhoging van de renderprestaties zijn, maar er is mogelijk een grote hoeveelheid geheugen voor nodig. Gebruik de functie voor het in cache plaatsen van bitmaps voor complexe vectorcontent, zoals complexe overgangen of tekst.

Het inschakelen van bitmapcaches voor een geanimeerd object met complexe vectorillustraties (zoals tekst of verlopen) verbetert de prestaties. Maar als het in de cache plaatsen van bitmaps·is ingeschakeld voor een weergaveobject, zoals een filmclip waarvan de tijdlijn wordt afgespeeld, krijgt u het tegenovergesteld resultaat. Voor elk frame moet de runtime de bitmap in de cache bijwerken en deze vervolgens opnieuw op het scherm tekenen. Hiervoor is een groot aantal CPU-cycli nodig. De functie voor bitmapcaching heeft alleen voordelen wanneer de bitmap in de cache eenmaal gegenereerd kan worden en vervolgens kan worden gebruikt zonder dat de bitmap hoeft te worden bijgewerkt.

Als u bitmapcaching inschakelt voor een Sprite-object, kunt u het object verplaatsen zonder dat de runtime de bitmap in de cache opnieuw hoeft te genereren. Het wijzigen van de `x`- en `y`-eigenschappen van het object leidt niet tot het opnieuw genereren van het object. Wanneer echter wordt geprobeerd het object te roteren, te schalen of de alfawaarde van het object te wijzigen, moet de runtime de bitmap in de cache opnieuw genereren. Dit heeft een nadelige invloed op de prestaties.

Opmerking: *De eigenschap `DisplayObject.cacheAsBitmapMatrix` die beschikbaar is in AIR of Packager for iPhone, kent deze beperking niet. Als u de eigenschap `cacheAsBitmapMatrix` gebruikt, kunt u een weergaveobject roteren, schalen, schuintrekken en de alfawaarde van een object wijzigen zonder dat een bitmap opnieuw wordt gegenereerd.*

Een in cache geplaatste bitmap kan aanzienlijk meer geheugen gebruiken dan een standaardfilmclipinstantie. Als de filmclip in het werkgebied bijvoorbeeld 250 x 250 pixels bedraagt, zal deze ongeveer 250 KB gebruiken als deze in de cache is geplaatst in plaats van 1 KB als deze niet in de cache is geplaatst.

Het volgende voorbeeld toont een Sprite-object dat een afbeelding van een appel bevat. De volgende klasse is verbonden met het symbool van de appel:

```
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
```

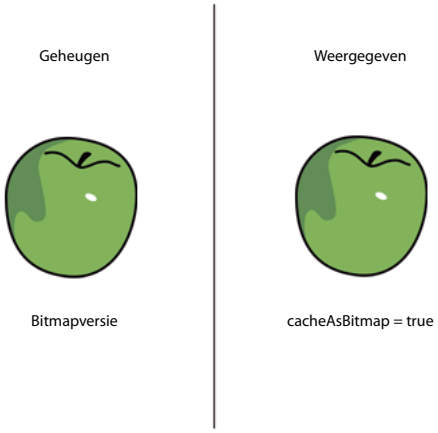
```
        {
            removeEventListener(Event.ENTER_FRAME,handleMovement);
        }

        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();
        }
    }
}
```

De code gebruikt de Sprite-klasse in plaats van de MovieClip-klasse omdat geen tijdlijn nodig is voor elke appel. Voor de beste prestaties gebruikt u het meest lichtgewichte object mogelijk. Vervolgens wordt de klasse geïnstantieerd met de volgende code:

```
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);
        }
    }
}
```

Als de gebruiker op de muis klikt, worden de appels gemaakt zonder caching. Als de gebruiker op de C-toets (toetscode 67) drukt, worden de appelvectoren als bitmaps in de cache geplaatst en op het scherm weergegeven. Met deze techniek worden de renderprestaties aanzienlijk verhoogd, zowel op het bureaublad als op mobiele telefoons, wanneer de processor langzaam is.

Hoewel het gebruik van de functie voor het in cache plaatsen van bitmaps de renderprestaties verhoogt, zorgt deze functie mogelijk ook voor een hoge geheugenconsumptie. Zodra een object in de cache wordt geplaatst, wordt het oppervlak ervan vastgelegd als een transparante bitmap en in het geheugen opgeslagen, zoals weergegeven in het volgende diagram:

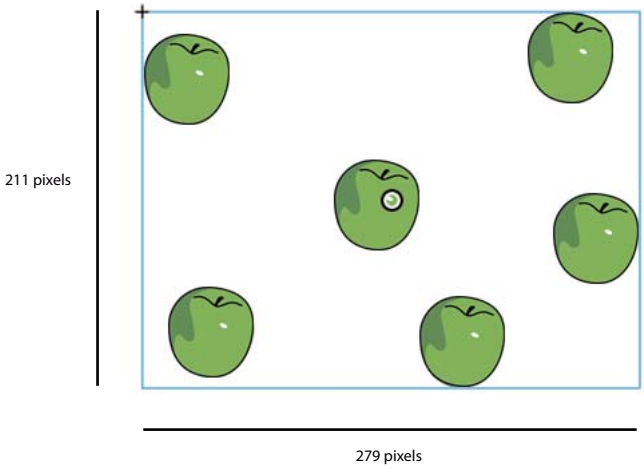


Object en oppervlakbitmap opgeslagen in het geheugen

Met Flash Player 10.1 en AIR2.5 wordt het geheugenverbruik geoptimaliseerd op de manier die wordt beschreven in het hoofdstuk "Filters en dynamische bitmapontladingen" op pagina 20. Als een weergaveobject is verborgen of niet zichtbaar is op het scherm en de bijbehorende bitmap in het geheugen een tijdje niet wordt gebruikt, wordt deze vrijgegeven.

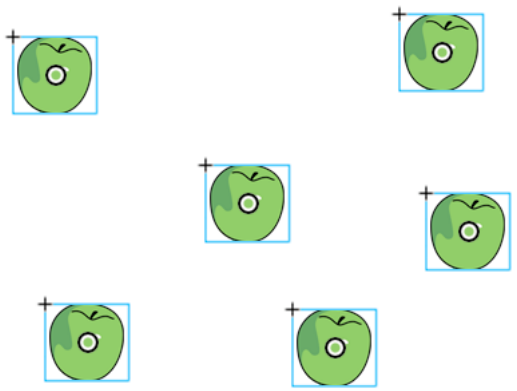
Opmerking: *Als de eigenschap* `opaqueBackground` *van het weergaveobject is ingesteld op een bepaalde kleur, neemt de runtime aan dat het weergaveobject ondoorzichtig is. Wanneer het object wordt gebruikt met de eigenschap* `cacheAsBitmap`*, maakt de runtime een niet-transparante 32-bits bitmap in het geheugen. Het alfakanaal wordt ingesteld op 0xFF, wat de prestaties verhoogt, omdat er geen transparantie vereist is om de bitmap op het scherm te tekenen. Als alfa-overvloeiing kan worden voorkomen, gaat de rendering nog sneller. Als de huidige schermdiepte beperkt is tot 16 bits, wordt de bitmap in het geheugen opgeslagen als een 16-bits afbeelding. Gebruik van de eigenschap* `opaqueBackground` *activeert bitmapcaching niet op impliciete wijze.*

Om geheugen te besparen, gebruikt u de eigenschap `cacheAsBitmap` en activeert u deze op elk weergaveobject in plaats van op de container. Het in cache plaatsen van bitmaps op de container activeren zorgt ervoor dat de uiteindelijke bitmap veel meer geheugen nodig heeft omdat een transparante bitmap wordt gemaakt met afmetingen van 211 x 279 pixels. De afbeelding gebruikt ongeveer 229 KB geheugen:



Het in cache plaatsen van bitmaps op de container activeren

Als u de container in het cachegeheugen plaatst, loopt u bovendien het risico dat de hele bitmap in het geheugen wordt bijgewerkt wanneer een appel op een frame wordt verplaatst. Als u het in cache plaatsen van bitmaps activeert op afzonderlijke instanties, worden zes oppervlakken van 7 KB in het geheugen opgeslagen, wat een totaal van 42 KB betekent.



Het in cache plaatsen van bitmaps op instanties activeren

Als u toegang krijgt tot elke appelinstantie via de weergavelijst en de methode `getChildAt()` aanroept, worden referenties opgeslagen in een Vector-object voor eenvoudigere toegang:

```
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);
        }
    }
}
```

Vergeet niet dat bitmapcaching tot betere rendering leidt als de cachecontent niet op elk frame wordt geroteerd, geschaald of gewijzigd. Bij elke andere transformatie behalve translatie op de X- en Y-as wordt de rendering echter niet verbeterd. In deze gevallen werkt de Flash Player de bitmapkopie in de cache bij voor elke transformatie die plaatsvindt op het weergaveobject. Het bijwerken van de kopie in de cache kan resulteren in een hoog verbruik van de processor en de batterij en in langzame prestaties. En weer geldt dat de eigenschap `cacheAsBitmapMatrix` die beschikbaar is in AIR of Packager for iPhone, deze beperking niet kent.

De volgende code wijzigt de alfawaarde in de bewegingsmethode waardoor de dekking van de appel op elke frame wordt gewijzigd.

```
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();
}
```

Het in cache plaatsen van bitmaps gebruiken leidt tot verminderde prestaties. Als de alfawaarde wordt gewijzigd, wordt de runtime gedwongen de bitmap in de cache bij te werken wanneer de alfawaarde wordt gewijzigd.

Filters vertrouwen op bitmaps die worden bijgewerkt wanneer de afspeelknop van een filmclip in de cache wordt verplaatst. Als dus een filter wordt gebruikt, wordt de eigenschap `cacheAsBitmap` automatisch ingesteld op `true`. De volgende afbeelding illustreert een geanimeerde filmclip:

Filmclip met animatie

Vermijd het gebruik van filters op geanimeerde inhoud omdat dit voor prestatieproblemen kan zorgen. In de volgende afbeelding voegt de ontwerper een slagschaduwfilter toe:

Filmclip met animatie en filter voor slagschaduw

Het resultaat is dat de bitmap opnieuw gegenereerd moet worden wanneer de tijdlijn in de filmclip wordt afgespeeld. De bitmap moet ook opnieuw gegenereerd worden als de content op andere wijze wordt gewijzigd dan via een eenvoudige x- of y-transformatie. Elke frame dwingt de runtime de bitmap opnieuw te tekenen, zodat meer processorbronnen moeten worden gebruikt, de prestaties afnemen en het batterijverbruik toeneemt.

Paul Trani verschaft in de volgende educatieve video's voorbeelden van de manier waarop u met Flash Professional en ActionScript bitmaps kunt gebruiken om afbeeldingen te optimaliseren:

- Afbeeldingen optimaliseren
- Afbeeldingen optimaliseren met ActionScript

Transformatiematrices voor bitmaps in de cache in AIR

Stel de eigenschap `cacheAsBitmapMatrix` in wanneer u bitmaps in de cache in AIR-toepassingen voor mobiele apparaten gebruikt.

In het AIR-profiel voor mobiele apparaten kunt u een Matrix-object toewijzen aan de eigenschap `cacheAsBitmapMatrix` van een weergaveobject. Wanneer u deze eigenschap instelt, kunt u elke tweedimensionale transformatie op een object toepassen zonder dat de bitmap in de cache opnieuw hoeft te worden gegenereerd. U kunt ook de alfa-eigenschap wijzigen zonder zonder dat de bitmap in de cache opnieuw hoeft te worden gegenereerd. De eigenschap `cacheAsBitmap` moet ook op `true` zijn ingesteld en voor het object mogen geen 3D-eigenschappen zijn ingesteld.

Als u de eigenschap `cacheAsBitmapMatrix` instelt, wordt de bitmap in de cache gegenereerd, zelfs als het weergaveobject buiten het scherm is geplaatst, niet zichtbaar is of als de eigenschap `visible` van het object is ingesteld op `false`. Als u de eigenschap `cacheAsBitmapMatrix` opnieuw instelt met behulp van een matrixobject met een andere transformatie, wordt de bitmap in de cache ook opnieuw gegenereerd.

De matrixtransformatie die u toepast op de eigenschap `cacheAsBitmapMatrix` wordt toegepast op het weergaveobject op het moment dat het wordt gerenderd in de bitmapcache. Als de transformatie een schaal 2 bevat, is de gerenderde bitmap dus twee keer zo groot als de gerenderde vector. De renderer past de omgekeerde transformatie toe op de bitmap in de cache, zodat de uiteindelijke weergave er hetzelfde uitziet. U kunt de bitmap in de cache kleiner maken om het geheugengebruik te verminderen. Dit gaat mogelijk ten koste van de renderprecisie. U kunt in sommige gevallen de bitmap ook groter maken om de renderkwaliteit te verbeteren. In dit geval wordt er meer geheugen gebruikt. In het algemeen kunt u echter beter een identiteitsmatrix gebruiken. Dit is een matrix die geen transformatie toepast om wijzigingen in uiterlijk te voorkomen, zoals in het volgende voorbeeld wordt getoond:

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

Als de eigenschap `cacheAsBitmapMatrix` eenmaal is ingesteld, kunt u het object schalen, roteren, schuintrekken en het object omzetten zonder dat bitmaps opnieuw worden gegenereerd.

U kunt ook de alfawaarde wijzigen binnen het bereik van 0 en 1. Als u de alfawaarde via de eigenschap `transform.colorTransform` wijzigt in een kleurtransformatie, moet de alfawaarde die in het transformatieobject wordt gebruikt tussen 0 en 255 liggen. Als u de kleurtransformatie op een andere manier wijzigt, wordt de bitmap in de cache opnieuw gegenereerd.

Stel altijd de eigenschap `cacheAsBitmapMatrix` in als u de eigenschap `cacheAsBitmap` instelt op `true` in inhoud die u voor mobiele apparaten maakt. Houd er rekening mee dat dit het volgende nadeel kan hebben. Nadat een object is geroteerd, geschaald of scheefgetrokken, kan het uiteindelijke renderen het schalen van bitmaps of aliasing van artefacten veroorzaken in vergelijking met een normale vectorrendering.

Het handmatig in cache plaatsen van bitmaps


Gebruik de BitmapData-klasse om het in cache plaatsen van bitmaps aan te passen.

Het volgende voorbeeld gebruikt één gerasterde bitmapversie van een weergaveobject opnieuw en verwijst naar hetzelfde BitmapData-object. Wanneer elk weergaveobject wordt geschaald, wordt het originele BitmapData-object in het geheugen niet bijgewerkt en niet opnieuw getekend. Met deze methode worden processorbronnen bespaard en worden toepassingen sneller uitgevoerd. Wanneer een weergaveobject wordt geschaald, wordt de ingesloten bitmap uitgerekt.

Hier is de bijgewerkte `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();
        }
    }
}
```

De alfawaarde wordt nog steeds op elk frame gewijzigd. De volgende code geeft de oorspronkelijke bronbuffer door aan elke BitmapApple-instantie:

```
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);
}
```

Deze techniek vergt slechts weinig van het geheugen, aangezien er slechts één cachebitmap wordt gebruikt door het geheugen die door alle `BitmapApple`-instanties wordt gedeeld. Bovendien wordt de originele bronbitmap nooit bijgewerkt ondanks eventuele wijzigingen die worden gemaakt in de `BitmapApple`-instanties, zoals alfa, roteren en schalen. Het gebruik van deze techniek voorkomt een vermindering van de prestaties.

Voor een vloeiende uiteindelijke bitmap stelt u de eigenschap `smoothing` in op `true`:

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

    smoothing = true;

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

Aanpassing van de Stage-kwaliteit kan de prestaties ook verbeteren. Stel de Stage-kwaliteit in op `HIGH` vóór de rastering wordt uitgevoerd en schakel nadien over naar `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 );
}
```

Een handige techniek om content met antialiasing op het scherm te krijgen, is het in- en uitschakelen van de Stage-kwaliteit vóór en na het tekenen van de vector naar een bitmap. Deze techniek kan bijzonder effectief zijn, ongeacht de uiteindelijke Stage-kwaliteit. U kunt bijvoorbeeld een bitmap met antialiasing maken met tekst met antialising, ook als de Stage-kwaliteit is ingesteld op `LOW`. Deze techniek is niet mogelijk met de eigenschap `cacheAsBitmap`. In dat geval wordt door het instellen van de Stage-kwaliteit op `LOW` de vectorkwaliteit bijgewerkt, waardoor de bitmapoppervlakken in het geheugen en ook de uiteindelijke kwaliteit worden bijgewerkt.

Gedragingen isoleren

Isoleer voor zover mogelijk de gebeurtenis `Event.ENTER_FRAME` in één handler.

De code kan verder worden geoptimaliseerd door de gebeurtenis `Event.ENTER_FRAME` in de `Apple`-klasse te isoleren in één handler. Met deze techniek bespaart u CPU-bronnen. Het volgende voorbeeld illustreert deze andere aanpak, waarbij de BitmapApple-klasse de verplaatsingen niet meer verwerkt.

```
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;
        }
}
```

De volgende code instantieert de appels en verwerkt hun bewegingen in één handler:

```
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;
            }
        }
}
```

Het resultaat is één `Event.ENTER_FRAME`-gebeurtenis die de beweging afhandelt in plaats van 200 handlers die elke appel bewegen. U kunt de volledige animatie eenvoudig pauzeren, hetgeen handig kan zijn in een game.

Een eenvoudige game kan bijvoorbeeld gebruikmaken van de volgende handler:

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

In de volgende stap moeten de appels communiceren met de muis of het toetsenbord. Hiervoor moet de `BitmapApple`-klasse worden gewijzigd.

```
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);
        }
}
```

Het resultaat is dat de `BitmapApple`-klassen interactief zijn, zoals traditionele Sprite-objecten. De instanties worden echter gekoppeld aan één bitmap waarvan het aantal bits niet wordt bijgewerkt wanneer de weergaveobjecten worden getransformeerd.

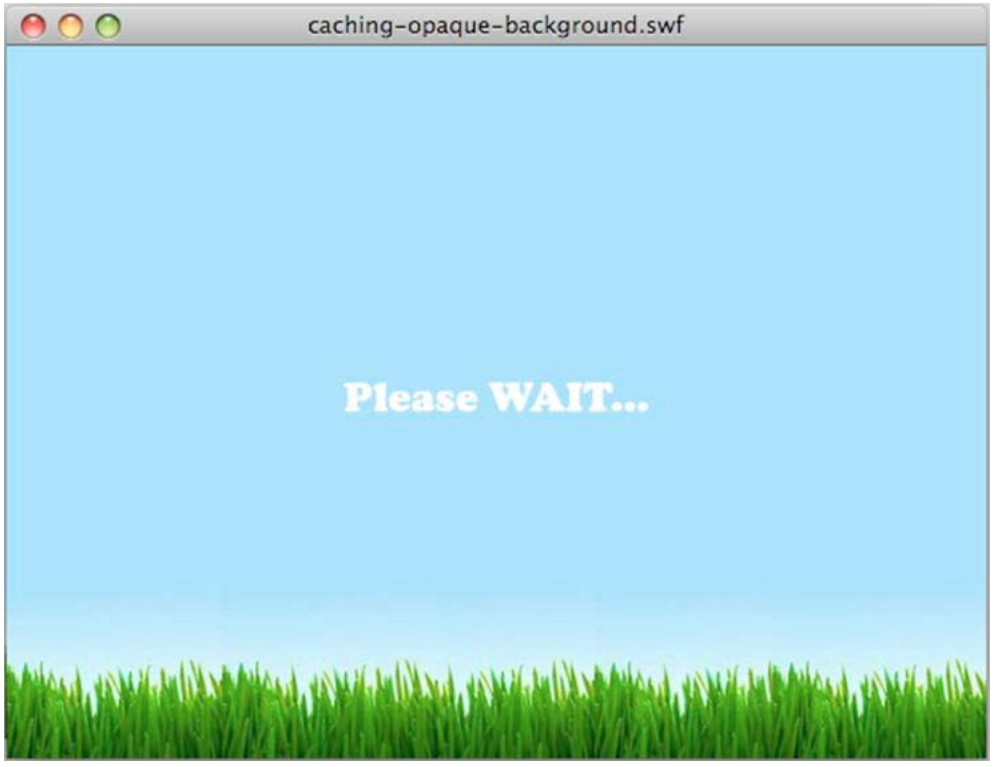
Tekstobjecten renderen

Gebruik de functie voor het in cache plaatsen van bitmaps en de eigenschap `opaqueBackground` om de rendering van tekst te verbeteren.

De Flash-tekstengine biedt een aantal fraaie optimalisaties. Bij talloze klassen is echter de weergave van één regel tekst vereist. Om deze reden vereist het maken van een bewerkbaar tekstveld met de TextLine-klasse een grote hoeveelheid geheugen en veel regels ActionScript-code. De TextLine-klasse wordt het best gebruikt voor statische en niet-bewerkbare tekst, waarvoor de rendering sneller gaat en minder geheugen vereist is.

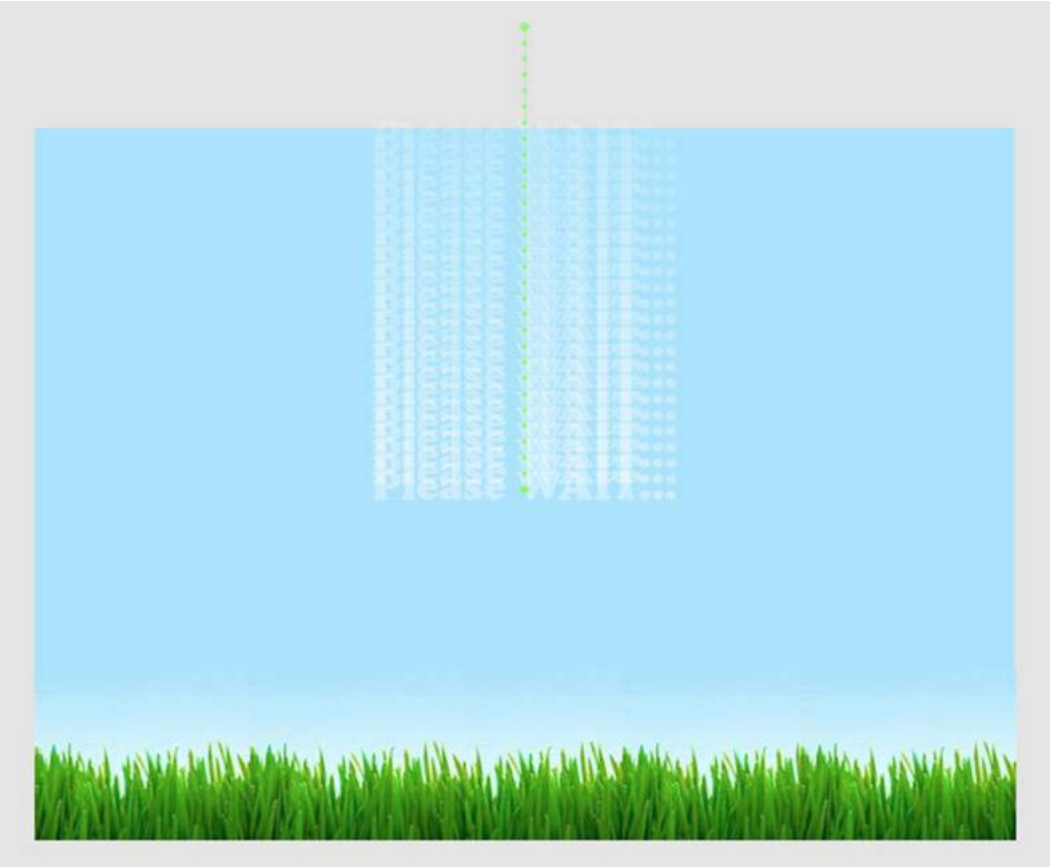
Met de functie voor het in cache plaatsen van bitmaps kunt u vectorinhoud als bitmaps in de cache plaatsen om de renderprestaties te verbeteren. Deze functie is nuttig voor complexe vectorinhoud en ook wanneer deze wordt gebruikt met tekstinhoud waarvoor de verwerking moet worden gerenderd.

Het volgende voorbeeld toont hoe de functie voor het in cache plaatsen van bitmaps en de eigenschap `opaqueBackground` kunnen worden gebruikt om de renderprestaties te verbeteren. De volgende afbeelding toont een typisch welkomstscherm dat kan worden weergegeven wanneer een gebruiker wacht tot iets is geladen:



Welkomstscherm

De volgende afbeelding toont de versnelling die programmatisch wordt toegepast op het TextField-object. De tekst neemt traag af van boven aan de scène tot het midden van de scène:



Versnelling van tekst

De volgende code maakt de versnelling. De variabele `preloader` slaat het huidige doelobject op om de prestaties schadende eigenschaplook-ups te minimaliseren:

```
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 );
}
```

De functie `Math.abs()` kan inline worden geplaatst om het aantal functieaanroepen te reduceren en de prestaties verder te verbeteren. U kunt het best het int-type gebruiken voor de eigenschappen `destX` en `destY` zodat u vaste-puntwaarden hebt. Als u het int-type gebruikt, kunt u de perfecte magnetische pixeluitlijning verkrijgen zonder dat u de waarden handmatig moet afronden met behulp van langzame methoden zoals `Math.ceil()` of `Math.round()`. Deze code rondt de coördinaten niet af op int, omdat het object niet vloeiend zou worden verplaatst als de waarden op constante wijze zouden worden afgerond. Het object kan schokkerige bewegingen vertonen, omdat de coördinaten op elk frame op het dichtstbijzijnde gehele getal worden uitgelijnd. Deze techniek kan echter handig zijn wanneer u een uiteindelijke positie instelt voor een weergaveobject. Gebruik de volgende code niet:

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

De volgende code is veel sneller:

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

De vorige code kan nog verder worden geoptimaliseerd door gebruik te maken van operators voor het bitsgewijze verplaatsen om de waarden te splitsen:

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

Met de functie voor het in cache plaatsen van bitmaps kan de runtime eenvoudiger objecten renderen door gebruik te maken van dynamische bitmaps. In het huidige voorbeeld is de filmclip die het TextField-object bevat, in de cache geplaatst:

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

Een aanvullende manier om de prestaties te verbeteren is door de alfatransparantie te verwijderen. Alfatransparantie is een bijkomende belasting voor de runtime bij het tekenen van transparante bitmapafbeeldingen, zoals in de vorige code. U kunt de eigenschap `opaqueBackground` gebruiken om dit probleem te omzeilen door een kleur als achtergrond op te geven.

Wanneer u de eigenschap `opaqueBackground` gebruikt, gebruikt het bitmapoppervlak in het geheugen nog steeds 32 bits. De alfaverschuiving wordt echter ingesteld op 255 en er wordt geen transparantie toegepast. Het resultaat hiervan is dat de eigenschap `opaqueBackground` het geheugenverbruik niet vermindert, maar dat de renderprestaties wel worden verbeterd wanneer de functie voor het in cache plaatsen van bitmaps wordt gebruikt. De volgende code bevat alle optimalisaties:

```
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 );
}
```

De animatie is nu geoptimaliseerd en het in cache plaatsen van bitmaps is geoptimaliseerd door de transparantie te verwijderen: Denk eraan dat u de Stage-kwaliteit op `LOW` en `HIGH` kunt instellen tijdens de verschillende stadia van de animatie terwijl de functie voor het in cache plaatsen van bitmaps wordt gebruikt:

```
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 dit geval wordt door het wijzigen van de Stage-kwaliteit de runtime gedwongen om het bitmapoppervlak op het TextField-object opnieuw te genereren zodat het overeenkomt met de huidige Stage-kwaliteit. Daarom wijzigt u best de Stage-kwaliteit niet terwijl de functie voor het in cache plaatsen van bitmaps wordt gebruikt.

Een handmatige aanpak van het in cache plaatsen van bitmaps had hier kunnen worden gebruikt. Om de eigenschap `opaqueBackground` te simuleren kan de clip op een niet-transparant BitmapData-object worden getekend. Dit dwingt de runtime niet om het bitmapoppervlak opnieuw te genereren.

Deze techniek werkt goed voor inhoud die niet verandert in de loop van de tijd. Als de inhoud van het tekstveld echter kan worden veranderd, kunt u een andere strategie overwegen. Beeld u bijvoorbeeld een tekstveld in dat constant wordt bijgewerkt met een percentage dat voorstelt hoeveel van de toepassing al is geladen. Als het tekstveld, of het weergaveobject dat het bevat, als bitmap in de cache is geplaatst, moet het oppervlak worden gegenereerd telkens als de inhoud wordt gewijzigd. U kunt in dit geval bitmaps niet handmatig in cache plaatsen, omdat de content van het weergaveobject constant verandert. Deze constante wijziging zou betekenen dat u de `BitmapData.draw()`-methode handmatig moet aanroepen om de bitmap in de cache bij te werken.

Houd er rekening mee dat vanaf Flash Player 8 (en AIR 1.0) een tekstveld waarvan de rendering is ingesteld op Anti-Alias voor leesbaarheid perfect anti-aliased blijft, ongeacht de waarde van de Stage-kwaliteit. Deze aanpak verbruikt minder geheugen, maar vereist meer processorverwerking en de rendering verloopt ook langzamer dan met de functie voor het in cache plaatsen van bitmaps.

De volgende code maakt gebruik van deze aanpak:

```
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 );
    }
}
```

Het gebruik van deze optie (Anti-Alias for Readability) voor tekst in beweging wordt niet aangeraden. Wanneer de tekst wordt geschaald, zorgt deze optie ervoor dat de tekst probeert uitgelijnd te blijven, wat een verplaatsingseffect veroorzaakt. Als de inhoud van het weergaveobject echter constant verandert en u geschaalde tekst nodig hebt, kunt u de prestaties in mobiele toepassingen verbeteren door de kwaliteit in te stellen op `LOW`. Wanneer de beweging is voltooid, schakelt de kwaliteit opnieuw om naar `HIGH`.

GPU

GPU-rendering in Flash Player-toepassingen

Een belangrijke nieuwe functie van Flash Player 10.1 is dat de GPU gebruikt kan worden om grafische content te renderen op mobiele apparatuur. Voorheen werden afbeeldingen uitsluitend gerenderd via de CPU. Gebruik van de GPU optimaliseert rendering van filters, bitmaps, video en tekst. Vergeet niet dat GPU-rendering niet altijd even nauwkeurig is als softwarerendering. Content kan enigszins blokkerig overkomen wanneer de hardwarerenderer wordt gebruikt. Bovendien kent Flash Player 10.1 een beperking die voorkomt dat Pixel Bender-effecten op het scherm worden gerenderd. Bij gebruik van hardwareversnelling kunnen deze effecten worden gerenderd als een zwart vierkantje.

Hoewel Flash Player 10 over een GPU-versnellingsfunctie beschikte, werd de GPU niet gebruikt voor het berekenen van de afbeeldingen. De GPU werd uitsluitend gebruikt om afbeeldingen naar het scherm te verzenden. In Flash Player 10.1 wordt de GPU gebruikt om de afbeeldingen te berekenen, waarmee de rendersnelheid aanzienlijk kan worden verbeterd. Dit ontlast bovendien de CPU, hetgeen nuttig is op apparaten met beperkte bronnen, zoals mobiele apparaten.

Teneinde optimale prestaties te bereiken, wordt de GPU-modus automatisch ingesteld wanneer inhoud wordt uitgevoerd op mobiele apparaten. Hoewel `wmode` niet langer ingesteld hoeft te worden op `gpu` om GPU-rendering te verkrijgen, kunt u GPU-versnelling uitschakelen wanneer u `wmode` instelt op `opaque` of `transparent`.

Opmerking: *Flash Player op de desktop gebruikt nog steeds de CPU om softwareredering uit te voeren. Softwarerendering wordt gebruikt omdat stuurprogramma's op de desktop aanzienlijk variëren en renderingsverschillen kunnen benadrukken. Er kunnen ook renderingsverschillen optreden tussen de desktop en enkele mobiele apparaten.*

GPU-rendering in AIR-toepassingen voor mobiele apparaten

Schakel hardwareversnelling voor afbeeldingen in een AIR-toepassing in door `<renderMode>gpu</renderMode>` in het toepassingsbeschrijvingsbestand op te nemen. U kunt de renderingmodes niet wijzigen tijdens runtime. Op desktopcomputers wordt de instelling `renderMode` genegeerd; GPU-versnelling voor afbeeldingen wordt momenteel niet ondersteund.

Beperkingen van de GPU-renderingmodus

Wanneer u de GPU-renderingmodus in AIR 2.5 gebruikt, bestaan de volgende beperkingen:

- Als de GPU een object niet kan renderen, wordt het object niet weergegeven. Er is geen fallback voor CPU-rendering.
- De volgende overvloeimodi worden niet ondersteund: laag, alfa, wissen, bedekking, fel licht, lichter en donkerder.
- Filters worden niet ondersteund.
- PixelBender wordt niet ondersteund.
- Veel GPU-eenheden hebben een maximale structuurgrootte van 1024 x 1024. In ActionScript wordt dit de maximale uiteindelijke gerenderde grootte van een weergaveobject na alle transformaties.
- Adobe raadt het gebruik van de GPU-renderingmodus in AIR-toepassingen waarin video wordt afgespeeld, niet aan.
- In de GPU-renderingmodus worden tekstvelden niet altijd verplaatst naar een zichtbare locatie wanneer het virtuele toetsenbord wordt geopend. Voer een van de volgende handelingen uit om ervoor te zorgen dat uw tekstveld zichtbaar is terwijl de gebruiker tekst invoert: Plaats het tekstveld in de bovenste helft van het scherm of verplaats het naar de bovenste helft als het veld de focus krijgt.

- De GPU-renderingmodus is uitgeschakeld voor sommige apparaten waarop de modus niet betrouwbaar werkt. Zie de AIR-ontwikkelaarsrelease voor de meest recente informatie.

GPU-renderingmodus: aanbevolen procedures

Als u de volgende richtlijnen in acht neemt, wordt de GPU-rendering sneller uitgevoerd:

- Beperk het aantal items dat zichtbaar is in het werkgebied. Het duurt enige tijd om items te renderen en samen te stellen met de andere items eromheen. Wanneer u een weergaveobject niet meer wilt weergeven, stelt u de eigenschap `visible` van het object in op `false`. Verplaatst het object niet zo maar naar een locatie buiten het werkgebied, plaats het niet achter een ander object en stel de eigenschap `alpha` niet in op 0. Als u het weergaveobject niet meer nodig hebt, verwijdert u het uit het werkgebied met `removeChild()`.
- Gebruik objecten opnieuw in plaats van ze te maken en ze vervolgens weer te verwijderen.
- Maak bitmapformaten die bijna even groot zijn, maar minder groot zijn dan 2^n bij 2^m bits. De afmetingen hoeven niet 2 tot een bepaalde macht te zijn, maar ze moeten dicht in de buurt van een macht van 2 liggen en mogen niet groter zijn. Een afbeelding van 31 bij 15 pixels wordt bijvoorbeeld sneller gerenderd dan een afbeelding van 33 bij 17 pixels. (31 en 15 zijn net iets minder dan een macht van 2, 32 en 16.)
- Stel, indien mogelijk, de parameter repeat in op false wanneer de methode `Graphic.beginBitmapFill()` wordt aangeroepen.
- Gebruik niet te veel kleuren. Gebruik de achtergrondkleur als achtergrond. Plaats grote lagen niet boven op elkaar. Elke pixel die moet worden getekend, heeft zijn prijs.
- Vermijd vormen met lange dunne punten, randen die zichzelf snijden of veel fijne details langs de randen. Het kost meer tijd om deze vormen te renderen dan weergaveobjecten met vloeiende randen.
- Beperk de grootte van weergaveobjecten.
- Schakel cacheAsBitMap en cacheAsBitmapMatrix in voor weergaveobjecten waarvan de afbeeldingen niet regelmatig worden bijgewerkt.
- Vermijd het gebruik van de teken-API van ActionScript (de klasse Graphics) bij het maken van afbeeldingen. Maak indien mogelijk die objecten statisch tijdens het ontwerpen.
- Schaal bitmapelementen tot de uiteindelijke grootte voordat u ze importeert.

De GPU-renderingmodus in AIR 2.0.3 voor mobiele apparaten

De GPU-rendering kent meer beperkingen in AIR-toepassingen voor mobiele apparaten die zijn gemaakt met de Packager for iPhone. De GPU wordt alleen gebruikt voor bitmaps, effen vormen en weegaveobjecten waarvoor de eigenschap `cacheAsBitmap` is ingesteld. Ook voor objecten waarvoor `cacheAsBitmap` en `cacheAsBitmapMatrix` is ingesteld, kan de GPU objecten renderen die roteren of schalen. De GPU wordt gelijktijdig voor andere weergaveobjecten gebruikt en dit resulteert meestal in matige renderingprestaties.

Tips voor het optimaliseren van de GPU-renderprestaties

Hoewel GPU-rendering de prestaties van SWF-inhoud aanzienlijk kan verbeteren, speelt het ontwerp van de inhoud een belangrijke rol. Het is goed mogelijk dat instellingen die altijd heel geschikt waren voor softwarerendering soms niet zo geschikt zijn voor GPU-rendering. De volgende tips kunnen u helpen goede prestaties te bereiken met GPU-rendering zonder negatieve invloed op de prestaties van softwarerendering.

Opmerking: *Mobiele apparaten met ondersteuning voor hardwarerendering openen SWF-inhoud vaak via het web. Met oog op de beste prestaties op alle schermen verdient het daarom aanbeveling de volgende tips op te volgen wanneer u SWF-inhoud maakt.*

- Vermijd het gebruik van `wmode=transparent` of `wmode=opaque` in HTML-insluitparameters. Deze modi kunnen de prestaties nadelig beïnvloeden. Ze kunnen ook leiden tot iets minder nauwkeurige synchronisatie tussen geluid en beeld, zowel in software- als in hardwarerendering. Bovendien bieden vele platforms geen ondersteuning voor GPU-rendering als deze modi zijn geactiveerd, hetgeen een bijzonder nadelige invloed heeft op de prestaties.
- Gebruik alleen de overvloeimodi Normaal en Alfa. Vermijd het gebruik van andere overvloeimodi, gebruik vooral de laagovervloeimodus niet. Niet alle overvloeimodi kunnen op getrouwe wijze worden gereproduceerd als ze geworden gerenderd met de GPU.
- Wanneer een GPU vectorafbeeldingen rendert, worden de afbeeldingen eerst onderverdeeld in netten van kleine driehoekjes voordat ze worden getekend. Dit wordt tessellatie genoemd. Tessellatie beïnvloedt de prestaties slechts weinig, maar hoe complexer de vorm, hoe groter deze invloed. U kunt de invloed op de prestaties minimaliseren door morphingvormen te vermijden, omdat GPU-rendering deze tesseleert op elk frame.
- Vermijd het gebruik van curven die zichzelf doorsnijden, bijzonder smalle gebieden met curven (zoals een smalle maansikkel) en ingewikkelde details langs de randen van een vorm. Het is lastig voor de GPU om dergelijke vormen te tesseleren in driehoekige netten. Dat valt te verklaren aan de hand van twee vectoren: een vierkant van 500 × 500 en een maansikkel van 100 × 10. Het is heel eenvoudig voor de GPU om het grote vierkant te renderen, dat bestaat immers gewoon uit twee driehoeken. Er zijn echter vele driehoeken nodig om de curve van de maansikkel te beschrijven. Daarom is het ingewikkelder om de vorm te renderen, ook al bestaat deze uit minder pixels.
- Vermijd grote schaalwijzigingen, aangezien dergelijke wijzigingen er ook toe kunnen leiden dat de GPU de afbeeldingen nogmaals tesseleert.
- Probeer overtekenen ook zoveel mogelijk te vermijden. Overtekenen verwijst naar het in lagen plaatsen van meerdere grafische elementen, zodat deze elkaar verbergen. Bij gebruik van de softwarerenderer wordt elke pixel slechts één keer getekend. In geval van softwarerendering worden de prestaties van de toepassing dus niet nadelig beïnvloed, ongeacht het aantal grafische elementen dat elkaar bedekt op de desbetreffende pixellocatie. De hardwarerenderer daarentegen tekent elke pixel voor elk element, ongeacht of andere elementen dat gebied bedekken of niet. Wanneer twee rechthoeken elkaar overlappen, tekent de hardwarerenderer het overlappende gebied twee keer, terwijl de softwarerenderer dat maar één keer doet.

  Op een bureaublad waarop gebruik wordt gemaakt van de softwarerenderer merkt u dus doorgaans niets van vertraagde prestaties wegens overtekenen. Een groot aantal overlappende vormen kan de prestaties van apparaten met GPU-rendering echter nadelig beïnvloeden. U kunt objecten beter uit de weergavelijst verwijderen in plaats van ze te verbergen.
- Vermijd ook het gebruik van een grote gevulde rechthoek als een achtergrond. Stel in plaats daarvan de achtergrondkleur van het werkgebied in.
- Vermijd, waar mogelijk, de standaardmodus voor het vullen van bitmaps door bitmaps te herhalen. U bereikt betere resultaten met de beperkingsmodus voor bitmaps.

Asynchrone bewerkingen

Gebruik liever asynchrone versies van bewerkingen in plaats van synchrone versies, voor zover mogelijk.

Synchrone bewerkingen worden uitgevoerd zodra de code dat aangeeft, en de code wacht tot deze bewerkingen zijn voltooid voordat er verder wordt gegaan. Dit betekent dus dat deze bewerkingen in de toepassingscodefase van de framelusbewerking worden uitgevoerd. Als een synchrone bewerking te lang duurt, wordt de framelusbewerking verlengd, wat er toe kan leiden dat de weergave hapert of lijkt vast te lopen.

Wanneer de code een asynchrone bewerking uitvoert, wordt deze niet altijd meteen uitgevoerd. De code en andere toepassingscode in de huidige uitvoeringsthread worden zonder onderbreking uitgevoerd. Zodra het mogelijk is wordt de bewerking uitgevoerd, en tegelijkertijd wordt ernaar gestreefd om renderingproblemen te voorkomen. Soms vindt de uitvoering op de achtergrond plaats en dus niet als deel van de framelusbewerking. Wanneer de bewerking is voltooid, wordt een gebeurtenis verzonden. Uw code kan naar aanleiding van die gebeurtenis een volgende taak uitvoeren.

Asynchrone bewerkingen worden gepland en gesplitst·om renderingproblemen te voorkomen. Hierdoor dragen asynchrone versies van bewerkingen·bij aan een snelle uitvoering van de toepassing zonder onderbrekingen. Zie “Waargenomen prestaties versus werkelijke prestaties” op pagina 3 voor meer informatie.

Bij het uitvoeren van asynchrone bewerkingen is er echter wel sprake van overhead. De werkelijke uitvoeringstijd kan langer duren voor asynchrone bewerkingen, vooral bij bewerkingen die in korte tijd worden voltooid.

In de runtime zijn veel bewerkingen van zichzelf synchroon of asynchroon, en kunt u niet zelf kiezen op welke wijze ze worden uitgevoerd. In Adobe AIR kunt u echter kiezen uit drie soorten bewerkingen die naar wens synchroon of asynchroon kunnen worden uitgevoerd:

- Bewerkingen van de klassen File en FileStream

  Een groot aantal bewerkingen van de klasse File kunnen synchroon of asynchroon worden uitgevoerd. De methoden voor het kopiëren of verwijderen van bestanden of directory's en het weergeven van de inhoud van directory's hebben allemaal een asynchrone versie. Bij deze methoden wordt het achtervoegsel 'Async' aan de naam van de asynchrone versie toegevoegd. Als u bijvoorbeeld een bestand asynchroon wilt verwijderen, roept u de methode `File.deleteFileAsync()` aan in plaats van de methode `File.deleteFile()`.

  Wanneer u een object FileStream gebruikt om te lezen van of te schrijven naar een bestand, bepaalt·de manier waarop het object FileStream wordt geopend of de bewerkingen asynchroon worden uitgevoerd. Gebruik de methode `FileStream.openAsync()` voor asynchrone bewerkingen. Het schrijven van gegevens wordt asynchroon uitgevoerd. Het lezen van gegevens wordt in delen uitgevoerd, waardoor de gegevens deel voor deel beschikbaar zijn. Met de synchrone methode van het object FileStream wordt het gehele bestand gelezen voordat de code verder wordt uitgevoerd.

- Lokale SQL-databasebewerkingen

  Wanneer u met een lokale SQL-database werkt, worden alle bewerking via een object SQLConnection uitgevoerd in de synchrone of asynchrone modus. Als u wilt opgeven dat bewerkingen asynchroon moeten worden uitgevoerd, opent u de verbinding met de database via de methode `SQLConnection.openAsync()` in plaats van de methode `SQLConnection.open()`. Wanneer databasebewerkingen asynchroon worden uitgevoerd, vinden deze op de achtergrond plaats. De database-engine wordt helemaal niet in de lusbewerking uitgevoerd, waardoor de databasebewerkingen meestal geen renderingproblemen tot gevolg hebben.

  Raadpleeg “Prestaties van SQL-databases” op pagina 91 voor meer informatie over het verbeteren van prestaties van de lokale SQL-database.

- Zelfstandige Pixel Bender-arceringen

Met de klasse ShaderJob kunt u een afbeelding of gegevensset uitvoeren via een Pixel Bender-arcering en hebt u toegang tot de ruwe resultaatgegevens. Wanneer de methode `ShaderJob.start()` wordt aangeroepen, wordt de arcering standaard asynchroon uitgevoerd. De uitvoering vindt plaats op de achtergrond, en maakt dus geen gebruik van de lusbewerking. Als u wilt afdwingen dat het object ShaderJob synchroon wordt uitgevoerd (dit wordt niet aanbevolen), geeft u de waarde `true` door aan de eerste parameter van de methode `start()`.

Naast deze ingebouwde mechanismen voor het asynchroon uitvoeren van code kunt u ook uw eigen code zo structureren dat deze asynchroon wordt uitgevoerd in plaats van synchroon. Als u code schrijft voor het uitvoeren van·een taak die mogelijk lang duurt, kunt u de code zo structureren dat deze in delen wordt uitgevoerd. Wanneer de code is opgedeeld, kan de rendering plaatsvinden tussen de uitvoering van de afzonderlijke blokken code door, waardoor er minder kans op renderingproblemen is.

Hieronder volgt een aantal technieken voor het splitsen·van code. Het algemene idee achter al deze technieken is dat de code zo wordt geschreven dat deze per keer slechts een deel van het werk uitvoert. U kunt bijhouden wat de code doet en wanneer deze stopt. U gebruikt een mechanisme, bijvoorbeeld een object Timer, om herhaaldelijk te controleren of er nog werk gedaan moet worden en om extra werk in delen uit te voeren, totdat al het werk is voltooid.

Er bestaan een paar patronen waarmee code zo gestructureerd wordt dat het werk wordt gesplitst. De volgende artikelen en codebibliotheken geven een beschrijving van deze patronen en bieden code waarmee u de patronen in uw toepassingen kunt implementeren:

- Asynchronous ActionScript Execution (artikel van Trevor McCauley met meer achtergrondinformatie en diverse implementatievoorbeelden)
- Parsing & Rendering Lots of Data in Flash Player (artikel van Jesse Warden met achtergrondinformatie en voorbeelden van twee benaderingen: het “builderpatroon” en “green threads”)
- Green Threads (artikel van Drew Cummins waarin de “green threads”-techniek wordt beschreven aan de hand van voorbeeldcode)
- greenthreads (opensource codebibliotheek van Charlie Hubbard voor het implementeren van “green threads” in ActionScript. Zie greenthreads Quick Start voor meer informatie.)
- Threads in ActionScript 3 op·http://www.adobe.com/go/learn_fp_as3_threads_nl(artikel van Alex Harui, met een voorbeeldimplementatie van de 'pseudo threading'-techniek)

Transparante vensters

Gebruik in AIR-toepassingen in plaats van een ondoorzichtig venster een ondoorzichtig rechthoekig toepassingsvenster.

Als u een ondoorzichtig venster wilt gebruiken als startvenster van een AIR-desktoptoepassing, stelt u de volgende waarde in het XML-toepassingsbeschrijvingsbestand in:

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

Voor vensters die door middel van toepassingscode worden gemaakt maakt u een object NativeWindowInitOptions, waarbij de eigenschap `transparent` is ingesteld op `false` (de standaardwaarde). Geef dit door aan de constructor NativeWindow terwijl het object NativeWindow wordt gemaakt:

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

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

Voor een Flex Window-component moet u ervoor zorgen dat de eigenschap transparent van de component is ingesteld op false (de standaardwaarde) voordat u de methode `open()` van het object Window aanroept.

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

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

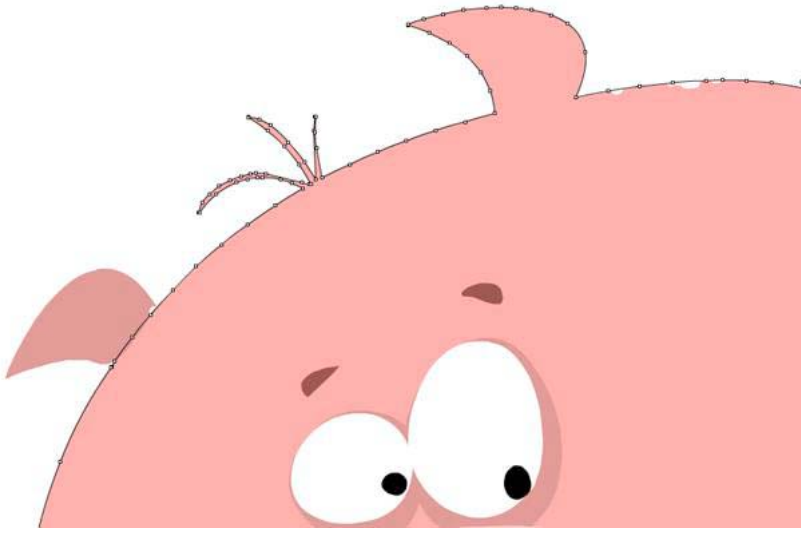
Bij een transparant venster kan het zijn dat een deel van het bureaublad van de gebruiker of andere toepassingsvensters achter het toepassingsvenster worden weergegeven. Hierdoor is·voor het renderen van een transparant venster meer geheugen nodig. Het renderen van een rechthoekig, ondoorzichtig venster, of dit nu gebruik maakt van een besturingssysteemchrome of een aangepast chrome, vormt een·minder zware belasting.

Gebruik alleen een transparant venster als u een niet-rechthoekige weergave wilt of wilt dat de inhoud op de achtergrond door het toepassingsvenster wordt weergegeven.

Vectorvormen vloeiend maken

Vloeiende vormen verbeteren de renderprestaties.

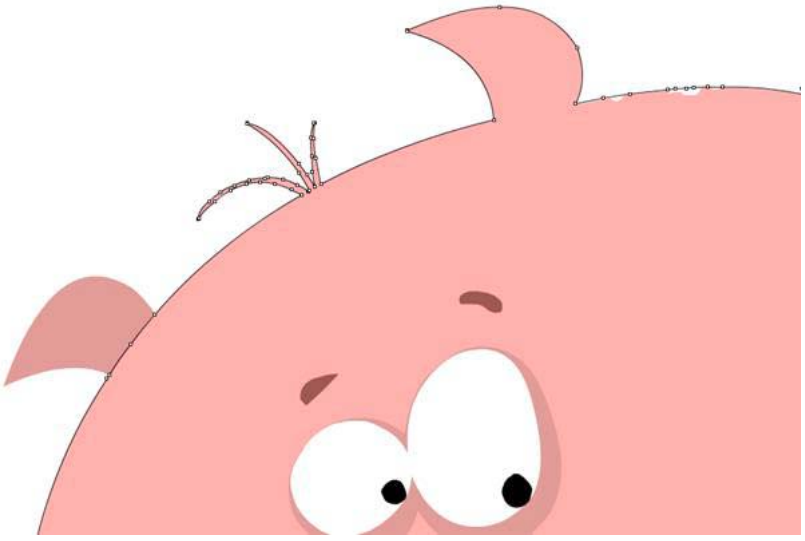
Anders dan voor bitmapinhoud, zijn voor vectorinhoud veel berekeningen nodig, vooral als gaat om verlopen en complexe paden met veel besturingspunten. Zorg er als ontwerper of ontwikkelaar voor dat vormen voldoende geoptimaliseerd zijn. In de volgende afbeelding ziet u niet-vereenvoudigde paden met veel besturingspunten.



Niet-geoptimaliseerde paden

Met het gereedschap Vloeiend in Flash Professional kunt u extra besturingspunten verwijderen. Een gelijksoortig gereedschap is beschikbaar in Adobe® Illustrator®. Het totaal aantal punten en paden kan worden bekeken in het deelvenster Documentinfo.

Bij het vloeiend maken worden extra besturingspunten verwijderd, waardoor het uiteindelijke SWF-bestand kleiner wordt en renderprestaties worden verbeterd. In de volgende afbeelding ziet u dezelfde paden nadat deze vloeiend zijn gemaakt.



Geoptimaliseerde paden

Zolang u paden niet te eenvoudig maakt, verandert er door deze optimalisatie visueel niets. De gemiddelde framesnelheid van uw definitieve toepassing kan echter aanzienlijk beter worden als u complexe paden vereenvoudigt.

Hoofdstuk 6: Netwerkinteractie optimaliseren

Verbeteringen voor netwerkinteractie

In Flash Player 10.1 en AIR 2.5 wordt een set nieuwe functies geïntroduceerd voor netwerkoptimalisatie op alle platforms, inclusief cyclische buffering en slim zoeken.

Cyclische buffering

Wanneer u media-inhoud op mobiele apparaten laadt, kunnen er problemen optreden die u bijna nooit zou verwachten op een bureaubladcomputer. U kunt bijvoorbeeld sneller zonder schijfruimte of geheugen raken. Als u video laadt, downloaden de desktopversie van Flash Player 10.1 en AIR 2.5 het volledige FLV-bestand (of MP 4-bestand) naar de harde schijf en slaan dit bestand in het cachegeheugen op. Vervolgens speelt de runtime de video af vanuit het cachebestand. Het komt niet vaak voor dat er onvoldoende ruimte op de harde schijf beschikbaar is. Als dit wel het geval is, stopt de desktop het afspelen van de video.

Op een mobiel apparaat is de schijfruimte eerder op. Als het apparaat onvoldoende schijfruimte heeft, stopt de runtime het afspelen niet, zoals bij de desktop. In plaats daarvan gebruikt de runtime het cachebestand opnieuw door vanaf het begin van het bestand opnieuw naar dit bestand te schrijven. De gebruiker kan doorgaan met het bekijken van de video. De gebruiker kan niet zoeken in het gebied van de video waarnaar opnieuw is geschreven behalve het begin van het bestand. Cyclische buffering wordt niet standaard gestart. Het kan worden gestart tijdens het afspelen en ook aan het begin van het afspelen als de film groter is dan de beschikbare schijfruimte of RAM-geheugen. De runtime heeft ten minste 4 MB RAM of 20 MB schijfruimte nodig om cyclische buffering te kunnen gebruiken.

Opmerking: *Als het apparaat voldoende schijfruimte heeft, gedraagt de mobiele versie van de runtime zich net zo als de desktopversie. Houd er rekening mee dat een buffer in het RAM-geheugen wordt gebruikt als fallback als het apparaat geen schijf heeft of de schijf vol is. Bij de compilatie kan er geen limiet worden ingesteld voor de grootte van het cachebestand en de RAM-buffer. De structuur van sommige MP4-bestanden vereist dat het volledige bestand wordt gedownload voordat het bestand kan worden afgespeeld. De runtime detecteert deze bestanden en verhindert het downloaden als er onvoldoende schijfruimte is. Het MP4-bestand kan nu niet worden afgespeeld. U kunt het best het downloaden van deze bestanden niet aanvragen.*

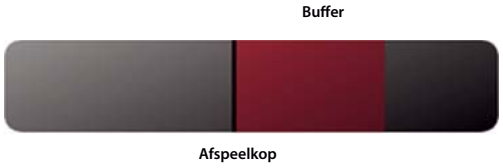
Als ontwikkelaar moet u er rekening mee houden dat zoeken alleen werkt binnen de grenzen van de stream in cache. `NetStream.seek()` mislukt soms als de verschuiving zich buiten het bereik bevindt. In dit geval wordt een `NetStream.Seek.InvalidTime`-gebeurtenis verzonden.

Slim zoeken

Opmerking: *Voor slim zoeken is Adobe® Flash® Media Server 3.5.3 vereist.*

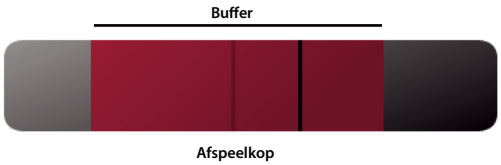
In Flash Player 10.1 en AIR 2.5 wordt een nieuw gedrag met de naam slim zoeken geïntroduceerd om de gebruikerservaring bij het afspelen van streaming video te verbeteren. Als de gebruiker een bestemming zoekt binnen de grenzen van de buffer, gebruikt de runtime de buffer opnieuw om onmiddellijk zoeken mogelijk te maken. In eerdere versies van de runtime werd de buffer niet opnieuw gebruikt. Als een gebruiker bijvoorbeeld een video afspeelt van een streaming server en de buffertijd is ingesteld op 20 seconden (`NetStream.bufferTime`) en de gebruiker heeft geprobeerd 10 seconden vooruit te zoeken, gooit de runtime alle buffergegevens weg in plaats van de reeds geladen 10 seconden opnieuw te gebruiken. Dit gedrag dwingt de runtime om vaker nieuwe gegevens van de server op te vragen en leidt bij trage verbindingen tot een matige afspeelkwaliteit.

De onderstaande afbeelding illustreert hoe de buffer in de vorige versie van de runtime werkte. De eigenschap `bufferTime` geeft het aantal seconden aan dat vooruit en vooraf wordt geladen, zodat de buffer kan worden gebruikt zonder dat de video wordt gestopt als de verbinding wegvalt.

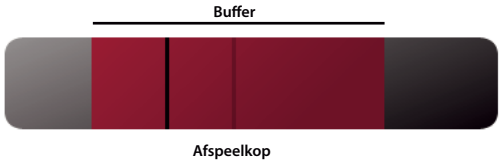


Buffergedrag vóór de functie slim zoeken

Met de functie slim zoeken gebruikt de runtime nu de buffer om onmiddellijk achteruit of vooruit te zoeken wanneer de gebruiker de video versleept. In de volgende afbeelding wordt dit nieuwe gedrag geïllustreerd:



Vooruit zoeken met de functie slim zoeken



Achteruit zoeken met de functie slim zoeken

Slim zoeken gebruikt de buffer opnieuw wanneer de gebruiker vooruit of achteruit zoekt zodat het afspelen sneller en vlotter verloopt. Een van de voordelen van dit nieuwe gedrag is het besparen van bandbreedte voor video-uitgevers. Als het zoeken echter buiten de grenzen van de buffer wordt uitgevoerd, treedt het standaardgedrag op en vraagt de runtime nieuwe gegevens aan bij de server.

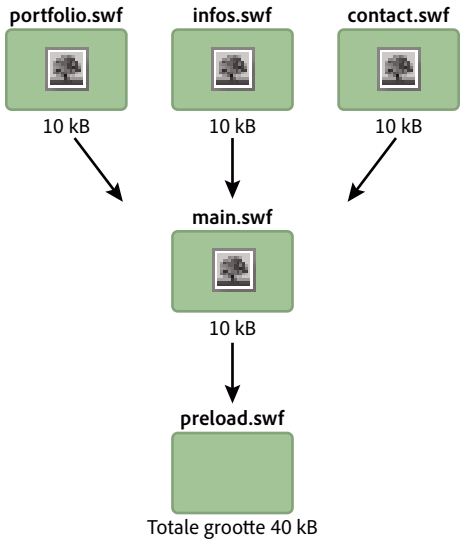
Opmerking: *Dit gedrag is niet van toepassing op het progressief downloaden van video.*

Om slim zoeken in te schakelen stelt u `NetStream.inBufferSeek` in op `true`.

Externe inhoud

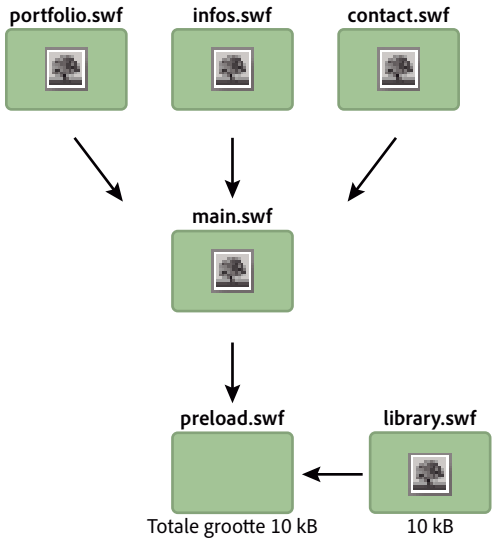
Verdeel uw toepassing in meerdere SWF-bestanden.

Mobiele apparaten hebben mogelijk beperkte toegang tot het netwerk. Als u uw inhoud snel wilt laden, verdeelt u uw toepassing in meerdere SWF-bestanden. Probeer de codelogica en elementen opnieuw te gebruiken over de hele toepassing. Laat ons bijvoorbeeld even het volgende diagram bekijken waarbij een toepassing is verdeeld in meerdere SWF-bestanden.



Toepassing die is opgedeeld in meerdere SWF-bestanden

In dit voorbeeld bevat elk SWF-bestand zijn eigen kopie van dezelfde bitmap. Deze duplicatie kan worden vermeden door een bij uitvoering gedeelde bibliotheek te gebruiken, zoals wordt weergegeven in het volgende diagram:



Een gedeelde runtime-bibliotheek gebruiken

Als u deze techniek gebruikt, wordt een bij uitvoering gedeelde bibliotheek geladen om de bitmap beschikbaar te stellen voor andere SWF-bestanden. De ApplicationDomain-klasse slaat alle klassedefinities op die zijn geladen en stelt deze beschikbaar tijdens uitvoering via de `getDefinition()`-methode.

Een bij uitvoering gedeelde bibliotheek kan eveneens alle codelogica bevatten. De volledige toepassing kan worden bijgewerkt tijdens de uitvoering zonder dat deze opnieuw moet worden gecompileerd. De volgende code laadt een bij uitvoering gedeelde bibliotheek en extraheert de definitie die tijdens de uitvoering in het SWF-bestand zit. Deze techniek kan worden gebruikt met lettertypen, bitmaps, geluiden of elke ActionScript-klasse:

```
// 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.");
}
```

Het ophalen van de definitie kan worden vereenvoudigd door de klassedefinities te laden in het toepassingsdomein van het geladen SWF-bestand:

```
// 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.");
}
```

De klassen die beschikbaar zijn in het geladen SWF-bestand kunnen vervolgens worden gebruikt door de `getDefinition()`-methode aan te roepen op het huidige toepassingsdomein. U kunt ook toegang krijgen tot de klassen door de `getDefinitionByName()`-methode aan te roepen. Met deze techniek wordt bandbreedte bespaard door lettertypen en grote elementen slechts eenmaal te laden. Elementen worden nooit geëxporteerd naar andere SWF-bestanden. De enige beperking is dat de toepassing moet worden getest en door het loader.swf-bestand moet worden uitgevoerd. Dit bestand laadt eerst de elementen en vervolgens de verschillende SWF-bestanden waaruit de toepassing is samengesteld.

Invoer-/uitvoerfouten

Gebeurtenishandlers en foutberichten voor invoer-/uitvoerfouten bieden.

Op een mobiel apparaat is het netwerk mogelijk minder betrouwbaar dan op een bureaubladcomputer met een snelle internetverbinding. Toegang tot externe inhoud op mobiele apparaten heeft twee beperkingen: beschikbaarheid en snelheid. Daarom moet u ervoor zorgen dat de elementen lichtgewichtelementen zijn en moet u handlers toevoegen voor elke IO_ERROR-gebeurtenis om feedback naar de gebruiker te sturen.

Stelt u zich bijvoorbeeld eens een gebruiker voor die via een mobiele telefoon op uw website bladert en plots de netwerkverbinding verliest tussen twee metrostations. Op het moment dat de verbinding wegviel werd een dynamisch element geladen. Op het bureaublad kunt u een lege gebeurtenislistener gebruiken om te voorkomen dat een fout bij uitvoering wordt weergegeven, omdat dit scenario bijna nooit zou voorkomen. Op een mobiel apparaat moet u echter de situatie met meer dan slechts een lege listener afhandelen.

De volgende code komt niet overeen met een invoer-/uitvoerfout. Gebruik deze niet zoals weergegeven:

```
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;
}
```

Het is beter een dergelijke fout af te handelen en een foutbericht voor de gebruiker weer te geven. De volgende code handelt dit probleem correct af:

```
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
}
```

Denk eraan dat u de gebruiker steeds een manier aanbiedt om de inhoud opnieuw te laden. Dit gedrag kan worden geïmplementeerd in de `onIOError()`-handler.

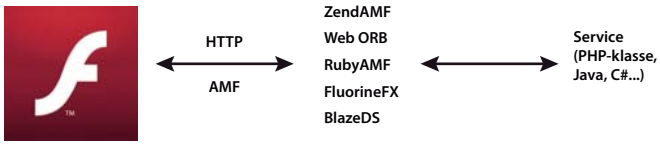
Flash Remoting

Gebruik Flash Remoting en AMF voor geoptimaliseerde gegevenscommunicatie tussen de client en de server.

U kunt XML gebruiken om externe inhoud in SWF-bestanden te laden. XML is echter onbewerkte tekst die door de runtime wordt geladen en geparseerd. XML werkt het best bij toepassingen die een beperkte hoeveelheid inhoud laden. Als u een toepassing ontwikkelt die een grote hoeveelheid inhoud ontwikkelt, kunt u de Flash Remoting-technologie en Action Message Format (AMF) gebruiken.

AMF is een binaire indeling die wordt gebruikt om gegevens te delen tussen een server en de runtime. Het gebruik van AMF reduceert de omvang van de gegevens en verhoogt de snelheid van de gegevensoverdracht. Aangezien AMF een eigen indeling voor de runtime is, voorkomt u met het verzenden van AMF-gegevens naar de runtime een geheugenintensieve serialisatie en deserialisatie aan de clientzijde. De externe gateway handelt deze taken af. Als een ActionScript-gegevenstype naar een server wordt verzonden, handelt de externe gateway de serialisatie voor u af op de server. De gateway zendt u ook het overeenkomstige gegevenstype. Dit gegevenstype is een klasse die op de server is gemaakt en waarmee een set methoden wordt weergegeven die kunnen worden aangeroepen vanuit de runtime. Flash Remoting-gateways zijn onder andere ZendAMF, FluorineFX, WebORB en BlazeDS, een officiële open source Java Flash Remoting-gateway van Adobe.

De volgende afbeelding geeft het Flash Remoting-concept weer:



Flash Remoting

Het volgende voorbeeld gebruikt de NetConnection-klasse om verbinding te maken met een Flash Remoting-gateway:

```
// 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 ?");
```

Verbinding maken met een externe gateway is redelijk eenvoudig. Het gebruik van Flash Remoting kan echter nog eenvoudiger worden gemaakt door de RemoteObject-klasse in de Adobe ® Flex® SDK te gebruiken.

Opmerking: *Externe SWF-bestanden, zoals die van het Flex-raamwerk, kunnen worden gebruikt in een Adobe® Flash® Professional-project. Het gebruik van SWC-bestanden staat u toe de RemoteObject-klasse en afhankelijkheden te gebruiken zonder dat u de rest van de Flex SDK moet gebruiken. Geavanceerde ontwikkelaars kunnen, indien nodig, zelfs rechtstreeks communiceren met een externe gateway via de onbewerkte Socket-klasse.*

Overbodige netwerkbewerkingen

Plaats elementen na het laden lokaal in de cache, zodat deze niet elke keer opnieuw vanaf het netwerk hoeven te worden geladen.

Als de toepassing elementen zoals media of gegevens laadt, plaatst u deze in de cache door ze op het lokale apparaat op te slaan. Bij elementen die niet vaak worden gewijzigd, hoeft de cache slechts af en toe te worden bijgewerkt. Dit is bijvoorbeeld het geval als de toepassing één keer per dag controleert of er een nieuwe versie van een afbeeldingsbestand is, of elke twee uur controleert of er nieuwe gegevens zijn.

U kunt elementen op verschillende manieren in de cache plaatsen, afhankelijk van het type en de kenmerken van het element:

- Media-elementen zoals afbeeldingen en video: sla de bestanden met behulp van de klassen File en FileStream in het bestandssysteem op
- Afzonderlijke gegevenswaarden of kleine hoeveelheden gegevens: sla de waarden met behulp van de klasse SharedObject als lokale gedeelde objecten op
- Grotere hoeveelheden gegevens: sla de gegevens in een lokale database op, of serialiseer de gegevens en sla deze in een bestand op.

Voor het in de cache plaatsen van gegevenswaarden is in het open-source AS3CoreLib-project de klasse ResourceCache opgenomen die het laden en het in de cache plaatsen van de waarden voor u uitvoert.

Hoofdstuk 7: Werken met media

Video

Ga naar Webinhoud optimaliseren voor levering aan mobiele apparaten op de website Adobe Developer Connection voor meer informatie over het optimaliseren van de videoprestaties op mobiele apparaten.

Raadpleeg in het bijzonder de volgende secties:

- *Video afspelen op mobiele apparatuur*
- *Codevoorbeelden*

Deze secties bevatten informatie voor het ontwikkelen van videospelers voor mobiele apparatuur, zoals:

- Richtlijnen voor het coderen van video
- Tips en trucs
- De prestaties van de videospeler profileren
- Implementatie van een referentievideospeler

StageVideo

Gebruik de klasse StageVideo om te profiteren van hardwareversnelling voor de presentatie van video.

Zie De klasse StageVideo gebruiken voor presentatie met hardwareversnelling in de ActionScript 3.0-ontwikkelaarsgids voor informatie over het gebruik van het StageVideo-object.

Audio

Vanaf Flash Player 9.0.115.0 en AIR 1.0 kan de runtime AAC-bestanden (AAC Main, AAC LC en SBR) afspelen. Een eenvoudige optimalisatiemethode is het gebruik van AAC- in plaats van MP3-bestanden. De AAC-indeling biedt betere kwaliteit en kleinere bestanden dan de MP3-indeling bij een vergelijkbare bitsnelheid. Verlaag de bestandsgrootte om bandbreedte te besparen. Dit is een belangrijke factor op mobiele apparatuur zonder supersnelle internetverbinding.

Hardware-audiodecodering

Net als voor videodecodering zijn voor audiodecodering hoge CPU-cycli vereist. Ook audiodecodering kan worden geoptimaliseerd door benutting van de beschikbare hardware. Flash Player 10.1 en AIR 2.5 kunnen tijdens de decodering van AAC-bestanden (LC-, HE/SBR-profielen) of MP 3-bestanden (PCM wordt niet ondersteund) audiostuurprogramma's voor de hardware detecteren en gebruiken om de prestaties te verbeteren. Het CPU-verbruik wordt drastisch gereduceerd. Dat betekent dat de batterij langer meegaat en de CPU beschikbaar is voor andere bewerkingen.

Opmerking: *Bij gebruik van de AAC-indeling wordt het AAC Main-profiel niet ondersteund op apparatuur, aangezien de meeste apparaten niet de juiste hardwareondersteuning verschaffen.*

Hardwareaudiodecodering is een transparant proces voor gebruikers en ontwikkelaars. Wanneer de runtime audiostreams gaat afspelen, wordt, net als bij video, eerst de hardware gecontroleerd. Hardware-audiocodering vindt plaats als er een hardwarestuurprogramma beschikbaar is en de audio-indeling wordt ondersteund. Ook als de inkomende AAC- of MP3 Stream-decodering via de hardware kan worden verwerkt, kan de hardware soms niet alle effecten verwerken. Soms verwerkt de hardware vanwege hardwarebeperkingen bijvoorbeeld geen audiomixes en resampling.

Hoofdstuk 8: Prestaties van SQL-databases

Ontwerp van toepassingen voor databaseprestaties

Wijzig de eigenschap `text` van een object SQLStatement na het uitvoeren niet meer. Gebruik in plaats daarvan één instantie SQLStatement voor elke SQL-instructie en gebruik instructieparameters voor de verschillende waarden.

Voordat een SQL-instructie wordt uitgevoerd, wordt deze door de runtime voorbereid (gecompileerd) om de stappen te bepalen die intern moeten worden uitgevoerd om de instructie uit te voeren. Als u `SQLStatement.execute()` aanroept voor een SQLStatement-instantie die nog niet is·uitgevoerd, wordt de instructie automatisch voorbereid voordat deze wordt uitgevoerd. De volgende keren dat u de methode `execute()` aanroept, is de instructie al voorbereid, mits de eigenschap `SQLStatement.text` niet is gewijzigd. Als gevolg daarvan wordt deze sneller uitgevoerd.

Als u het maximale voordeel wilt halen uit het hergebruik van instructies, en waarden tussen de uitvoering van instructies door gewijzigd moeten worden, gebruikt u instructieparameters om uw instructie aan te passen. (U geeft instructieparameters op via de associatieve-array-eigenschap `SQLStatement.parameters`.) Bij het wijzigen van de eigenschap `text` van de SQLStatement-instantie moet de runtime de instructie opnieuw voorbereiden. Als u de waarden van instructieparameters wijzigt, is dit niet het geval.

Als u een SQLStatement-instantie opnieuw gebruikt, moet uw toepassing een verwijzing naar de SQLStatement-instantie opslaan wanneer deze is voorbereid. Hiervoor definieert u de variabele als een variabele van het type 'klasse' en niet van het type 'functie'. Als u de SQLStatement-instantie als variabele van het type klasse wilt definiëren, kunt u dit het beste doen door de toepassing zo te structureren dat een SQL-instructie aan één klasse wordt toegewezen. U kunt ook een groep instructies die gecombineerd worden uitgevoerd toewijzen aan één klasse. (Deze techniek wordt het command design-patroon genoemd.) Wanneer u de instanties als lidvariabelen van de klasse definieert, blijven ze bestaan zolang de instantie van de klasse waaraan ze zijn toegewezen in de toepassing bestaat. Als minimale oplossing kunt u gewoon een variabele definiëren die de SQLStatement-instantie buiten een functie bevat zodat de instantie in het geheugen blijft bestaan. Definieer de SQLStatement-instantie bijvoorbeeld als een lidvariabele in een ActionScript-klasse of als een niet-functievariabele in een JavaScript-bestand. Vervolgens kunt u de parameterwaarden van de instructie instellen en de methode `execute()` van de instructie aanroepen wanneer u de query daadwerkelijk wilt uitvoeren.

Gebruik database-indexeringen om het vergelijken en sorteren van gegevens sneller uit te voeren.

Wanneer u een index voor een kolom maakt, slaat de database een kopie van de gegevens van die kolom op. De kopie wordt alfabetisch of numeriek gesorteerd opgeslagen. De sortering zorgt ervoor dat de database snel waarden kan zoeken (bijvoorbeeld met de operator voor gelijkheid) en de gevonden gegevens kan sorteren met de component `ORDER BY`.

Database-indexen worden voortdurend vernieuwd, waardoor bewerkingen waarmee gegevens worden gewijzigd (INSERT of UPDATE) voor die tabel enigszins langzamer verlopen. Het ophalen van gegevens kan echter wel veel sneller zijn. Ondanks deze netto prestatiewinst is het niet verstandig om zomaar elke kolom van elke tabel te indexeren. Hanteer in plaats daarvan een strategie voor het definiëren van indexen. Gebruik de volgende richtlijnen om uw indexeringsstrategie te plannen:

- Indexeer alleen kolommen die worden gebruikt in samengevoegde tabellen, WHERE-componenten·of ORDER BY-componenten
- Als kolommen vaak tezamen worden gebruikt, indexeert u deze tezamen in één index

- Als een kolom tekstgegevens bevat die alfabetisch worden opgehaald, geeft u als sortering COLLATE NOCASE voor de index op


U kunt SQL-instructies ook vooraf compileren op momenten dat de toepassing niet actief is.

De eerste keer wordt een SQL-instructie langzamer uitgevoerd omdat de SQL-tekst moet worden voorbereid (gecompileerd) door de database-engine. Aangezien het voorbereiden en uitvoeren van een instructie zeer veeleisend kan zijn, is het verstandig om de initiële gegevens vooraf te laden en vervolgens andere instructies op de achtergrond uit te voeren:

1 Laad de gegevens die de toepassing het eerst nodig heeft.

2 Nadat het initiële opstarten van uw toepassing is voltooid, of tijdens een andere periode van inactiviteit binnen de toepassing, voert u andere instructies uit.

Stel dat uw·toepassing bij het weergeven van het beginscherm helemaal geen verbinding maakt met de database. Wacht in dat geval totdat het scherm wordt weergegeven voordat u de databaseverbinding opent. Maak tot slot de SQLStatement-instanties en voer deze waar mogelijk uit.

Het is echter ook mogelijk dat uw toepassing bij het opstarten onmiddellijk bepaalde gegevens weergeeft, zoals het resultaat van een query. In dat geval voert u de SQLStatement-instantie voor de desbetreffende query uit. Nadat de initiële gegevens zijn geladen en weergegeven, maakt·u SQLStatement-instanties voor andere databasebewerkingen en voert u indien mogelijk later andere instructies uit die nodig zijn.

Als u SQLStatement-instanties hergebruikt, is de extra tijd die het voorbereiden van de instructie kost slechts eenmalig. Dit heeft waarschijnlijk geen grote gevolgen voor de algehele prestaties.


Groepeer meerdere bewerkingen waarin SQL-gegevens worden gewijzigd in een transactie.

Stel dat u een groot aantal SQL-instructies uitvoert waarbij gegevens worden toegevoegd of gewijzigd (`INSERT`- of `UPDATE`-instructies). U kunt een aanzienlijke prestatieverbetering verkrijgen door alle instructies binnen een expliciete transactie uit te voeren. Als u niet expliciet een transactie begint, wordt elke instructie binnen zijn eigen, automatisch gegenereerde transactie uitgevoerd. Nadat elke transactie (elke instructie) is voltooid, schrijft de runtime de resultaatgegevens naar het databasebestand op de schijf.

Anderzijds moet u rekening houden met wat er gebeurt als u expliciet een transactie genereert en de instructies in het kader van die transactie uitvoert. De runtime voert alle wijzigingen in het geheugen uit en schrijft ze vervolgens alle tegelijk naar het databasebestand wanneer de transactie wordt bevestigd. Het naar schijf schrijven van gegevens is doorgaans het meest tijdsintensieve deel van de bewerking. Dit betekent dat u de prestaties aanzienlijk kunt verbeteren door de gegevens alle tegelijk naar schijf te schrijven in plaats van één keer per SQL-instructie.


Verwerk de grote hoeveelheden resultaten van `SELECT`-query's in delen met behulp van de methode `execute()` (met de parameter `prefetch`) en de methode·`next()` van de klasse SQLStatement.

Stel dat u een SQL-instructie uitvoert waarmee een grote resultaatset·wordt opgehaald. De toepassing verwerkt elke rij gegevens in een lusbewerking. De gegevens worden bijvoorbeeld opgemaakt of er worden objecten van gemaakt. Het verwerken van zulke gegevens kan lang duren, waardoor er renderingproblemen kunnen optreden zoals een vastgelopen scherm. Zoals beschreven in "Asynchrone bewerkingen" op pagina 76 kunt u het werk splitsen. Dankzij de API van de SQL-database is het heel eenvoudig om gegevensverwerking te splitsen.

De methode `execute()` van de klasse SQLStatement heeft een optionele parameter `prefetch` (de eerste parameter). Als u een waarde opgeeft, bepaalt u het maximumaantal resultaatrijen dat door de database wordt geretourneerd wanneer de verwerking is voltooid:

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

Wanneer de eerste serie resultaten wordt geretourneerd, kunt u de methode `next()` aanroepen om de uitvoering van de instructie voort te zetten en de volgende serie resultaatrijen op te halen. Net zoals de methode `execute()` accepteert de methode `next()` een parameter `prefetch` waarmee u kunt aangeven hoeveel rijen er maximaal worden geretourneerd:

```
// 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);
        }
    }
}
```

U kunt de methode `next()` blijven aanroepen totdat alle gegevens zijn geladen. Zoals hierboven aangegeven kunt u bepalen wanneer alle gegevens zijn geladen. Controleer de eigenschap `complete` van het object SQLResult dat elke keer dat de methode `execute()` of `next()` wordt voltooid wordt gemaakt.

Opmerking: *Gebruik de parameter `prefetch` en de methode `next()` om de verwerking van resultaatgegevens te splitsen. Gebruik deze parameter en methode niet om de resultaten van een query te beperken tot een gedeelte van de resultaatset. Als u slechts een beperkt aantal rijen van de resultaatset van een instructie wilt ophalen. gebruikt u de component `LIMIT` van de instructie `SELECT`. Ook als er sprake is van een grote resultaatset kunt u met de parameter `prefetch` en de methode `next()` de verwerking van de resultaten splitsen.*

Gebruik meerdere asynchrone objecten SQLConnection in combinatie met één database om meerdere instructies tegelijk uit te voeren.

Wanneer een object SQLConnection door middel van de methode `openAsync()` met een database wordt verbonden, wordt het object·op de achtergrond uitgevoerd en niet in de belangrijkste uitvoeringsthread van de runtime. Daarnaast voert elke SQLConnection een eigen achtergrondthread uit. Als u meerdere objecten SQLConnection gebruikt, kunt u effectief meerdere SQL-instructies tegelijk uitvoeren.

Deze benadering heeft mogelijk ook een paar negatieve effecten. Ten eerste is er voor elk extra object SQLStatement extra geheugen nodig. Daarnaast vormt het gelijktijdig uitvoeren van bewerkingen een zwaardere belasting voor de processor, vooral op computers met slechts één CPU of CPU-core. Om deze redenen is deze benadering niet geschikt voor gebruik op mobiele apparaten.

Bovendien kan het potentiële voordeel van het hergebruik van objecten SQLStatements verloren gaan wanneer een object SQLStatement aan één object SQLConnection wordt gekoppeld. Hierdoor kan het object SQLStatement niet worden hergebruikt als het bijbehorende object SQLConnection al wordt gebruikt.

Wanneer er meerdere objecten SQLConnection met één database zijn verbonden, voert elk object zijn instructies in een eigen transactie uit. Zorg ervoor dat u deze afzonderlijke transacties verwerkt in code waarmee gegevens worden gewijzigd, zoals toevoegen, wijzigen of verwijderen.

Paul Robertson heeft een opensource codebibliotheek gemaakt waarmee u de voordelen van het gebruik van meerdere objecten SQLConnection kunt benutten en tegelijkertijd de mogelijke negatieve effecten tot een minimum kunt beperken. De bibliotheek maakt gebruik van·een groep objecten SQLConnection en beheert de bijbehorende objecten SQLStatement. Op deze manier wordt ervoor gezorgd dat de objecten SQLStatement worden hergebruikt en dat er meerdere objecten SQLConnection beschikbaar zijn voor het gelijktijdig uitvoeren van meerdere instructies. Ga naar http://probertson.com/projects/air-sqlite/ voor meer informatie en om de bibliotheek te downloaden.

Databasebestanden optimaliseren

Vermijd wijzigingen in het databaseschema.

Wijzig indien mogelijk het schema (de tabelstructuur) van een database niet meer nadat u gegevens hebt toegevoegd aan de tabellen van de database. In een databasebestand bevinden de tabeldefinities zich doorgaans aan het begin van het bestand. Wanneer u verbinding maakt met een database, worden deze definities door de runtime geladen. Wanneer u gegevens toevoegt aan databasetabellen, worden deze gegevens na de tabeldefinitiegegevens geplaatst. Als u echter wijzigingen in het schema aanbrengt, worden de nieuwe tabeldefinitiegegevens gecombineerd met de tabelgegevens in het databasebestand. Als u bijvoorbeeld een kolom aan een tabel toevoegt of een nieuwe tabel toevoegt, kan dit ertoe leiden dat verschillende gegevenstypen worden gecombineerd. Als de tabeldefinitiegegevens zich niet aan het begin van het databasebestand bevinden, duurt het langer om een verbinding met de database te openen. Het openen van een verbinding duurt langer omdat de runtime meer tijd nodig heeft om de tabeldefinitiegegevens uit de verschillende delen van het bestand te lezen.

Gebruik de methode `SQLConnection.compact()` om een database te optimaliseren nadat het schema is gewijzigd.

Roep voor het uitvoeren van schemawijzigingen de methode `SQLConnection.compact()` aan·nadat de wijzigingen zijn voltooid. Hierdoor wordt het databasebestand opnieuw ingedeeld zodat alle tabeldefinitiegegevens zich aan het begin van het bestand bevinden. Het uitvoeren van de bewerking `compact()` kan echter lang duren, met name bij grote databasebestanden.

Niet-noodzakelijke runtime-verwerking van database

Gebruik een volledig gekwalificeerde tabelnaam (inclusief databasenaam) in de SQL-instructie.

Geef in instructies altijd expliciet de naam van de database op naast de tabelnaam. (Gebruik 'main' voor de hoofddatabase.) De volgende code bijvoorbeeld bevat een expliciete databasenaam `main`:

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

Door de databasenaam expliciet op te geven zorgt u ervoor dat de runtime niet elke verbonden database hoeft te doorzoeken om de vereiste tabel te vinden. Bovendien kan de runtime niet de verkeerde database kiezen. Neem altijd deze regel in acht, zelfs als een SQLConnection met maar één database is verbonden. De SQLConnection is namelijk op de achtergrond ook verbonden met een tijdelijke database die toegankelijk is via SQL-instructies.

Gebruik expliciete kolomnamen in de SQL-instructies `INSERT` en `SELECT`.

In het volgende voorbeeld wordt het gebruik van expliciete kolomnamen getoond:

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

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

Vergelijk de voorgaande voorbeelden met deze: Gebruik deze codestijl niet:

```
-- 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
```

Zonder expliciete kolomnamen is er extra tijd nodig om de kolomnamen te achterhalen. Als u in een instructie `SELECT` een jokerteken gebruikt in plaats van expliciete kolommen, haalt de runtime extra gegevens op. Voor deze extra gegevens is extra verwerkingstijd nodig en worden overbodige objectinstanties gemaakt.

Zorg ervoor dat een tabel niet meerdere keren wordt samengevoegd in een instructie, tenzij de tabel met zichzelf wordt vergeleken.

Naarmate SQL-instructies groter worden, kan het gebeuren dat een databasetabel per ongeluk meerdere keren met de query wordt samengevoegd. Vaak hoeft de tabel maar één keer te worden gebruikt om hetzelfde resultaat te bereiken. Een tabel meerdere keren samenvoegen gebeurt vooral als u een of meerdere weergaven in een query gebruikt. Het kan bijvoorbeeld voorkomen dat u een tabel met de query samenvoegt, en tegelijkertijd een weergave samenvoegt die gegevens uit die tabel bevat. Deze twee bewerkingen leiden tot meer dan één samenvoeging.

Efficiënte SQL-syntaxis

Gebruik `JOIN` (in de `FROM`-component) om een tabel in een query op te nemen in plaats van een subquery in de `WHERE`-component. Deze tip is ook van toepassing als u de gegevens in een tabel alleen nodig hebt om te filteren, en niet als resultaatset.

Het samenvoegen van meerdere tabellen in de `FROM`-component levert betere prestaties dan het gebruik van een subquery in een `WHERE`-component.

Gebruik liever geen SQL-instructies die geen gebruik kunnen maken van indexen. Dit zijn instructies die gebruik maken van geaggregeerde functies in een subquery, een instructie `UNION` in een subquery of een component `ORDER BY` met een instructie `UNION`.

Met een index kan de verwerking van een `SELECT`-query aanzienlijk worden versneld. Maar bepaalde SQL-syntaxis zorgt ervoor dat de database geen indexen kan gebruiken en gedwongen wordt om de werkelijke gegevens te gebruiken voor zoek- of sorteerbewerkingen.

Gebruik de operator `LIKE` liever niet, vooral niet met een jokerteken aan het begin, bijvoorbeeld `LIKE('%XXXX%')`.

De bewerking `LIKE` ondersteunt het gebruik van zoekopdrachten met jokertekens en is daarom langzamer dan vergelijkingen met exacte overeenkomst. Met name als u een jokerteken aan het begin van een zoektekenreeks plaatst, kan de database geen indexen gebruiken tijdens de zoekopdracht. In dat geval moet de volledige tekst van elke rij van de tabel worden doorzocht.

Gebruik de operator `IN` liever niet. Als de mogelijke waarden van tevoren bekend zijn, kan de `IN`-bewerking worden geschreven met behulp van `AND` of `OR` voor een snellere verwerking.

De tweede van de volgende twee instructies wordt sneller uitgevoerd. Deze instructie is sneller omdat deze gebruik maakt van eenvoudige expressies van gelijkheid in combinatie met `OR` in plaats van de instructies `IN()` of `NOT IN()`:

```
-- 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
```

Gebruik andere alternatieve SQL-instructies om de prestaties te verbeteren.

Zoals aangegeven in de voorgaande voorbeelden, kan de manier waarop een SQL-instructies is geschreven van invloed zijn op de databaseprestaties. Er zijn vaak meerdere manieren om de SQL-instructie `SELECT` te schrijven om een specifieke resultaatset op te halen. In sommige gevallen is de ene·benadering aanzienlijk sneller dan de andere. Naast de voorgaande suggesties kunt u meer te weten komen over verschillende SQL-instructies en de bijbehorende prestaties via speciale bronnen·over de SQL-taal.

Prestaties van SQL-instructies

Vergelijk verschillende SQL-instructies met elkaar om te bepalen welke sneller is.

De beste manier om de prestaties van meerdere versies van een SQL-instructie te vergelijken is deze rechtstreeks met uw database en gegevens te testen.

De volgende hulpprogramma's·geven aan hoe lang het uitvoeren van SQL-instructies duurt. Gebruik deze om de snelheid van verschillende versies van instructies met elkaar te vergelijken:

- Run! (Hulpprogramma voor het schrijven en testen van AIR SQL-query's van Paul Robertson)
- Lita (SQLite-beheerprogramma van David Deraedt)

Hoofdstuk 9: Benchmarking en implementatie

Benchmarking

Er zijn een aantal tools beschikbaar voor benchmarkingtoepassingen. U kunt de door Flash-communityleden ontworpen Stats- en PerformanceTest-klasse gebruiken. U kunt ook de analyse gebruiken in Adobe® Flash® Builder™ en de FlexPMD-tool.

De Stats-klasse

Als u uw code tijdens de uitvoering wilt profileren met behulp van de releaseversie van de runtime en zonder een extern hulpmiddel, kunt u de Stats-klasse gebruiken. Deze is ontwikkeld door mr. doob van de Flash-community. U kunt de Stats-klasse vanaf het volgende adres downloaden: https://github.com/mrdoob/Hi-ReS-Stats.

Met de Stats-klasse kunt u het volgende bijhouden:

- Het aantal per seconde gerenderde frames (hoe hoger het aantal, hoe beter).
- Het aantal milliseconden waarin een frame wordt gerenderd (hoe lager het aantal, hoe beter).
- De hoeveelheid geheugen die de code in beslag neemt. Als deze hoeveelheid met elk frame toeneemt, kan het zijn dat uw toepassing een geheugenlek heeft. Het is belangrijk dit eventuele lek te onderzoeken.
- De maximale hoeveelheid geheugen die de toepassing gebruikt.

Als u de Stats-klasse hebt gedownload, kunt u deze gebruiken met de volgende compacte code:

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

Door gebruik van voorwaardelijke compilatie in Adobe® Flash® Professional of Flash Builder, kunt u het Stats-object inschakelen:

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

U kunt de complicatie van het Stats-object in- of uitschakelen door de waarde van de `DEBUG`-constante te veranderen. U kunt op deze manier alle codelogica vervangen die u niet in uw toepassing wilt compileren.

De PerformanceTest-klasse

Om uitvoering van ActionScript-code te profileren, heeft Grant Skinner een tool ontworpen voor integratie in een workflow voor het testen van eenheden. U kunt een aangepaste klasse doorgeven aan de PerformanceTest-klasse die een aantal tests uitvoert op uw code. Met de PerformanceTest-klasse kunt u gemakkelijk benchmarking uitvoeren op verschillende werkwijzen. U kunt de klasse PerformanceTest downloaden vanaf het volgende adres: http://www.gskinner.com/blog/archives/2009/04/as3_performance.html.

De Flash Builder-analyse

Bij Flash Builder hoort een analyse waarmee u bijzonder gedetailleerde benchmarking kunt uitvoeren op uw code.

Opmerking: *Gebruik de foutopsporingsversie van Flash Player om de analyse te openen, anders verschijnt er een foutbericht.*

U kunt de analyse ook gebruiken met inhoud die in Flash Professional is gemaakt. U doet dit door het gecompileerde SWF-bestand uit een ActionScript- of Flex-project te laden in Flash Builder en de analyse er op uit te voeren. Zie "Profiling Flex applications" in Using Flash Builder 4 voor meer informatie over de analyse.

FlexPMD

Adobe Technical Services heeft de tool FlexPMD uitgebracht waarmee u de kwaliteit van ActionScript 3.0-code kunt controleren. FlexPMD is een ActionScript-tool en valt te vergelijken met JavaPMD. FlexPMD verbetert de kwaliteit van de code door een ActionScript 3.0- of Flex-brondirectory te controleren. De tool detecteert slechte codepraktijken, zoals ongebruikte code, onnodig complexe code, te lange code of onjuist gebruik van de levensduur van de Flex-component.

FlexPMD is een Adobe-open-source-project dat beschikbaar is op het volgende adres: http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD. Er is ook een Eclipse-insteekmodule beschikbaar op het volgende adres: http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD+Eclipse+plugin.

Met FlexPMD kunt u code gemakkelijker controleren, zodat u zeker weet dat deze zuiver en geoptimaliseerd is. De ware kracht van FlexPMD schuilt in de uitbreidbaarheid. Ontwikkelaars kunnen namelijk hun eigen regels opstellen voor het controleren van alle mogelijke code. U kunt bijvoorbeeld een set regels maken die intensief gebruik van filters opsporen of welke andere slechte codepraktijk dan ook die u wilt uitroeien.

Implementatie

Wanneer u de uiteindelijke versie van uw toepassing exporteert naar Flash Builder, dient u ervoor te zorgen dat u de releaseversie exporteert. Wanneer u de releaseversie exporteert, wordt de foutopsporingsinformatie uit het SWF-bestand verwijderd. Verwijdering van de foutopsporingsinformatie reduceert de grootte van het SWF-bestand en zorgt dat de toepassing sneller wordt uitgevoerd.

U exporteert de releaseversie van uw project met gebruik van het deelvenster Project in Flash Builder en de optie voor het exporteren van de releasebuild.

Opmerking: *Wanneer u uw project compileert in Flash Professional, kunt u niet kiezen tussen de release- en foutopsporingsversie. Het gecompileerde SWF-bestand is standaard een releaseversie.*