Optimera prestanda för ADOBE® FLASH® -plattformar

Juridiska meddelanden

Mer information om juridiska meddelanden finns i http://help.adobe.com/sv_SE/legalnotices/index.html.

Innehåll

Kapitel 1: Inledning

Adobe® AIR®- och Adobe® Flash® Player-program fungerar på många plattformar, bland annat stationära datorer, mobilenheter, tablet-datorer och tv-enheter. Med olika kodexempel och fallstudier som presenteras i detta dokument får utvecklare tips på hur de bäst kan distribuera dessa program. Ämnena omfattar:

- Spara minneskapacitet
- Minimera processoranvändning
- Förbättra ActionScript 3.0-prestanda
- Öka återgivningshastigheten
- Optimera nätverksinteraktion
- Arbeta med ljud och video
- Optimera SQL-databasprestanda
- Testning och distribution av program

De flesta av optimeringarna gäller program på alla enheter, både i AIR-miljön och i Flash Player-miljön. Tillägg och undantag för särskilda enheter behandlas också.

En del av optimeringarna har fokus på funktioner som introducerades i Flash Player 10.1 och AIR 2.5, men många av dem gäller även tidigare versioner av AIR och Flash Player.

Grundläggande om exekvering av körtidskod

En viktig del av att förstå hur du kan förbättra programmets prestanda är att känna till hur koden körs på Flash-plattformen. Körtidskoden fungerar i en slinga där vissa åtgärder inträffar i varje ”bildruta”. En bildruta är i det här fallet bara ett block av tid som fastställs av bildrutefrekvensen som angetts för programmet. Mängden tid som tilldelas varje bildruta motsvarar bildrutefrekvensen. Om du t.ex. anger en bildrutefrekvens på 30 bildrutor per sekund kommer körtidsmodulen att försöka göra så att varje bildruta varar i en trettiondels sekund.

Du anger den inledande bildrutefrekvensen för programmet under redigeringen. Du kan ställa in bildrutefrekvensen med inställningarna i Adobe® Flash® Builder™ eller Flash Professional. Du kan även ange den inledande bildrutefrekvensen i koden. Du kan ställa in bildrutefrekvensen i ett program med enbart ActionScript med metadatataggen `[SWF(frameRate="24")]` i rotdokumentklassen. I MXML ställer du in attributet `frameRate` i taggen Application eller WindowedApplication.

Varje bildruteslinga består av två faser som delas upp i tre delar: händelser, händelsen `enterFrame` och återgivning.

Den första fasen består av två delar (händelser och händelsen `enterFrame`) som båda kan göra att din kod anropas. Körtidshändelser tas emot och skickas under den första delen av fas 1. Händelserna kan representera slutförande eller förlopp för asynkrona åtgärder, t.ex. svar på inläsning av data över ett nätverk. De kan även inkludera händelser på grund av användarinmatningar. Körtidsmodulen verkställer din kod i avlyssnarna som du har registrerat när händelser skickas. Om inga händelser inträffar, väntar körtidsmodulen på att slutföra körningsfasen utan att utföra en åtgärd. Körtidsmodulen ökar inte bildrutefrekvensen på grund av inaktivitet. Om händelser inträffar under andra delar av körningscykeln, köar körtidsmodulen händelserna och skickar dem i nästa bildruta.

Den andra delen av fas 1 är händelsen `enterFrame`. Händelsen skiljer sig från de övriga eftersom den alltid skickas en gång per bildruta.

Bildruteslingans återgivningsfas startar när alla händelser har skickats. Då beräknar körtidsmodulen tillståndet för alla synliga element på skärmen och ritar dem på skärmen. Sedan upprepas processen, precis som en löpare som springer runt en bana.

Obs! *För händelser som inkluderar en* `updateAfterEvent`*-egenskap kan du tvinga att återgivningen sker omedelbart i stället för att återgivningsfasen inväntas. Du bör dock undvika att använda* `updateAfterEvent` *eftersom det ofta leder till prestandaproblem.*

Det är enklast att tänka sig att de två faserna i bildruteslingan tar lika lång tid. Under halva bildruteslingan körs händelsehanterare och programkod och under den andra halvan sker återgivningen. I verkligheten är det dock ganska annorlunda. Ibland tar programkoden mer än hälften av den tillgängliga tiden i bildrutan och dess tidstilldelning utökas vilket minskar den tillgängliga tiden för återgivning. I andra fall kan återgivningen ta mer än halva bildrutetiden, speciellt när komplicerat visuellt innehåll som filter och blandningslägen används. Eftersom den verkliga tiden för faserna är flexibel, liknas bildruteslingan ofta för en ”elastisk löparbana”.

Om de kombinerade åtgärderna i bildruteslingan (kodkörning och återgivning) tar för lång tid kan körtidsmodulen inte bevara bildrutefrekvensen. Bildrutan expanderar och tar längre än den tilldelade tiden vilket innebär en fördröjning innan nästa bildruta aktiveras. Om t.ex. en bildruteslinga tar längre än en trettiondels sekund kan körtidsmodulen inte uppdatera skärmen med 30 bildrutor per sekund. Upplevelsen försämras när bildrutefrekvensen saktar ned. I bästa fall blir animering ryckig. I värsta fall låser sig programmet och fönstret töms.

Se följande resurser för mer information om verkställande av körtidskod på Flash-plattformen och återgivningsmodellen:

- Flash Player mentalmodell - den elastiska löparbanan (artikel av Ted Patrick)
- Asynkrona körningar i ActionScript (artikel av Trevor McCauley)
- Optimera Adobe AIR för kodkörning, minne och återgivning på http://www.adobe.com/go/learn_fp_air_perf_tv_se (videopresentation av Sean Christmann från MAX-konferensen)

Upplevda prestanda jämfört med verkliga prestanda

De som till sist avgör om programmet fungerar bra är användarna. Utvecklare kan mäta programmets prestanda i termer om hur lång tid det tar att utföra vissa åtgärder eller hur många objektinstanser som skapas. Sådana mätdata är dock inte relevanta för slutanvändarna. Ibland mäter användare prestandan med andra kriterier. Körs t.ex. programmet snabbt och jämnt och reagerar det snabbt på indata? Inverkar det negativt på systemets prestanda? Du kan mäta upplevda prestanda genom att ställa följande frågor till dig själv:

- Fungerar animeringarna jämnt eller ryckigt?
- Körs videoinnehåll jämnt eller ryckigt?
- Spelas videoklipp upp kontinuerligt eller stannar de och fortsätter igen?
- Flimrar eller töms fönstret under längre åtgärder?
- När jag skriver text, håller textinmatningen takten eller sker en fördröjning?
- Händer något omedelbart när jag klickar eller tar det en stund?
- Låter CPU:ns fläkt mer när programmet körs?
- Tar batteriet snabbt slut om programmet körs på en bärbar dator eller mobil enhet?
- Reagerar andra program sämre när programmet körs?

Skillnaden mellan upplevda prestanda och verkliga prestanda är viktig. Sättet att uppnå bästa möjliga upplevda prestanda inte alltid är detsamma som att uppnå absolut snabbaste prestanda. Se till att programmet aldrig kör så mycket kod att skärmen inte kan uppdateras och hämta användarindata ofta. I vissa fall måste du uppnå denna balans genom att dela upp programuppgifterna i delar så att körtidsmodulen kan uppdatera skärmen emellanåt. (Mer information finns i ”Återgivningsprestanda” på sidan 48.)

Tipsen och teknikerna som beskrivs här förbättrar både den verkliga körningen av koden och hur användarna upplever prestandan.

Målinriktade optimeringar

Vissa prestandaförbättringar leder inte till en märkbar förbättring för användarna Det är viktigt att du inriktar prestandaoptimeringarna på problemområden i programmet. Vissa prestandaoptimeringar är allmänt goda rutiner som alltid kan följas. Om andra optimeringar är bra eller inte beror på programmets krav och den förväntade målgruppen. Program fungerar alltid bättre om du inte använder animeringar, video, grafikfilter och effekter. Men en av anledningarna till att använda Flash-plattformen är just medie- och grafikfunktionerna som gör att du kan skapa innehållsrika program. Överväg om den önskade detaljrikedomen är lämplig för prestandaegenskaperna på de maskiner och enheter som ska köra ditt program.

Ett vanligt råd är att du bör undvika att optimera för tidigt. För vissa prestandaoptimeringar måste koden skrivas på ett sätt som är svårare att läsa eller mindre flexibelt. Sådan kod är svårare att underhålla när den har optimerats. För sådana optimeringar är det ofta bättre att vänta och sedan avgöra om ett visst stycke kod fungerar dåligt innan koden optimeras.

Att förbättra prestandan innebär ibland att du måste göra kompromisser. I idealiska fall bör hastigheten med vilken programmet utför åtgärder öka om du minskar mängden minne som programmet använder. En sådan förbättring är dock inte alltid möjlig. Om ett program låser sig när en åtgärd utförs måste du ofta dela upp arbetet på flera bildrutor. Eftersom arbetet delas upp är det troligt att den totala tiden för att slutföra en process ökar. Det är dock möjligt att användaren inte märker den extra tidsåtgången om programmet fortsätter att svara på indata och inte fryser.

Ett sätt att ta reda på vad som bör optimeras och om optimeringarna fungerar bra, är att utföra prestandatester. Flera tekniker och tips hur du testar prestandan finns i ”Testning och distribution” på sidan 95.

Se följande resurser för mer information hur du avgör vilka delar av ett program som är lämpliga för optimering:

- Prestandaoptimera program för AIR på http://www.adobe.com/go/learn_fp_goldman_tv_se (videopresentation av Oliver Goldman från MAX-konferensen)
- Prestandaoptimera Adobe AIR-program på http://www.adobe.com/go/learn_fp_air_perf_devnet_se (Adobe Developer Connection-artikel av Oliver Goldman som baseras på presentationen)

Kapitel 2: Spara minneskapacitet

Vid programutveckling är det alltid viktigt att tänka på att spara på minneskapaciteten. Detta gäller även för vanliga datorprogram. För mobila enheter är det emellertid alltid viktigt att tänka på minnesförbrukningen och det är värt besväret att försöka begränsa hur mycket minne som ska användas för programmet.

Visningsobjekt

Välj ett lämpligt visningsobjekt.

I ActionScript 3.0 finns en stor uppsättning visningsobjekt. Ett av de enklaste optimeringstipsen för att begränsa minnesanvändningen är att använda rätt typ av visningsobjekt. Om det är enkla former som inte är interaktiva ska du använda Shape-objekt. Om det är interaktiva objekt som inte behöver en tidslinje använder du Sprite-objekt och för animeringar på en tidslinje använder du MovieClip-objekt. Välj alltid det mest effektiva objektet för ditt program.

I följande kod visas minnesanvändning för olika visningsobjekt:

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

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

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

Med metoden `getSize()` visas hur många byte ett objekt upptar av minnet. Du märker att användning av flera MovieClip-objekt i stället för enkla Shape-objekt kan resultera i onödig minnesanvändning när MovieClip-objektets egenskaper inte är nödvändiga.

Primitiva typer

Använd metoden `getSize()` för att utvärdera koden och för att bestämma det mest effektiva objektet för uppgiften.

Alla primitiva typer med undantag för String använder 4 till 8 byte av minnet. Det går inte att optimera minnet genom att använda en speciell primitiv typ:

```
// 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
```

Ett Number, som representerar ett 64-bitars värde, tilldelas 8 byte genom AVM (ActionScript Virtual Machine), om det inte tilldelats ett värde. Alla andra primitiva typer sparas på 4 byte.

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

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

För typen String fungerar det annorlunda. Hur mycket som tilldelas beror på hur lång strängen är.

```
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
```

Använd metoden `getSize()` för att utvärdera koden och för att bestämma det mest effektiva objektet för uppgiften.

Återanvända objekt


Du ska återanvända objekt i stället för att återskapa dem.

Ytterligare ett enkelt sätt att optimera minnet är att återanvända objekt och undvika att återskapa dem när det är möjligt. I en slinga ska du exempelvis inte använda följande kod:

```
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);
}
```

När du återanvänder Rectangle-objekt i varje slingiteration används mer minne och det går långsammare eftersom ett nytt objekt skapas vid varje iteration. Gör i stället få följande sätt:

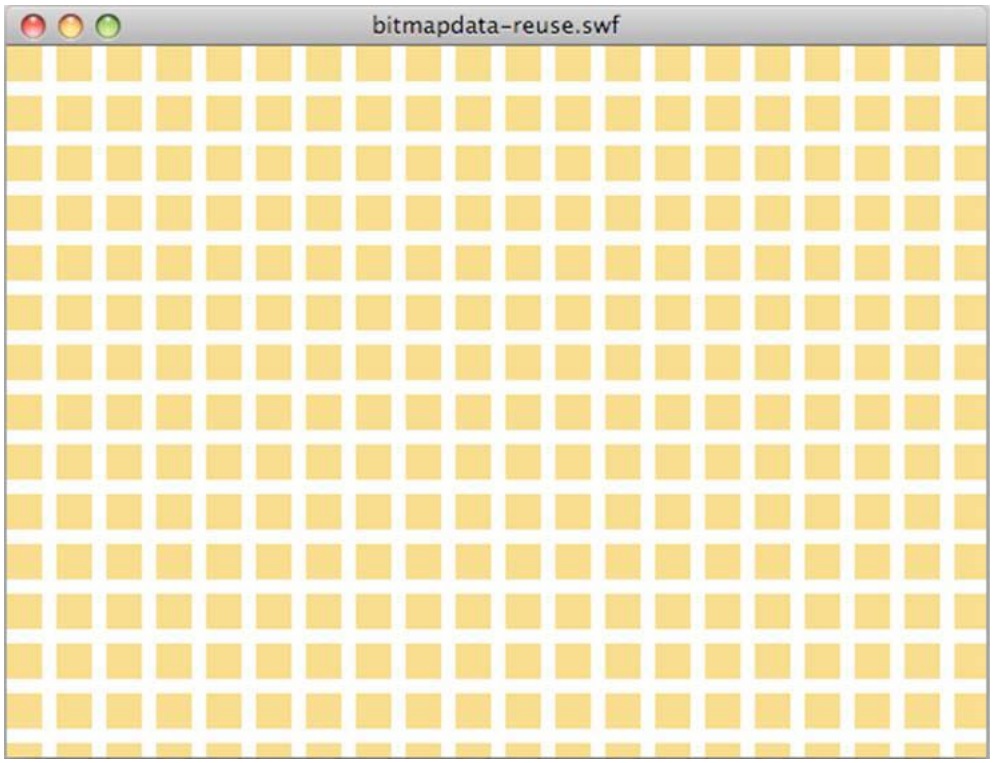
```
const MAX_NUM:int = 18;
const COLOR:uint = 0xCCCCCC;

// 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);
}
```

I det föregående exemplet användes ett objekt med en relativ liten minnespåverkan. I nästa exempel visas större minnesbesparingar när ett BitmapData-objekt återanvänds. Med följande kod skapas en indelningseffekt som kräver mycket minneskapacitet:

```
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);
}
```

Obs! *När du använder positiva värden är det mycket effektivare att göra det avrundade värdet till int än att använda metoden* `Math.floor()`.

I följande bild visas resultatet av bitmappsindelningen:



Resultat av bitmappsindelning

I en optimerad version skapas en enskild BitmapData-instans som refereras av flera bitmappsinstanser och som ger samma resultat:

```
// 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);
}
```

Detta arbetssätt sparar cirka 700 kB av minnet, vilket är en avsevärd besparing i traditionella mobilenheter. Varje bitmappsbehållare kan hanteras utan att den ursprungliga BitmapData-instansen behöver ändras med bitmappsegenskaperna:

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

I följande bild visas resultatet av bitmappsomformningarna:

Resultat av bitmappsomformningar

Fler hjälpavsnitt

”Bitmappscachning” på sidan 54

Objektpooler

Använd objektpooler när det är möjligt.

Ytterligare en viktig optimeringsteknik kallas objektpooler där objekt återanvänds över tiden. Du kan skapa ett definierat antal objekt när du initierar programmet och spara dem i en pool, som exempelvis i ett Array- eller ett Vector-objekt. När du är klar med ett objekt inaktiverar du det så att det inte förbrukar några processorresurser och tar bort alla gemensamma referenser. Du ska emellertid inte ange att referensen ska vara `null` för objektet eftersom det då kan omfattas av skräpinsamlingen. Du återplacerar helt enkelt objektet i poolen och hämtar det när du behöver ett nytt objekt.

Genom att återanvända objekt minskar du behovet av att skapa instanser av objekt, vilket kan vara kapacitetskrävande. Dessutom minskas chanserna att skräpinsamlingen körs, vilket kan medföra att programmet körs långsammare. Med följande kod exemplifieras objektpoolstekniken:

```
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;
        }
    }
}
```

I SpritePool-klassen skapas en pool med nya objekt när programmen initieras. Metoden `getSprite()` returnerar instanser av dessa objekt och metoden `disposeSprite()` frisläpper dem. Koden medger att poolen växer vid behov. Det är dessutom möjligt att skapa en pool med fast storlek där nya objekt inte tilldelas när poolen är full. Försök om möjligt att skapa nya objekt i slingor. Mer information finns i ”Frigöra minne” på sidan 11. I följande kod används klassen SpritePool för att hämta nya instanser:

```
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 );
    }
}
```

Med följande kod tas alla visningsobjekt bort från visningslistan när någon klickar med musen och återanvänder dem senare i en annan åtgärd:

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

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

Obs! *Poolvektorn refererar alltid till Sprite-objekten. Om du vill ta bort objektet fullständigt från minnet behöver du en* `dispose()`*-metod i SpritePool-klassen, vilken tar bort alla återstående referenser.*

Frigöra minne

Ta bort alla referenser till ett objekt så att skräpinsamlingen aktiveras.

Du kan inte starta skräpinsamlingen direkt med den släppta versionen av Flash Player. Om du vill vara säker på att ett objekt samlas in ska du ta bort ett alla referenser till objektet. Tänk på att den äldre `delete-` operatorn som används i ActionScript 1.0 och 2.0 uppträder annorlunda i ActionScript 3.0. Den kan endast användas för att ta bort dynamiska egenskaper i ett dynamiskt objekt.

Obs! *Du kan anropa skräpinsamlaren direkt i Adobe® AIR® och i felsökningsversionen av Flash Player.*

I exempelvis följande kod anges att en Sprite-referens ska vara `null`:

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

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

Tänk på att när ett objekt sätts till `null`, så behöver detta inte innebära att det tas bort från minnet. Skräpinsamlaren körs ibland inte om det tillgängliga minnet inte är tillräckligt litet. Skräpinsamlingen är inte förutsägbar. Minnestilldelning, inte radering av objekt, utlöser skärpinsamling. När skräpinsamlingen körs kommer objektdiagram som ännu inte samlats in att hittas. Inaktiva objekt i diagram upptäcks genom att objekt som refererar till varandra, och som inte längre används i programmet, hittas. Inaktiva objekt som upptäckts på detta sätt tas bort.

I stora program kan den här bearbetningen kräva mycket minneskapacitet och den kan påverka prestandan och medföra en märkbar försämring i programkörningen. Försök begränsa skräpinsamlingen genom att återanvända objekt så mycket som möjligt. Du kan dessutom ange att referensen ska vara null så att skräpinsamlaren ägnar mindre bearbetningstid åt att söka efter objekten. Se på skräpinsamlingen som en försäkring och hantera alltid objektens livscykler explicit när det är möjligt.

Obs! *Att ange en referens till ett visningsobjekt som null garanterar inte att objektet fryses. Objektet fortsätter att förbruka processorcykler tills det skräpsamlas. Se till att du inaktiverar objektet korrekt innan du anger dess referens som* `null`.

Skräpinsamlingen kan startas med metoden `System.gc()`, som är tillgänglig i Adobe AIR och i felsökningsversionen av Flash Player. Du kan använda profileraren som medföljer Adobe® Flash® Builder för att starta skräpinsamlingen manuellt. När du kör skräpinsamlingen lägger du märke till hur programmet svarar och om objekten tas bort på ett korrekt sätt från minnet.

Obs! *Om ett objekt användes som en händelseavlyssnare kan du låta andra objekt referera till det. Om du gör det ska du ta bort händelseavlyssnare som använder metoden* `removeEventListener()` *innan du anger att referensen ska vara* `null`.

Dessutom kan mängden minne som används för bitmappar minska omedelbart. I exempelvis klassen BitmapData finns en `dispose()`-metod. I nästa exempel skapas en BitmapData-instans på 1,8 MB. Det aktuella minnet växer till 1,8 MB och egenskapen `System.totalMemory` returnerar ett mindre värde:

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

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

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

Därefter tas BitmapData bort manuellt från minnet och minnesanvändningen kontrolleras ännu en gång:

```
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
```

Trots att metoden `dispose()` tar bort pixlarna från minnet, måste referensen ändå anges till `null` för att den ska frisläppas fulständigt. Anropa alltid metoden `dispose()` och ange att referensen ska vara `null`, när du inte längre behöver ett BitmapData-objekt, så att minnet friställs omedelbart.

Obs! *I Flash Player 10.1 och AIR 1.5.2 introduceras en ny metod i System-klassen med namnet* `disposeXML()`*. Med den här metoden kan du göra ett XML-objekt omedelbart tillgängligt för skräpinsamling, genom att skicka XML-trädet som en parameter.*

Fler hjälpavsnitt

”Frysa och tina upp objekt” på sidan 27

Använda bitmappar

Att använda vektorer i stället för bitmappar är ett bra sätt att spara minneskapacitet. Om du emellertid använder vektorer, speciellt i ett stort antal, kan behovet av processor- eller grafikprocessorresurser öka avsevärt. Att använda bitmappar är ett bra sätt att optimera återgivningen, eftersom färre bearbetningsresurser krävs för att rita pixlar på skärmen än för att återge vektorinnehåll.

Fler hjälpavsnitt

”Manuell bitmappscaching” på sidan 62

Bitmappsnedsampling

För att bättre kunna optimera minnesanvändningen kommer 32-bitars ogenomskinliga bilder att minskas till 16-bitars bilder när en 16-bitars skärm upptäcks i Flash Player. Den här nedsamplingen medför att hälften av minnesresurserna förbrukas och att bilderna kommer att återges snabbare. Den här funktionen är endast tillgänglig i Flash Player 10.1 för Windows Mobile.

Obs! *I versioner före Flash Player 10.1 sparades alla pixlar som skapades i minnet på 32 bitar (4 byte). En enkel logotyp på 300 x 300 pixlar förbrukade 350 kB (300*300*4/1024) av minnet. Tack vare det här nya beteendet kommer samma ogenomskinliga bild att förbruka endast 175 kB. Om bilden är genomskinlig kommer den inte att nedsamplas till 16 bitar och den behåller samma storlek i minnet. Den här funktionen gäller endast för inbäddade bitmappar och bilder som läsas in under körningen (PNG, GIF och JPG).*

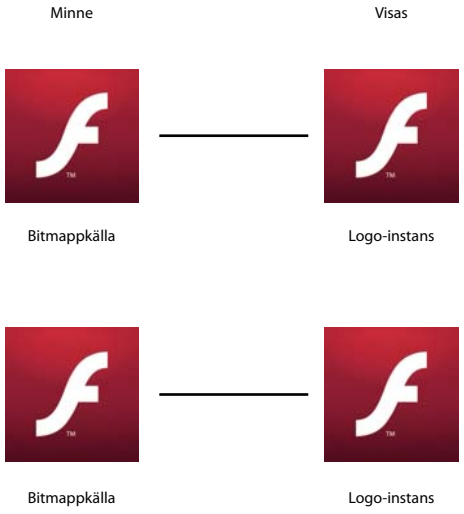
På mobilenheter kan det vara svårt att se skillnaden mellan en bild som återges med 16 bitar eller med 32 bitar. Om det är en bild som innehåller ett fåtal färger går det inte att se skillnaden. Även för mer komplexa bilder är det svårt att upptäcka skillnaderna. Det kan emellertid uppträda vissa färgförändringar när du zoomar i bilden och en 16 bitars övertoning kanske inte är lika jämn som en 32 bitars.

BitmapData med en referens

Det är viktigt att optimera användningen av klassen BitmapData genom att återanvända instanser så mycket som möjligt. Flash Player 10.1 och AIR 2.5 innehåller också en ny funktion för alla plattformar, som kallas BitmapData med en referens. När du skapar BitmapData-instanser av en inbäddad bild, används en version av bitmappen för alla BitmapData-instanser. Om en bitmapp ändras senare tilldelas den en egen unik bitmapp i minnet. Den inbäddade bilden kan komma från ett bibliotek eller en [Embed]-tagg.

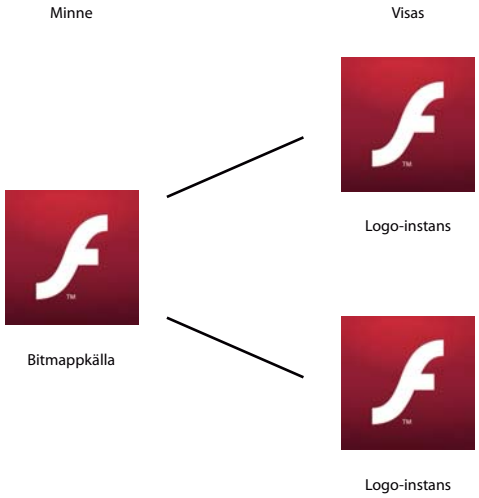
Obs! *Även befintligt innehåll kan dra nytta av den här nya funktionen, eftersom Flash Player 10.1 och AIR 2.5 automatiskt återanvänder bitmappar.*

När en inbäddad bild instansieras skapas en associerad bitmapp i minnet. Före Flash Player 10.1 och AIR 2.5 fick varje instans en separat bitmapp i minnet, som följande diagram visar:



Bitmappar i minnet före Flash Player 10.1 och AIR 2.5

När flera instanser av samma bild skapas i Flash Player 10.1 och AIR 2.5 används en enda version av bitmappen för alla BitmapData-instanser. I följande bild visas detta:

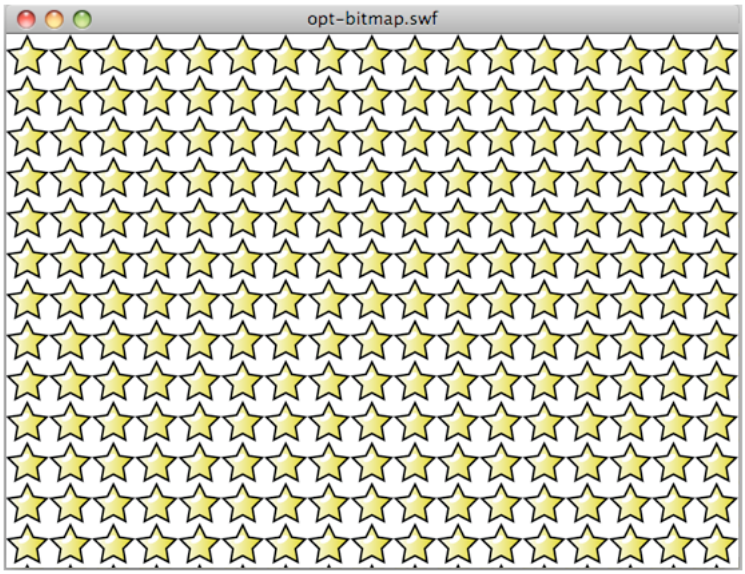


Bitmappar i minnet i Flash Player 10.1 och AIR 2.5

Det här sättet minskar drastiskt mängden minne som används i program med många bitmappar. I följande kod skapas flera instanser av en `Star`-symbol:

```
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)
    }
}
```

I följande bild visas resultatet av koden:



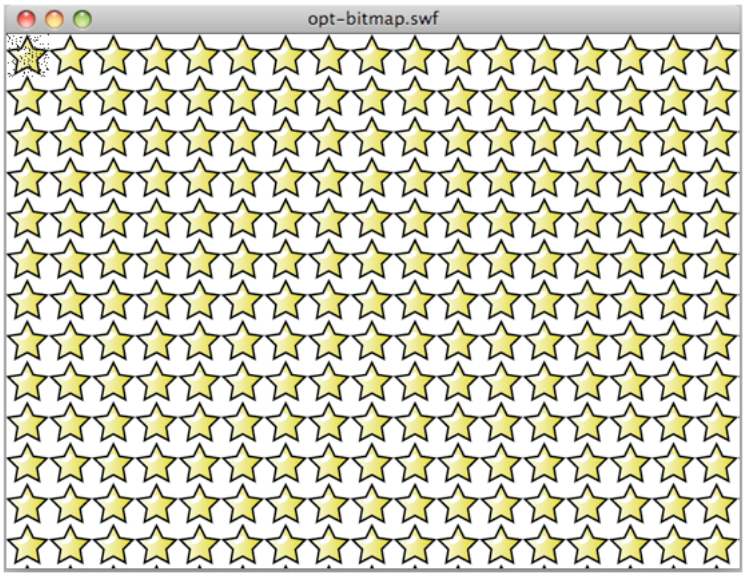
Resultat av kod som skapar flera instanser av en symbol

I Flash Player 10 använder animeringen ovan ungefär 1 008 kB minne. I Flash Player 10.1 använder animeringen, på stationära datorer och på mobilenheter, bara 4 kB.

I följande kod ändras en BitmapData-instans:

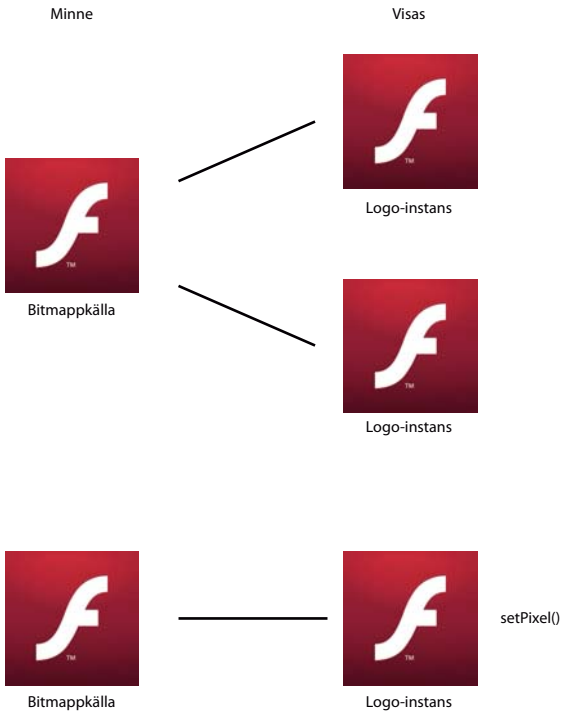
```
const MAX_NUM:int = 18;

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);
```

I bilden nedan visas resultatet av att en `Star`-instans ändras:



Resultatet av att ändra en instans

Internt tilldelar och skapar körningsmiljön automatiskt en bitmapp i minnet för att hantera pixelförändringarna. När en metod för klassen BitmapData anropas, skapas en ny instans i minnet och inga andra instanser uppdateras. I bilden nedan illustreras detta:



Resultatet i minnet av att en bitmapp ändras

Om en stjärna ändras skapas en ny kopia i minnet. Den slutliga animeringen använder ungefär 8 kB minne i Flash Player 10.1 och AIR 2.5.

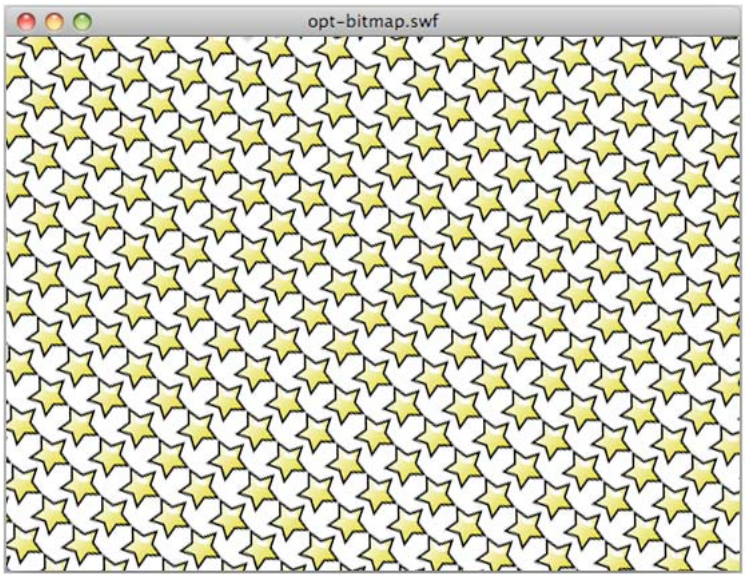
I det tidigare exemplet är varje enskild bitmapp tillgänglig för omformningar. När du vill skapa en indelningseffekt är metoden `beginBitmapFill()` den som lämpar sig bäst:

```
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);
```

Detta tillvägagångssätt ger samma resultat fast endast en BitmapData-instans skapas. Om du vill rotera stjärnorna kontinuerligt, i stället för varje Star-instans, ska du använda ett Matrix-objekt som roteras i varje bildruta. Skicka detta Matrix-objekt till metoden `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);
}
```

När du använder den här tekniken behövs ingen ActionScript-slinga för att skapa effekten. Miljön hanterar allt internt. I följande bild visas resultatet sedan stjärnorna omformats:



Resultatet av roterade stjärnor

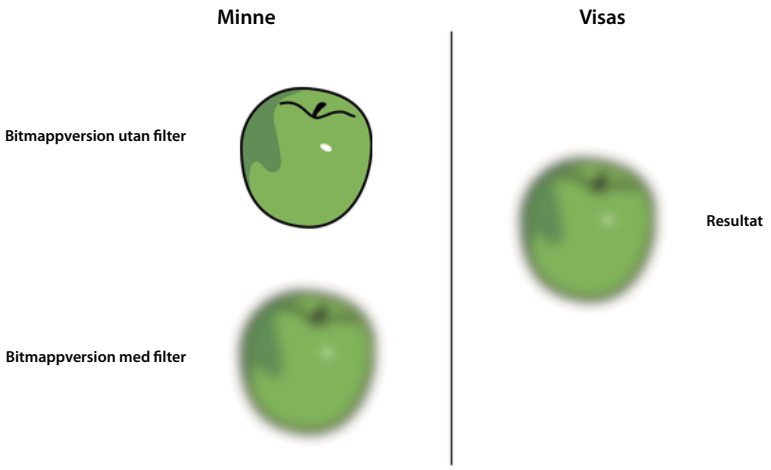
När du använder detta kommer uppdateringen av det ursprungliga BitmapData-källobjektet att automatiskt uppdatera objektet var helst det används på scenen, vilket gör att det här kan vara en användbar teknik. Det här innebär emellertid inte att varje stjärna kan skalförändras var och en för sig som i det tidigare exemplet.

Obs! *När du använder flera instanser av samma bild, beror uppritningen på om en klass är kopplad till originalbitmappen i minnet. Om ingen klass är kopplad till bitmappen, kommer bilder att ritas upp som Shape-objekt med bitmappsfyllningar.*

Filter och borttagning av dynamiska bitmappar

Undvik filter, inklusive filter bearbetade i Pixel Bender

Försök att minimera användningen av effekter som filter, bland annat filter som bearbetas i mobilenheter via Pixel Bender. När ett filter används på ett visningsobjekt skapas två bitmappar i minnet. Varje bitmapp har samma storlek som visningsobjektet. Det första skapas som en rastrerad version av visningsobjektet, vilket i sin tur används för att skapa ytterligare en bitmapp med filtret tillämpat:



Två bitmappar i minnet när filter används

När egenskapen för det ena filtret ändras, uppdateras båda bitmapparna i minnet för att skapa den resulterande bitmappen. Den här bearbetningen kräver viss processorkapacitet och två bitmappar kan kräva ganska mycket minne.

Flash Player 10.1 och AIR 2.5 har ett nytt filtreringsbeteende på alla plattformar. Om filtret inte ändras inom 30 sekunder, eller om det döljs eller inte visas på skärmen, frigörs minnet för den bitmapp som inte är filtrerad.

Detta innebär att hälften av minnesresurserna som används för ett filter går att frigöra på alla plattformar. I exempelvis ett textobjekt med ett oskärpefilter används texten enbart som en enkel dekoration och den ändras inte. Efter 30 sekunder kommer den icke filtrerade bitmappen i minnet att frigöras. Samma sak händer om texten är dold eller inte visas på skärmen i 30 sekunder. När en av filteregenskaperna ändras kommer den icke filtrerade bitmappen i minnet att återskapas. Den här funktionen kallas dynamisk bitmappsborttagning. Trots dessa förändringar ska du vara försiktig med att använda filter eftersom de fortfarande kräver mycket processor- eller grafikprocessorkapacitet när de ändras.

Ett tips är att använda bitmappar skapade med ett redigeringsverktyg, som exempelvis Adobe® Photoshop®, för att emulera filter när det är möjligt. Undvik att använda dynamiska bitmappar som skapas under körningen i ActionScript. Genom att använda externt skapade bitmappar kan belastningen på processorn och grafikprocessorn minskas, speciellt när filteregenskaperna inte förändras över tiden. Om det är möjligt bör du skapa eventuella bitmappseffekter i ett redigeringsverktyg. Du kan sedan visa bitmappen i körningsmiljön utan att behöva bearbeta den, vilket går mycket snabbare.

Direkt mipmapping

Använd mipmapping för att vid behov skalförändra stora bilder.

En annan ny funktion i Flash Player 10.1 och AIR 2.5 på alla plattformar rör mipmapping. I Flash Player 9 och AIR 1.0 introducerades en mipmappingsfunktion, som förbättrade kvaliteten och prestandan för nedskalade bitmappar.

Obs! *Mipmappingsfunktionen gäller endast för dynamiskt inlästa bilder eller inbäddade bitmappar. Mipmapping gäller inte för visningsobjekt som har filtrerats eller cache-lagrats. Mipmapping går endast att bearbeta om bredden och höjden på bitmappen har jämna värden. När höjden eller bredden har ett ojämnt värde kommer mipmappingen att avbrytas. Om exempelvis en bild är 250 x 250 kan den mipmappas ner till 125 x 125, men inte ytterligare. I detta exempel är minst ett av värdena ojämnt. Du uppnår bäst resultat för bitmappar med mått som är 2-potenser, till exempel 256 x 256, 512 x 512, 1024 x 1024 osv.*

Tänk dig att du har läst in en bild som är 1024 x 1024 och att du vill skala ned den för att skapa en miniatyr i ett galleri. Mipmappingsfunktionen återger bilden på ett korrekt sätt när den skalförändras eftersom de mellanliggande nedsamplade versionerna av bitmappen används som texturer. I tidigare versioner av körningsmiljön skapades mellanliggande nedskalade versioner av bitmappen i minnet. Om en bild på 1 024 x 1 024 lästes in och visades som 64 x 64 skulle varje halverad bitmapp skapas i äldre versioner av körningsmiljön. I detta exempel innebär det att bitmappar för 512 x 512, 256 x 256, 128 x 128 och 64 x 64 skulle skapas.

Flash Player 10.1 och AIR 2.5 har nu stöd för mipmapping direkt från den ursprungliga källan till önskad målstorlek. I föregående exempel skapades endast originalbitmappen på 4 MB (1024 x 1024) och den mipmappade bitmappen på 16 kB (64 x 64).

Mipmapping fungerar även för funktionen för borttagning av dynamiska bitmappar. Om endast bitmappen på 64 x 64 används, kommer originalbitmappen på 4 MB att frigöras från minnet. Om bitmappen måste återskapas kommer originalet att läsas in. Om dessutom andra mipmappade bitmappar med olika storlekar krävs, används mipp-kedjan av bitmappar för att skapa bitmappen. Om exempelvis en bitmapp på 1:8 måste skapas, kommer bitmapparna för 1:4, 1:2 och 1:1 att kontrolleras för att bestämma vilken av dem som ska läsas in i minnet först. Om inga andra versioner går att hitta kommer originalbitmappen på 1:1 att läsas in från resursen och användas.

Med JPEG-expandering kan mipmapping utföras inom det egna formatet. Direkt mipmapping tillåter att en stor bitmapp expanderas direkt till ett mipmapp-format utan att hela den okomprimerade bilden läses in. Det går betydligt fortare att generera mipmappen och minnet som används av stora bitmappar tilldelas inte och frigörs sedan. JPEG-bildens kvalitet är jämförbar med generella mipmapping-teknikers.

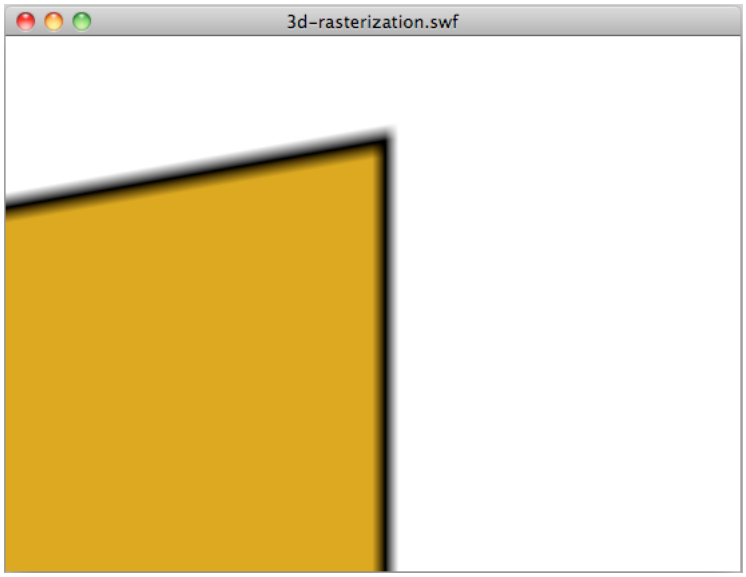
Obs! *Använd mipmapping sparsamt. Kvaliteten på nedskalade bitmappar kommer att förbättras, men dessutom påverkas bandbredden, minneskapaciteten och hastigheten. I vissa fall kan det vara bättre att använda en tidigare skalad version av bitmappen från ett externt verktyg och sedan importera den till programmet. Börja inte med stora bitmappar om du har för avsikt att skala ned dem.*

Använda 3D-effekter

Försök skapa 3D-effekter manuellt.

I Flash Player 10 och AIR 1.5 introducerades en 3D-motor, som du kan använda för att använda perspektivomformning på visningsobjekt. Du kan tillämpa dessa omformningar genom att använda egenskaperna `rotationX` och `rotationY` eller metoden `drawTriangles()` i Graphics-klassen. Med egenskapen `z` kan du dessutom få ett djup. Tänk på att varje perspektivomformat visningsobjekt rastreras som en bitmapp och därför kräver mer minneskapacitet.

I följande bild visas den kantujämning som skapas av rastreringen när perspektivomformning används:



Kantutjämning efter perspektivomformning

Kantutjämning är ett resultat av vektorinnehåll som rastreras dynamiskt som en bitmapp. Den här kantutjämningen sker när du använder 3D-effekter i datorversioner av AIR och Flash Player samt i AIR 2.0.1 och AIR 2.5 för mobiler. Kantutjämning används däremot inte i Flash Player för mobilenheter.

Du kan minska minnesanvändningen om du skapar 3D-effekter manuellt utan att förlita dig på det inbyggda API:t. Med den nya 3D-funktionen, som introducerades i Flash Player 10 och AIR 1.5, blir texturmappningen enklare tack vare metoder som `drawTriangles()`, som hanterar texturmappning internt.

Du måste bestämma om den 3D-effekt som du vill skapa ger bättre prestanda om den bearbetas i den ursprungliga API:n eller manuellt. Du ska titta på hur ActionScript körs och återgivingsprestandan samt hur mycket minne som används.

I AIR 2.0.1- och AIR 2.5-mobilprogram, i vilka du angett programegenskapen `renderMode` som `GPU`, hanterar grafikprocessorn (GPU) 3D-omformningar. Men om `renderMode` är `CPU` är det huvudprocessorn (CPU), inte grafikprocessorn, som hanterar 3D-omformningar. I Flash Player 10.1-program hanterar huvudprocessorn 3D-omformningar.

När huvudprocessorn hanterar 3D-omformningar bör du tänka på att det krävs två bitmappar i minnet om du använder en 3D-omformning på ett visningsobjekt. Den ena bitmappen är för källbitmappen, och den andra är för den perspektivomformade versionen. Detta medför att 3D-omformningar fungerar på ungefär samma sätt som filter. Du bör därför vara sparsam med att använda 3D-egenskaper när huvudprocessorn hanterar 3D-omformningar.

Textobjekt och minneshantering

Använd Adobe® Flash®-textmotorn för skrivskyddad text och TextField-objekt för indatatext.

I Flash Player 10 och AIR 1.5 introducerades en kraftfull ny textmotor, Adobe Flash Text Engine (FTE), som sparar minnesresurser. FTE är emellertid ett lågnivå-API som dessutom kräver ytterligare ett ActionScript 3.0-lager, som finns i paketet flash.text.engine.

Om du använder skrivskyddad text är det bäst att använda Flash-textmotorn (FTE), eftersom den ger låg minnesanvändning och bättre återgivning. Om det är indatatext är TextField-objekten ett bättre val eftersom mindre ActionScript-kod krävs för att skapa ett typiskt beteende, till exempel indatahantering och radbyte.

Fler hjälpavsnitt

”Återge textobjekt” på sidan 68

Händelsemodeller och återanrop

Försök använda enkla återanrop i stället för händelsemodeller.

Händelsemodellen i ActionScript 3.0 baseras på konceptet med att skicka objekt. Händelsemodellen är objektorienterad och optimerad för kodåteranvändning. Metoden `dispatchEvent()` går igenom listan med avlyssnare och anropar händelsehanterarmetoden på varje registrerat objekt. En av nackdelarna med händelsemodellen är emellertid att du troligtvis kommer att skapa många objekt över programmets livscykel.

Tänk dig att du måste skicka en händelse från tidslinjen för att ange slutet på en animeringssekvens. För att åstadkomma detta kan du skicka en händelse från en specifik bildruta på tidslinjen, vilket visas i följande kodexempel:

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

Klassen Document kan användas för att avlyssna händelsen med följande kod:

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

Även om detta är ett korrekt arbetssätt kan det gå långsammare att använda den ursprungliga händelsemodellen och dessutom förbrukas mer minne än om du valt en traditionell återanropsfunktion. Händelseobjekt måste skapas och tilldelas i minnet vilket medför att prestandan försämras. Om du exempelvis avlyssnar händelsen Event.ENTER_FRAME, skapas ett nytt händelseobjekt på varje bildruta för händelsehanteraren. Prestandan försämras speciellt för visningsobjekt eftersom inspelnings- och bubblingsfasen kan bli omfattande om visningslistan är komplex.

Kapitel 3: Minimera processoranvändning

Ytterligare ett viktigt område att fokusera på är processoranvändningen. Genom att optimera processoranvändningen förbättras prestandan, vilket i sin tur resulterar i bättre batterieffektivitet i mobila enheter.

Förbättrad processoranvändning i Flash Player 10.1

I Flash Player 10.1 introduceras två nya funktioner som leder till mindre processorutnyttjande. Funktionerna omfattar att pausa och återuppta SWF-innehåll när det hamnar utanför skärmen och att begränsa antalet Flash Player-instanser på en sida.

Pausa, strypa och återuppta

Obs! *Funktionen för att pausa, strypa och återuppta gäller inte Adobe® AIR®-program.*

För att optimera processor- och batteriutnyttjande introduceras i Flash Player 10.1 en ny funktion, som rör hanteringen av inaktiva instanser. Den här funktionen använder du för att begränsa processoranvändningen genom att pausa och återta SWF-filen när innehållet visas och tas bort från skärmen. Med den här funktionen kommer Flash Player att frigöra så mycket minneskapacitet som möjligt genom att ta bort objekt som kan återskapas när det spelade innehållet återtas. Innehåll anses vara utanför skärmen när hela innehållet är utanför.

Det kan finnas två anledningar till varför SWF-innehåll kommer utanför skärmen:

- Användaren rullar sidan vilket leder till att SWF-innehållet hamnar utanför skärmen.

  På det viset fortsätter eventuellt ljud- och videoinnehåll att spelas upp men återgivningen stoppas. Om ljud eller video inte spelas upp ska du ställa in HTML-parametern `hasPriority` på true så att uppspelningen eller körningen av ActionScript inte pausas. Du bör dock tänka på att SWF-innehåll som återges kommer att pausas när innehåll är utanför skärmen eller dolt, oberoende av värdet för HTML-parametern `hasPriority`.

- En flik öppnas i webbläsaren och SWF-innehållet flyttas till bakgrunden.

  I det här fallet, och oavsett vad värdet på HTML-taggen `hasPriority` är, saktas SWF-innehållet ned eller *stryps* till mellan 2 och 8 bildrutor i sekunden. Ljud- och videouppspelningen stoppas och ingen innehållsåtergivning bearbetas förrän SWF-innehållet blir synligt igen.

För Flash Player 11.2 och senare versioner på stationära Windows- och Mac-datorer kan du använda ThrottleEvent i programmet. Flash Player skickar en ThrottleEvent när Flash Player pausar, stryper eller återupptar uppspelningen.

ThrottleEvent är en s.k. utsändningshändelse, vilket betyder att den skickas av alla EventDispatcher-objekt som har en avlyssnare registrerad för den här händelsen. Du hittar mer information om utsändningshändelser i avsnittet om klassen DisplayObject.

Instanshantering

Obs! *Funktionen för instanshantering gäller inte Adobe® AIR®-program.*

Använd HTML-parametern `hasPriority` för att fördröja inläsningen av SWF-filer som ligger utanför skärmen.

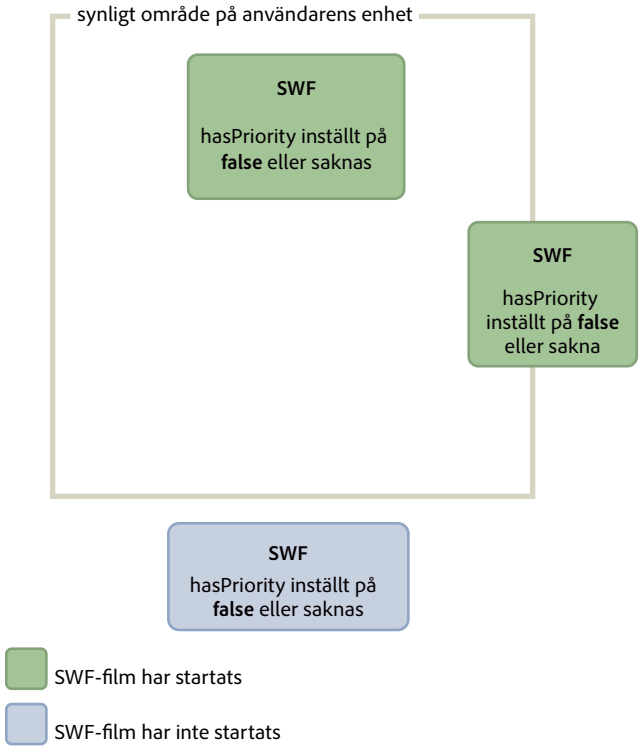
I Flash Player 10.1 introduceras en ny HTML-parameter med namnet `hasPriority`:

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

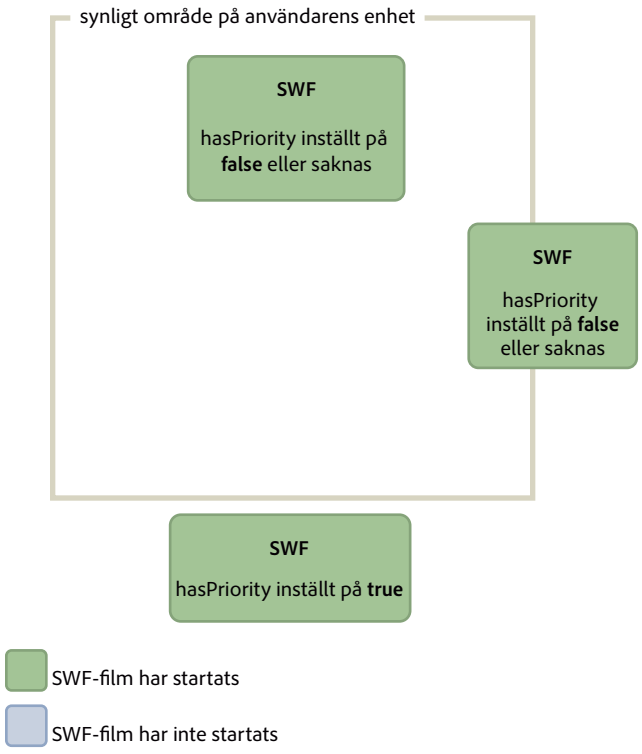
Den här funktionen använder du för att begränsa antalet Flash Player-instanser som startas på en sida. Genom att begränsa antalet instanser får du bättre processor- och batterikapacitet. Syftet är att tilldela SWF-innehåll speciella prioriteringar och att ge visst innehåll företräde före annat innehåll på en sida. Tänk dig följande enkla exempel: En användare tittar på en webbplats och indexsidan innehåller tre olika SWF-filer. En av dem visas, en annan är delvis synlig och den tredje är utanför skärmen, och han/hon behöver rulla. De båda första animeringarna startar normalt, men den sista fördröjs tills den blir synlig. Detta scenario är standardbeteende när parametern `hasPriority` inte är närvarande eller inställd på `false`. Om du vill vara säker på att SWF-filen startas, även när den inte visas, anger du att parametrarna `hasPriority` ska vara `true`. Oberoende av värdet på parametern `hasPriority` så kommer emellertid återgivningen av en SWF-fil som inte visas ständigt att vara tillfälligt stoppad.

Obs! *Om processorresurserna börjar ta slut kommer ingen Flash Player-instans att startas automatiskt, även om parametern* `hasPriority` *är* `true`*. Om nya instanser skapas med JavaScript sedan sidan lästs in, kommer dessa instanser att ignorera* `hasPriority`*-flaggan. Allt pixelinnehåll på 1x1 eller 0x0 startas och förhindrar att hjälp-SWF-filer kommer att fördröjas om webbmastern inte inkluderar* `hasPriority`*-flaggan. SWF-filer startas emellertid fortfarande när användaren klickar på dem. Detta beteende kallas ”click to play” på engelska.*

I bilden nedan visas effekten när du ställer in olika värden för parametern `hasPriority`:



Effekten av olika värden för parametern hasPriority



Effekten av olika värden för parametern hasPriority

Viloläge

I Flash Player 10.1 och AIR 2.5 introduceras en ny funktion för mobilenheter, som bidrar till att minska processorbearbetningen och därmed förlänger batteriets livstid. Funktionen arbetar med bakgrundsbelysningen som finns i många mobilenheter. Om en användare som kör ett mobilprogram till exempel störs och slutar använda telefonen, identifierar körningsmiljön att bakgrundsbelysningen försätts i viloläge. Bildrutefrekvensen minskar då till 4 bildrutor i sekunden (fps) och återgivningen pausas. För AIR-program inleds viloläget också när programmet placeras i bakgrunden.

ActionScript-koden kommer att fortsätta att köras i viloläget som om du ställde in egenskapen `Stage.frameRate` till 4 bildrutor per sekund. Däremot kommer återgivningen att stoppas så användaren kommer inte att se att spelaren körs med 4 bildrutor per sekund. En bildrutefrekvens på 4 bildrutor per sekund valdes, och inte noll, eftersom detta innebär att alla anslutningar förblir öppna (NetStream, Socket och NetConnection). Att byta till noll skulle innebära att öppna anslutningar skulle avbrytas. Ett uppdateringsintervall på 250 ms har valts (4 bildrutor per sekund), eftersom många enhetstillverkare använder denna bildrutefrekvens som uppdateringsintervall. Med det här värdet får körningsmiljön och enheten ungefär samma bildrutefrekvens.

Obs! *När körningsmiljön är i viloläge returnerar egenskapen* `Stage.frameRate` *bildrutefrekvensen för den ursprungliga SWF-filen, i stället för 4 bildrutor per sekund.*

När bakgrundsbelysningen aktiveras på nytt återtas återgivningen. Bildrutefrekvensen återgår till det ursprungliga värdet. Tänk dig en mediespelare där musik spelas. Om skärmen försätts i viloläge svarar körningsmiljön på detta utifrån den typ av innehåll som spelas upp. Nedan följer en lista över situationer och motsvarande beteende för körningsmiljön:

- Bakgrundsbelysningen försätts i viloläge och inget ljud-/videoinnehåll spelas upp: Återgivningen pausas och bildrutefrekvensen anges till 4 bildrutor per sekund.
- Bakgrundsbelysningen försätts i viloläge och ljud-/videoinnehåll spelas upp: Bakgrundsbelysningen fortsätter att vara aktiverad och användarupplevelsen fortsätter.
- Bakgrundsbelysningen går från viloläge till aktivt läge: Bildrutefrekvensen ställs in på den ursprungliga SWF-filens inställning och återgivningen fortsätter.
- Flash Player pausas medan ljud-/videoinnehåll spelas upp: Bakgrundsbelysningen återställs till standardsystembeteendet, eftersom ljud/video inte längre spelas upp.
- Mobilenheten tar emot ett telefonsamtal medan ljud-/videoinnehåll spelas upp: Återgivningen pausas och bildrutefrekvensen anges till 4 bildrutor per sekund.
- Bakgrundsbelysningens viloläge inaktiveras på en mobilenhet: Körningsmiljön fungerar som vanligt.

När bakgrundsbelysningen försätts i viloläge pausas återgivningen och bildrutefrekvensen går långsammare. Den här funktionen sparar på processorkapaciteten, men det går inte att förlita sig på den för att skapa en verklig paus, som exempelvis i ett spel.

Obs! *Ingen ActionScript-händelse skickas när körningsmiljön försätts i eller tas ur viloläge.*

Frysa och tina upp objekt

Frys ned och tina upp objekt med hjälp av händelserna `REMOVED_FROM_STAGE` *och* `ADDED_TO_STAGE`*.*

Om du vill optimera koden ska du alltid frysa och tina upp dina objekt. Det är viktigt för alla objekt att de kan frysas och tinas upp, men detta är speciellt viktigt för visningsobjekt. Även om visningsobjekten inte längre finns i visningslistan och väntar på att samlas in med skräpinsamlaren, kan de ändå använda kod som är processorintensiv. De kan till exempel fortfarande använda Event.ENTER_FRAME. Detta resulterar i att det är viktigt att frysa och tina upp objekt på ett korrekt sätt med händelserna `Event.REMOVED_FROM_STAGE` och `Event.ADDED_TO_STAGE`. I följande exempel visas en interaktion mellan ett filmklipp som spelas upp på scenen och tangentbordet:

```
// 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;
    }
}
```

Filmklipp som samverkar med tangentbordet

När du klickar på knappen Remove tas filmklippet bort från visningslistan.

```
// 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);
}
```

Trots att det tagits bort från visningslistan skickas händelsen `Event.ENTER_FRAME` från filmklippet. Filmklippet körs fortfarande men det återges inte. Du hanterar den här situationen korrekt genom avlyssna de rätta händelserna och ta bort händelseavlyssnare, för att på så sätt förhindra att processorkrävande kod kommer att köras:

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

När du klickar på knappen Show startas filmklippet om, händelsen `Event.ENTER_FRAME` avlyssnas igen och tangentbordet kan användas för att kontrollera filmklippet.

Obs! *Om ett visningsobjekt tas bort från visningslistan är det inte säkert att objektet kommer att frysas om referensen anges som* `null` *sedan det tagits bort. Om skräpinsamlaren inte körs kommer objektet att fortsätta förbruka minne och processorkraft, trots att objektet inte längre visas. Om du vill vara säker på att objektet förbrukar minsta möjliga processorkraft, ska du se till att frysa det fullständigt när du tar bort det på visningslistan.*

Från och med Flash Player 10 och AIR 1.5 finns även följande beteenden. Om spelhuvudet kommer till en tom bildruta fryses visningsobjektet automatiskt, även om du inte har implementerat något frysningsbeteende.

Frysningar också viktiga när du läser in fjärrinnehåll med klassen Loader. När du använde Loader-klassen i Flash Player 9 och AIT 1.0 var det nödvändigt att manuellt frysa innehållet genom att avlyssna händelsen `Event.UNLOAD`, som skickades av LoaderInfo-objektet. Varje händelse måste frysas manuellt, vilket inte alltid var en enkel uppgift. I Flash Player 10 och AIR 1.5 introducerades en viktig ny metod i klassen Loader, som kallas `unloadAndStop()`. Med denna metod kan du ta bort en SWF-fil, automatiskt frysa alla objekt i den inlästa SWF-filen och framtvinga körning av skräpinsamlaren.

I följande kod läses SWF-filen in och tas sedan bort med metoden `unload()`, vilket kräver mer bearbetning och manuell frysning:

```
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();
}
```

Det bästa sättet är att använda metoden `unloadAndStop()`, som hanterar frysning internt och som framtvingar att skräpinsamlingsprocessen körs:

```
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();
}
```

Följande åtgärder inträffar när metoden `unloadAndStop()` anropas:

- Ljud stoppas.
- Avlyssnare registrerade för SWF-filens huvudtidslinje tas bort.
- Tidtagarobjekt stoppas.
- Kringutrustning såsom kameror och mikrofoner frisläpps.
- Alla filmklipp stoppas.
- Utskick från `Event.ENTER_FRAME`, `Event.FRAME_CONSTRUCTED`, `Event.EXIT_FRAME`, `Event.ACTIVATE` och `Event.DEACTIVATE` stoppas.

Aktivera och inaktivera händelser

Använd händelserna `Event.ACTIVATE` och `Event.DEACTIVATE` för att identifiera bakgrundsinaktivitet och optimera programmet på rätt sätt.

Två händelser (`Event.ACTIVATE` och `Event.DEACTIVATE`) kan hjälpa dig att finjustera programmet så att det använder så få processorcykler som möjligt. Med hjälp av de här händelserna kan du avgöra när körningsmiljön får eller tappar fokus. Följaktligen kan du även optimera koden att reagera på förändringar i sammanhanget. Följande kod lyssnar på båda händelserna och ändrar dynamiskt bildrutefrekvensen till noll när programmet tappar fokus. Animeringen kan till exempel tappa fokus när användaren går till en annan flik eller placerar programmet i bakgrunden:

```
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;
}
```

När programmet får fokus igen återställs bildrutefrekvensen till det ursprungliga värdet. I stället för att ändra bildrutefrekvensen dynamiskt kan du göra andra optimeringar, till exempel frysa och tina objekt.

Med händelserna activate och deactivate kan du implementera en mekanism, som liknar funktionen "Pausa och återuppta" i vissa mobilenheter och Netbooks.

Fler hjälpavsnitt

”Programmets bildrutefrekvens” på sidan 53

”Frysa och tina upp objekt” på sidan 27

Musinteraktion

Överväg när det är möjligt att inaktivera musinteraktioner.

När du använder ett interaktivt objekt, till exempel ett MovieClip- eller Sprite-objekt, körs den interna koden för att identifiera och hantera musinteraktion. Att identifiera musinteraktionen kan vara processorkrävande när många interaktiva objekt visas på skärmen och speciellt om de är överlappande. Ett enkelt sätt att undvika denna bearbetning är att inaktivera musinteraktionen för objekt som inte kräver musinteraktion. I följande kod visas exempel på hur du kan använda egenskaperna `mouseEnabled` och `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;
```

Överväg att inaktivera musinteraktion när det är möjligt, eftersom detta belastar processorn mindre, vilket i sin tur minskar batterianvändningen på mobilenheter.

Tidtagar- eller ENTER_FRAME-händelser

Välj antingen tidtagar- eller `ENTER_FRAME`-händelser beroende på innehållt som animeras.

Tidtagare är att föredra framför`Event.ENTER_FRAME`-händelser för innehåll som inte är animerat och som utförs under en lång tid.

I ActionScript 3.0 finns det två sätt att anropa en funktion med specifika intervaller. Det första sättet är att använda `Event.ENTER_FRAME`-händelsen som skickas med visningsobjekt (DisplayObject). Det andra sättet är att använda en tidtagare. ActionScript-utvecklare föredrar ofta `ENTER_FRAME`-händelsen. Händelsen `ENTER_FRAME` skickas för varje bildruta. Detta innebär att det intervall som används för att anropa funktionen är relaterad till den aktuella bildrutehastigheten. Bildrutehastigheten hämtas med egenskapen `Stage.frameRate`. Det kan emellertid i vissa fall vara bättre att använda en tidtagare än att använda händelsen`ENTER_FRAME`. Om du exempelvis inte använder en animering, men vill att koden ska anropas med specifika tidsintervall, kan det vara bättre att använda tidtagaren.

En tidtagare kan uppträda på liknande sätt som en `ENTER_FRAME`-händelse, men en händelse kan skickas utan att vara knuten till en bildrutehastighet. Detta kan medföra vissa optimeringsfördelar. Tänk dig exempelvis ett videospelarprogram. I detta fall behöver du inte använda en hög bildrutehastighet eftersom det endast är programkontrollerna som flyttas.

Obs! *Videon påverkas inte av bildrutehastigheten eftersom videon inte är inbäddad på tidslinjen. Videon laddas i stället dynamiskt genom progressiv nedladdning eller direktuppspelning.*

I detta exempel är bildrutehastigheten så låg som 10 bildrutor per sekund. Tidtagaren uppdaterar kontrollerna med en hastighet av en uppdatering per sekund. Den högre uppdateringshastigheten är möjlig genom metoden `updateAfterEvent()` som finns i objektet TimerEvent. Den här metoden medför att skärmen uppdateras varje gång som det finns behov av att en händelse skickas från tidtagaren. Detta illustreras med följande kod:

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

När du anropar metoden `updateAfterEvent()` ändras inte bildrutehastighen. Det innebär bara att det innehåll som visas på skärmen, och som har ändrats, uppdateras. Tidslinjen forsätter att köras med 10 bildrutor per sekund. Tänk på att tidtagare och `ENTER_FRAME`-händelser inte fungerar helt perfekt på enheter med låg prestanda eller om händelsehanterarfunktionerna innehåller kod som kräver omfattande bearbetningar. Precis som för bildrutehastigheten för SWF-filer kan bildrutehastigheten för uppdateringen av tidtagaren variera i vissa situationer.

Minimera antalet Timer-objekt och registrerade `enterFrame`-hanterare i programmet.

För varje bildruta skickar körtidsmodulen en `enterFrame`-händelse till alla visningsobjekt i visningslistan. Trots att du kan registrera avlyssnare för `enterFrame`-händelsen med flera visningsobjekt innebär det att mer kod körs för varje bildruta. Överväg i stället att använda en centraliserad `enterFrame`-hanterare som verkställer all kod som ska köras i varje bildruta. Genom att centralisera koden är det enklare att hantera all kod som körs ofta.

Detsamma gäller när du använder timer-objekt, det finns risk för belastning när du skapar och skickar händelser från flera timer-objekt. Nedan följer några alternativa förslag om du måste aktivera olika åtgärder vid olika intervall:

- Använd så få timer-objekt som möjligt och gruppera åtgärderna efter hur ofta de inträffar.

  Använd t.ex. en timer som är inställd att aktiveras var 100:e millisekund för åtgärder som utförs ofta. Använd en annan timer inställd att aktiveras var 2000:e millisekund för åtgärder som inte utförs så ofta eller som körs i bakgrunden.

- Använd ett timer-objekt och låt åtgärderna aktiveras i multipler av timer-objektets intervall för `delay`-egenskapen.

  Anta t.ex. att du har vissa åtgärder som ska utföras var 100:e millisekund och andra som ska utföras var 200:e millisekund. I sådana fall ska du använda ett timer-objekt med ett `delay`-värde på 100 millisekunder. I händelsehanteraren `timer` ska du lägga till en villkorssats som bara kör 200 millisekundersåtgärderna varannan gång. Detta visas i följande exempel:

```
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;
}
```

Stoppa Timer-objekt när de inte används.

Om `timer`-händelsehanteraren för ett Timer-objekt endast utför åtgärder under vissa förhållanden ska du anropa timerns `stop()`-metod när inga villkor är true.

Minimera antalet ändringar av visningsobjektets utseende som kan göra att skärmen ritas om i `enterFrame`-händelsen eller Timer-hanterarna.

För varje bildruta ritar återgivningsfasen om den del av scenen som ändrats i bildrutan. Om området som ritas om är stort eller om det är litet, men innehåller ett stort antal eller komplexa visningsobjekt, behöver körtidsmodulen mer tid för återgivningen. Använd funktionen “show redraw regions” i felsökningsversionen av Flash Player eller AIR när du vill testa hur mycket som måste ritas om.

Mer information om att förbättra prestandan för åtgärder som upprepas finns i följande artikel:

- Skriva välkonstruerade, effektiva AIR-program (artikel och exempelprogram av Arno Gourdol)

Fler hjälpavsnitt

”Isolera beteenden” på sidan 65

Interpoleringssyndromet

Minska processoranvändningen genom att begränsa användningen av interpolering vilket leder till minskade processorbearbetningar, minskat minnesutnyttjande och bättre batterikapacitet.

Designers och utvecklare som skapar innehåll för Flash på stationära datorer har en tendens att använda många interpoleringar i sina program. När du producerar innehåll för mobila enheter med låg prestanda ska du försöka minimera användningen av rörelseinterpoleringar. Genom att begränsa användningen körs innehållet snabbare på långsammare enheter.

Kapitel 4: ActionScript 3.0-prestanda

Vector-klassen och Array-klassen

När det är möjligt ska du välja Vector-klassen i stället för Array-klassen.

Vector-klassen ger snabbare skriv- och läsåtkomst än Array-klassen.

Ett enkelt test visar fördelarna med Vector-klassen framför Array-klassen. I följande exempelkod visas en test för Array-klassen:

```
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
```

I följande exempelkod visas en test för Vector-klassen:

```
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
```

Detta exempel kan optimeras ytterligare genom att du anger en längd för vektorn och att den ska vara fast:

```
// 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
```

Om storleken på vektorn inte anges i förväg, kommer storleken att öka när utrymmet för vektorn tar slut. För varje gång som storleken på verktorn ökar tilldelas nya minnesblock. Det aktuella vektorinnehållet kopieras till det nya minnesblocket. Den här extra allokeringen och kopieringen av data påverkar prestandan negativt. Koden här ovan är prestandaoptimerad eftersom den inledande vektorstorleken är angiven. Däremot är koden inte optimerad för underhåll. Du underlättar underhållet genom att spara det återanvända värdet i en konstant som i exemplet här:

```
// 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
```

Försök när det är möjligt att använda API:er för vektorobjektet eftersom de troligtvis hanteras snabbare.

Rit-API

Använd rit-API för att få koden att köras snabbare.

Flash Player 10 och AIR 1.5 har ett nytt rit-API, som förbättrar prestanda för kodkörningen. Denna nya API har inte bättre återgivningsprestanda, men den kan dramatiskt minska antalet kodrader som du behöver skriva. Färre rader med kod kan ge bättre prestanda vid ActionScript-körning.

Denna nya rit-API-funktion innehåller följande metoder:

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

Obs! *Här kommer vi inte att fokusera på metoden* `drawTriangles()` *eftersom den används för 3D. Den här metoden kan emellertid ge bättre ActionScript-prestanda eftersom den hanterar ursprungliga texturmappningar.*

I följande kod anropas rätt metod för varje linje som ritas upp:

```
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 );
```

Nästa kod kommer att köras snabbare än den i det föregående exemplet, eftersom inte lika många kodrader körs här. Ju mer komplex sökväg desto större blir prestandavinsten med att använda metoden `drawPath()`:

```
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 );
```

Metoden `drawGraphicsData()` ger liknande prestandaförbättringar.

Inspelnings- och bubblingshändelser

Använd inspelnings- och bubblingshändelser för att minimera händelsehanterare.

Med händelsemodellen i ActionScript 3.0 introducerades begreppet med inspelnings- och bubblingshändelser. Genom att tillvarata fördelarna med en bubblingshändelse kan du få hjälp med att optimera ActionScript-kodens körningstid. Du kan registrera en händelsehanterare för ett objekt, i stället för flera objekt, för att få bättre prestanda.

Tänk dig ett spel där användaren måste klicka på äpplen så snabbt som möjligt för att förstöra dem. I spelet tas varje äpple som användaren klickat på bort från skärmen och ny poäng läggs till i resultatet. För att kunna avlyssna händelsen `MouseEvent.CLICK` som tar bort äpplet kanske du känner dig frestad att skriva följande kod:

```
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 );
}
```

I den här koden anropas metoden `addEventListener()` för varje Apple-instans. Dessutom tas med hjälp av metoden `removeEventListener()` varje avlyssnare bort när användaren klickar på ett äpple. Händelsemodellen i ActionScript 3.0 erbjuder emellertid en inspelnings- och bubblingsfas för vissa händelser, vilket gör att du kan avlyssna den från ett överordnat InteractiveObject. Det är därför möjligt att optimera koden här ovan och minimera antalet anrop till metoderna `addEventListener()` och `removeEventListener()`. I följande kod används inspelningsfasen för att avlyssna händelser från det överordnade objektet.

```
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 );
}
```

Koden är förenklad och mycket optimerad med endast ett anrop till metoden `addEventListener()` i den överordnade behållaren. Avlyssnare registreras inte längre i Apple-instansen så det finns inget behov att ta bort dem när användaren klickar på ett äpple. Hanteraren `onAppleClick()` kan optimeras ytterligare genom att stoppa spridningen av händelsen så att den inte fortsätter:

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

Bubblingsfasen kan även användas för att spela in händelsen genom att skicka `false` som den tredje parametern till metoden `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 );
```

Standardvärdet för inspelningsfasparametern är `false` så du kan utelämna den:

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

Arbeta med pixlar

Rita pixlar med metoden `setVector()`.

När du ritar pixlar kan du göra några enkla optimeringar bara genom att använda lämpliga metoder i klassen BitmapData. Ett snabbt sätt att rita pixlar är att använda metoden `setVector()`:

```
// 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 );
```

När du använder långsamma metoder såsom `setPixel()` eller `setPixel32()`, ska du använda metoderna `lock()` och `unlock()` för att få förloppen att gå snabbare. I följande kod används metoderna `lock()` och `unlock()` för att förbättra prestandan:

```
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
```

Metoden `lock()` i BitmapData-klassen används för att låsa en bild och förhindra att objekt som refererar till den uppdateras när BitmapData-objektet ändras. Om till exempel ett bitmappsobjekt refererar till ett BitmapData-objekt, kan du låsa BitmapData-objektet, ändra det och sedan låsa upp det. Bitmappsobjekt ändras inte förrän BitmapData-objektet är upplåst. Använd den här metoden tillsammans med `unlock()`-metoden före och efter flera anrop till `setPixel()`- eller `setPixel32()`-metoden för att få bättre prestanda. Genom att `lock()` och `unlock()` kommer skärmen inte att uppdateras i onödan.

Obs! *När pixlar bearbetas på en bitmapp, inte på visningslistan (dubbel buffring), kan detta ibland innebära att prestandan inte förbättras. Om ett bitmappsobjekt inte refererar bitmappsbufferten, kommer du inte att förbättra prestandan genom att använda `lock()` och `unlock()`. I Flash Player upptäcks att bufferten inte refereras och bitmappen kommer inte att återges på skärmen.*

Metoder som itererar över pixlar, till exempel `getPixel()`, `getPixel32()`, `setPixel()` och `setPixel32()`, kommer troligtvis att vara långsamma. Detta gäller speciellt för mobila enheter. Du ska om möjligt använda metoder som hämtar alla pixlar i ett anrop. När du vill läsa pixlar ska du använda metoden `getVector()` som är snabbare än metoden `getPixels()`. Tänk även på att använda API:er som förlitar sig på vektorobjekt när det är möjligt, eftersom de troligtvis körs snabbare.

Reguljära uttryck

Använd metoder i String-klassen, t.ex. `indexOf()`, `substr()` och `substring()`, i stället för reguljära uttryck för grundläggande sökning och extrahering av strängar.

Vissa åtgärder som kan utföras med ett reguljärt uttryck kan även utföras med metoder i String-klassen. Om du t.ex. vill ta reda på om en sträng innehåller en annan sträng kan du antingen använda metoden `String.indexOf()` eller ett reguljärt uttryck. En metod i String-klassen körs snabbare än motsvarande reguljära uttryck och ytterligare ett objekt behöver inte skapas.

Använd en icke hämtande grupp (”(?:xxxx)”) i stället för en grupp (”(xxxx)”) i ett reguljärt uttryck om du vill gruppera element men inte behöver isolera innehållet i gruppen i resultatet.

I reguljära uttryck med medelmåttlig komplexitet grupperar du ofta delar av uttrycket tillsammans. I följande reguljära uttrycksmönster skapar t.ex. parentesen en grupp runt texten ”ab.” Därför tillämpas kvantifieraren ”+” på gruppen i stället för ett tecken:

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

Som standard ”hämtas” innehållet i varje grupp. Du kan hämta innehållet i varje grupp i mönstret som en del av resultatet när det reguljära uttrycket verkställs. Att hämta sådana gruppresultat tar längre tid och kräver mer minne eftersom objekt skapas för att innehålla gruppresultaten. Som ett alternativ kan du använda en icke hämtande gruppsyntax genom att inkludera ett frågetecken och ett kolon efter det inledande parentestecknet. Syntaxen anger att tecknen beter sig som en grupp men inte hämtas av resultatet:

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

Att använda en icke hämtande gruppsyntax är snabbare och använder mindre minne än den vanliga gruppsyntaxen.

Överväg att använda ett alternativt reguljärt uttrycksmönster om ett reguljärt uttryck fungerar dåligt.

Ibland kan mer än ett reguljärt uttrycksmönster användas för att testa efter eller identifiera samma textmönster. Av olika anledningar verkställs vissa mönster snabbare än andra alternativ. Om du fastställer att ett reguljärt uttryck gör att koden körs långsammare än nödvändigt, ska du överväga att använda reguljära uttrycksmönster som uppnår samma resultat Testa de olika alternativa mönstren för att se vilket som är snabbast.

Olika typer av optimeringar

Använd ett TextField-objekt för metoden `appendText()` i stället för operatorn `+=`.

När du arbetar med egenskapen `text` i TextField-klassen ska du använda metoden `appendText()` i stället för operatorn `+=`. Du får bättre prestanda när du använder metoden `appendText()`.

I exempelvis följande kod används operatorn `+=` och slingan tar 1120 ms att slutföra:

```
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
```

I nästa exempel är operatorn `+=` ersatt med metoden `appendText()`:

```
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
```

Koden tar nu 847 ms att slutföra.


Uppdatera textfält utanför slingor när det är möjligt.

Denna kod kan optimeras ytterligare genom att använda en enkel teknik. Att uppdatera textfälten i varje slinga innebär att mycket intern bearbetningskapacitet används. Genom att enbart sammanfoga en sträng och tilldela den till textfältet utanför slingan, så kommer tiden det tar att köra koden att minska avsevärt. Koden tar nu 2 ms att slutföra.

```
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
```

När du arbetar med HTML-text är det tidigare tillvägagångssättet så långsamt så att du i vissa fall får ett `Timeout`-undantag i Flash Player. Ett undantag kan till exempel genereras om den bakomliggande maskinvaran är för långsam.

Obs! *För Adobe® AIR® genereras inte detta undantag.*

```
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 );
```

Genom att tilldela värdet till en sträng utanför slingan så krävs det nu bara 29 ms att slutföra den:

```
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
```

Obs! *I Flash Player 10.1 och AIR 2.5 har klassen String förbättrats så att strängarna använder mindre minne.*


Undvik, när det är möjligt, att använda hakparentesoperatorn.

Om du använder hakparentesoperatorn kan prestandan försämras. Du kan undvika att använda den genom att spara referensen i en lokal variabel. I följande exempel visas ett ineffektivt sätt att använda hakparentesoperatorn:

```
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
```

I följande optimerade version minskas användningen av hakparentesoperatorn:

```
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
```

Textbunden kod minskar, när det är möjligt, antalet funktionsanrop i koden.

Funktionsanrop kan vara kostsamma. Försök minska antalet funktionsanrop genom att flytta textbunden kod. Att flytta textbunden kod är ett bra sätt att optimera för bättre prestanda. Du ska emellertid tänka på att textbunden kod kan göra det besvärligare att återanvända koden och dessutom kan storleken på SWF-filen öka. Vissa funktionsanrop, som exempelvis Math-klassmetoder, är enkla att flytta i texten. I följande kod används metoden `Math.abs()` för att beräkna absoluta värdena:

```
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 beräkningar som utförs i `Math.abs()` kan göras manuellt och infogas:

```
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
```

När du flyttar funktionsanropet och gör det textbundet kommer du att få en kod som körs mer än fyra gånger snabbare. Detta arbetssätt är användbart i många situationer, men du måste tänka på hur det kan påverka återanvändbarheten och underhållet.

Obs! *Kodens storlek har stor påverkan på spelarens prestanda. Om programmet innehåller en stor mängd ActionScript-kod, kommer mycket tid i den virtuella maskinen att användas för att verifiera kod och JIT-kompliering. Egenskapssökningar kan bli långsammare på grund av djupare ärvda hierarkier och eftersom det interna cacheminnet förkastar mer. Om du vill minska kodstorleken ska du undvika Adobe® Flex®-ramverket, TLF-ramverksbiblioteket och ActionScript-bibliotek från tredje part.*


Undvik att utvärdera satser i slingor.

Den går att optimera ytterligare genom att inte utvärdera en stats i en slinga. Följande kod itereras över en matris, men den är inte optimerad eftersom matrislängden utvärderas vid varje iteration:

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

Det är bättre att lagra värdet och sedan återanvända det.

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

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


Använd omvänd ordning för while-slingor.

En while-slinga i omvänd ordning är snabbare än en framåtriktad slinga:

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

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

Dessa tips visar på några sätt att optimera ActionScript och hur en kodrad kan påverka prestanda och minneshantering. Det finns emellertid många andra sätt att optimera ActionScript. Du hittar mer information på följande plats: http://www.rozengain.com/blog/2007/05/01/some-actionscript-30-optimizations/.

Kapitel 5: Återgivningsprestanda

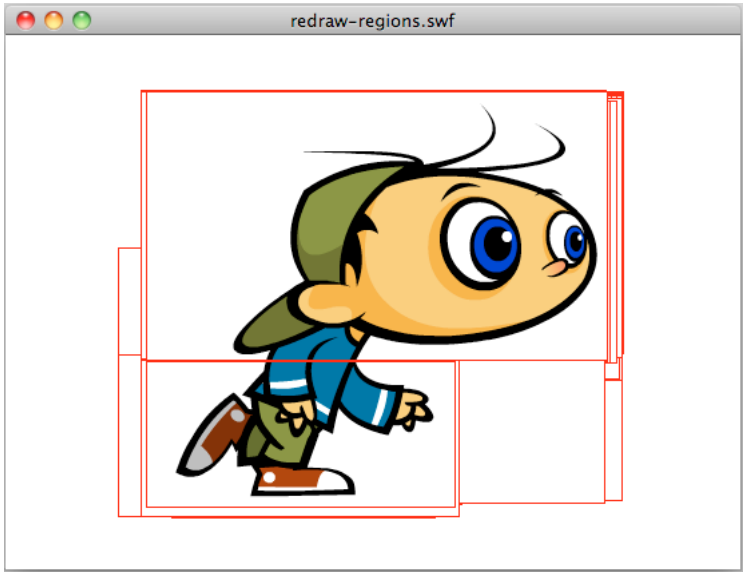
Omritningsområden

Använd alltid alternativet för omritningsområden när du skapar ditt projekt.

För att förbättra återgivningen är det viktigt att du använder alternativet för omritningsområden när du skapar ditt projekt. Med det alternativet kan du se de områden som Flash Player återger och bearbetar. Du kan aktivera det här alternativet genom att markera Visa omritningsområden på snabbmenyn i felsökningsversionen av Flash Player.

Obs! *Alternativet Visa omritningsområden är inte tillgängligt i Adobe AIR eller i den officiella versionen av Flash Player. (I Adobe AIR är snabbmenyn bara tillgänglig i datorprogram, men den har inga inbyggda objekt eller standardobjekt som Visa omritningsområden.)*

Bilden nedan visar hur det aktiverade alternativet fungerar med ett enkelt, animerat MovieClip på tidslinjen:



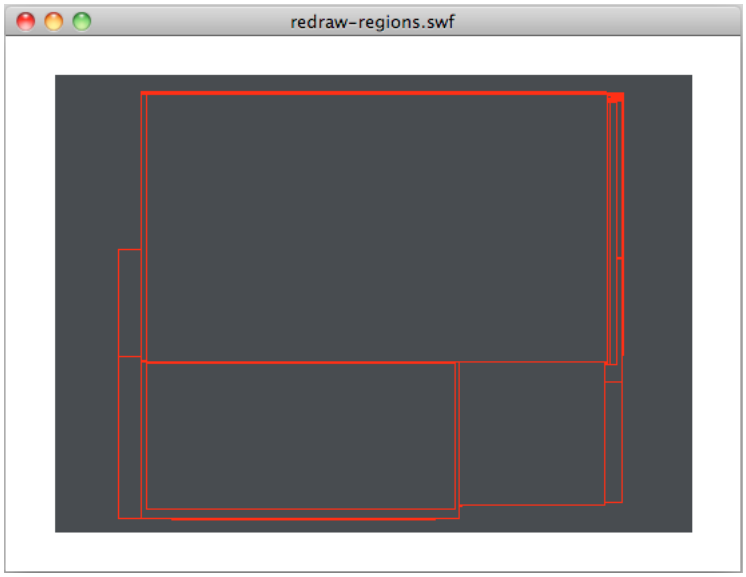
Alternativet för omritningsområden är aktiverat

Du kan även aktivera alternativet programmatiskt med metoden `flash.profiler.showRedrawRegions()`:

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

I Adobe AIR-program är detta det enda sättet att aktivera alternativet för omritningsområden.

Använd omritningsområden för att identifiera optimeringsmöjligheter. Tänk på att trots att vissa visningsobjekt inte syns kan de ändå förbruka processorcykler, eftersom de fortfarande återges. Bilden nedan illustrerar tankegången. En svart vektorform täcker den animerade figuren som springer. Bilden visar att visningsobjektet inte har tagits bort från visningslistan och fortfarande återges. Det är slöseri med processorcykler:



Omritade områden

Om du vill förbättra prestandan anger du egenskapen `visible` för den dolda springande figuren som `false` eller tar bort den helt från visningslistan. Du bör också stoppa dess tidslinje. Dessa steg ser till att visningsobjektet fryses och använder minimalt med processorkraft.

Kom ihåg att använda alternativet för omritningsområden under hela utvecklingsperioden. Om du använder det här alternativet slipper du obehagliga överraskningar, som onödiga omritningsområden och missade optimeringstillfällen, i slutet av projektet.

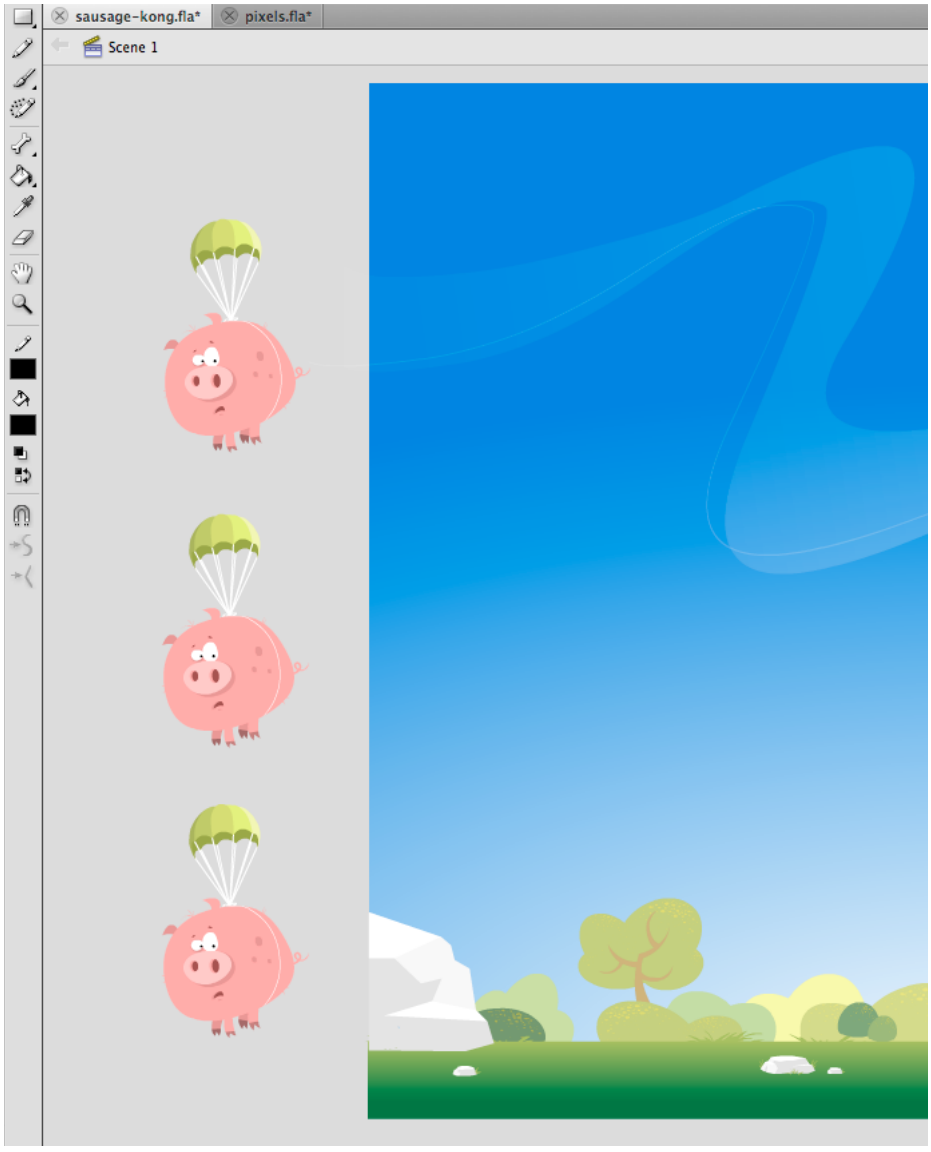
Fler hjälpavsnitt

”Frysa och tina upp objekt” på sidan 27

Innehåll utanför scenen

Undvik att placera innehåll utanför scenen. Placera i stället objekt i visningslistan vid behov.

Undvik också att placera grafiskt innehåll utanför scenen i största möjliga mån. Formgivare och utvecklare placerar ofta element utanför scenen för att de vill återanvända resurser under programmets livstid. Följande bild visar tekniken:



Innehåll utanför scenen

Även om element utanför scenen inte visas på skärmen och inte återges finns de ändå i visningslistan. Körningsmiljön fortsätter att köra interna tester på de här elementen för att se till att de förblir utanför skärmen och att användaren inte samverkar med dem. Därför bör du i största möjliga mån undvika att placera objekt utanför scenen och i stället ta bort dem från visningslistan.

Filmkvalitet

Använd rätt kvalitetsinställningar för scenen för att få bättre återgivning.

När du utvecklar innehåll för mobilenheter med små skärmar, till exempel telefoner, är bildkvaliteten mindre viktig än när du utvecklar för datorer. Om du ställer in kvaliteten för scenen korrekt kan återgivningsprestandan förbättras.

Följande inställningar för scenkvaliteten är tillgängliga:

- `StageQuality.LOW`: Prioriterar uppspelningshastighet framför utseende. Ingen kantutjämning används. Den här inställningen stöds inte i Adobe AIR för datorprogram eller tv-enheter.
- `StageQuality.MEDIUM`: Använder viss kantutjämning, men utjämnar inte skalade bitmappar. Den här inställningen är standardvärde för AIR på mobilenheter, men stöds inte i AIR för skrivbordsprogram eller för tv.
- `StageQuality.HIGH`: (Standard för stationära datorer) Prioriterar utseende framför uppspelningshastighet. Kantutjämning används alltid. Om SWF-filen inte innehåller animeringar utjämnas skalade bitmappar, men om SWF-filen innehåller animeringar utjämnas inte bitmapparna.
- `StageQuality.BEST`: Ger bäst visningskvalitet och tar inte hänsyn till uppspelningshastighet. Alla utdata kantutjämnas och skalade bitmappar utjämnas alltid.

`StageQuality.MEDIUM` ger ofta tillräcklig kvalitet för program på mobilenheter och i vissa fall kan till och med `StageQuality.LOW` räcka. Från och med Flash Player 8 kan text med kantutjämning återges korrekt, även när scenkvaliteten är inställd på `LOW`.

Obs! *På vissa mobilenheter används `MEDIUM` för bättre prestanda i Flash Player-program, trots att kvaliteten har ställts in på `HIGH`. Även om du ställer in att kvaliteten ska vara HIGH är det inte säkert att du märker någon skillnad eftersom skärmar på mobilenheter vanligtvis har bättre upplösning. (Upplösningen kan variera mellan olika enheter.)*

I följande figur är filkvaliteten inställd på `MEDIUM` och textåtergivningen på Kantutjämna för animering:

Medelscenkvalitet och textåtergivning med Kantutjämna för animering

Sceninställningarna påverkar textkvaliteten eftersom korrekt textåtergivningsinställning inte används.

Du kan ange textåtergivningen som Kantutjämning för läsbarhet. Den här inställningen ger en perfekt (kantutjämnad) textkvalitet oberoende av vilken kvalitetsinställning för scenen du använder.

Låg scenkvalitet och textåtergivning med Kantutjämna för läsbarhet

Du kan få samma återgivningskvalitet genom att välja Bitmappstext (ingen kantutjämning) som textåtergivning.

Låg scenkvalitet och textåtergivning med Bitmappstext (ingen kantutjämning)

De två sista exemplen visar att du kan få text med hög kvalitet oberoende av vilken scenkvalitet du använder. Den här funktionen har funnits sedan Flash Player 8 och den kan användas på mobila enheter. Tänk på att Flash Player 10.1 automatiskt växlar till `StageQuality.MEDIUM` på vissa enheter för att förbättra prestandan.

Alfablandning

Undvik om det är möjligt att använda alfaegenskaper.

Undvik att använda effekter som kräver alfablandning när du använder `alfaegenskaper` som exempelvis toningseffekter. När ett visningsobjekt använder alfablandning måste miljön kombinera färgvärdena för alla staplade visningsobjekt och för bakgrundsfärgen för att fastställa den slutliga färgen. Alfablandning kan därför belasta processorn mer än en ogenomskinlig färg. Den här extra bearbetningen kan påverka prestanda negativt på långsamma enheter. Undvik om det är möjligt att använda `alfaegenskaper`.

Fler hjälpavsnitt

”Bitmappscachning” på sidan 54

”Återge textobjekt” på sidan 68

Programmets bildrutefrekvens

I allmänhet ska du använda lägsta möjliga bildrutefrekvens för bättre prestanda..

Ett programs bildrutefrekvens avgör hur mycket tid som är tillgänglig för varje programkods- och återgivningscykel, så som beskrivs i ”Grundläggande om exekvering av körtidskod” på sidan 1. En högre bildrutefrekvens ger jämnare animeringar. När animeringar eller andra visuella ändringar inte sker finns det ofta ingen anledning att ha en hög bildrutefrekvens. En högre bildrutefrekvens förbrukar fler processorcykler och mer batterikraft än en lägre frekvens.

Nedan följer några allmänna riktlinjer för en lämplig bildrutefrekvens i program:

- Om du använder Flex-ramverket kan du använda standardvärdet som inledande bildrutefrekvens.
- En bildrutefrekvensen på minst 20 bildrutor per sekund tillräcklig om programmet innehåller animeringar. Mer än 30 bildrutor per sekund är ofta onödigt.
- Om programmet inte inkluderar animeringar är förmodligen en bildrutefrekvens på 12 bildrutor per sekund lagom.

Den lägsta möjliga bildrutefrekvensen kan variera beroende den aktuella aktiviteten i programmet. Mer information finns i nästa tips, ”Ändra bildrutefrekvensen i programmet dynamiskt”.

Använd en låg bildrutefrekvens om video är det enda dynamiska innehållet i programmet.

Körtidsmodulen spelar upp inläst videoinnehåll med videons ursprungliga bildrutefrekvens oavsett programmets bildrutefrekvens. En låg bildrutefrekvens försämrar inte användarens upplevelse om programmet saknar animeringar och annat visuellt innehåll som snabbt ändras.

Ändra bildrutefrekvensen i programmet dynamiskt.

Du kan definiera programmets inledande bildrutefrekvens i projekt- eller kompileringsinställningarna, men bildrutefrekvensen är inte fast inställd på det värdet. Du kan ändra bildrutefrekvensen genom att ställa in egenskapen `Stage.frameRate` (eller egenskapen `WindowedApplication.frameRate` i Flex).

Ändra bildrutefrekvensen allt efter de aktuella behoven i programmet. Du bör t.ex. sänka bildrutefrekvensen när animeringar inte används i programmet. Höj sedan bildrutefrekvensen när en animerad övergång ska börja. På samma sätt kan du i vanliga fall även sänka bildrutefrekvensen ytterligare om programmet körs i bakgrunden (när det inte är i fokus) Användaren koncentrerar sig förmodligen på ett annat program eller en annan uppgift.

Nedan följer några allmänna riktlinjer som du kan använda som utgångspunkt när du fastställer en lämplig bildrutefrekvens för olika aktiviteter:

- Om du använder Flex-ramverket kan du använda standardvärdet som inledande bildrutefrekvens.
- Ställ in bildrutefrekvensen på minst 20 bildrutor per sekund när animeringar visas. Mer än 30 bildrutor per sekund är ofta onödigt.
- En bildrutefrekvens på 12 bildrutor per sekund är förmodligen lagom när animeringar inte visas.
- Inlästa videoklipp spelas upp med den ursprungliga bildrutefrekvensen oavsett programmets bildrutefrekvens. En bildrutefrekvens på 12 bildrutor per sekund är förmodligen lagom om video är det enda rörliga innehållet i programmet.
- En bildrutefrekvens på 5 bildrutor per sekund är förmodligen lagom om programmet körs i bakgrunden.

- När ett AIR-program inte är synligt är en bildrutefrekvens på 2 bildrutor per sekund eller mindre förmodligen lämpligt. Den här riktlinjen gäller till exempel om ett program minimeras. Det gäller även stationära datorer om egenskapen `visible` för det inbyggda fönstret är `false`.

För program som skapas i Flex har klassen spark.components. WindowedApplication inbyggt stöd för dynamisk ändring av programmets bildrutefrekvens. Egenskapen `backgroundFrameRate` definierar programmets bildrutefrekvens när programmet inte är aktivt. Standardvärdet är 1, vilket ändrar bildrutefrekvensen i ett program som skapats med Spark-ramverket till 1 bildruta per sekund. Du kan ändra bildrutefrekvensen i bakgrunden genom att ställa in egenskapen `backgroundFrameRate`. Du kan även ställa in egenskapen på ett annat värde eller på -1 om du vill stänga av automatisk sänkning av bildrutefrekvensen.

Mer information om att dynamiskt ändra ett programs bildrutefrekvens finns i följande artiklar:

- Minska CPU-användningen i Adobe AIR (Adobe Developer Center-artikel med exempelkod av Jonnie Hallman)
- Skriva välkonstruerade, effektiva AIR-program (artikel och exempelprogram av Arno Gourdol)

Grant Skinner har skapat en klass för sänkning av bildrutefrekvensen. Du kan använda klassen i program om du automatiskt vill sänka bildrutefrekvensen när programmet körs i bakgrunden. Om du vill veta mer och hämta källkoden för klassen FramerateThrottler läser du Grants artikel "Idle CPU Usage in Adobe AIR and Flash Player" på http://gskinner.com/blog/archives/2009/05/idle_cpu_usage.html.

Adaptiv bildrutefrekvens

När du kompilerar en SWF-fil anger du en specifik bildrutefrekvens för filmen. I en begränsad miljö med låg processorhastighet kommer bildrutefrekvensen ibland att gå ner under uppspelningen. För att behålla en acceptabel bildrutefrekvens för användaren hoppar körningsmiljön över återgivning av vissa bildrutor. Genom att hoppa över återgivningen av vissa bildrutor kan bildrutefrekvensen bevaras på en acceptabel nivå.

Obs! *I det här fallet hoppas inga bildrutor över, utan bara återgivningen av bildrutornas innehåll. Koden körs ändå, och visningslistan uppdateras, men uppdateringarna visas inte på skärmen. Det finns inget sätt att ange ett tröskelvärde för bildrutefrekvensen, som anger hur många bildrutor som ska hoppas över, när bildrutefrekvensen inte kan hållas stabil i körningsmiljön.*

Bitmappscachning

Använd bitmappscachning när det är möjligt för komplext vektorinnehåll.

Du kan uppnå en bra optimering med hjälp av funktionen för bitmappscachning. Den här funktionen cache-lagrar ett vektorobjekt, återger det som en bitmapp internt och använder sedan bitmappen för återgivningen. Detta kan resultera i en avsevärd prestandaförbättring, men det kan kräva mycket minneskapacitet. Använd funktionen för bitmappscachning när du har komplext vektorinnehåll, såsom avancerade övertoningar eller text.

Att aktivera bitmappscachning för ett animerat objekt som innehåller komplex vektorgrafik (till exempel text eller övertoningar) leder till bättre prestanda. Om bitmappscachning är aktiverat för ett visningsobjekt, t.ex. ett filmklipp med en tidslinje som spelas, får du dock motsatt resultat. För varje bildruta måste körningsmiljön uppdatera den cachelagrade bitmappen och sedan rita om den på skärmen, vilket kräver många processorcykler. Funktionen för bitmappscachning är att föredra endast när den cachelagrade bitmappen kan genereras en gång och sedan användas utan att den behöver uppdateras.

Om du aktiverar bitmappscachning för ett Sprite-objekt kan objektet flyttas utan att den cachelagrade bitmappen behöver genereras om. Om du ändrar egenskaperna `x` och `y` för objektet leder detta inte till någon omgenerering. Om du däremot försöker rotera det, skalförändra det eller ändra dess alfavärde omgenereras den cachelagrade bitmappen, vilket resulterar i sämre prestanda.

Obs! *Egenskapen `DisplayObject.cacheAsBitmapMatrix`, som finns i AIR och i Packager for iPhone, har inte den här begränsningen. Om du använder egenskapen `cacheAsBitmapMatrix` kan du rotera, skala, skeva och ändra alfavärdet för ett visningsobjekt utan att utlösa en omgenerering av bitmappen.*

En cache-lagrad bitmapp kan använda mer minne än en vanlig filmklippsinstans. Om exempelvis filmklippet på scenen är 250 x 250 pixlar kommer det när det cache-lagras att uppta cirka 250 kB, i stället för 1 kB om det inte cache-lagras.

I följande exempel används ett Sprite-objekt som innehåller en bild av ett äpple. Följande klass kopplas till äppelsymbolen:

```
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
```

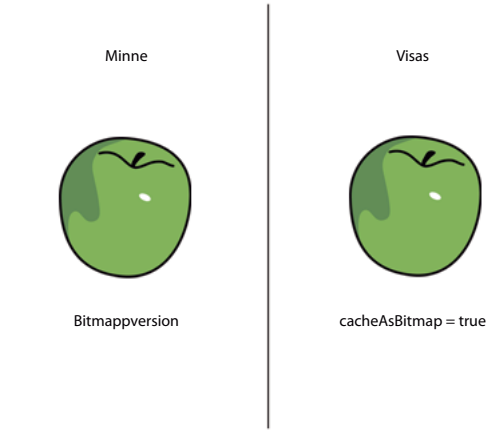
```
        {
            removeEventListener(Event.ENTER_FRAME,handleMovement);
        }

        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();
        }
    }
}
```

I koden används Sprite-klassen i stället för MovieClip-klass eftersom en tidslinje inte behövs för varje äpple. Du ska för att få bästa prestanda använda det enklast möjliga objektet. Därefter instansieras klassen med följande kod:

```
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);
        }
    }
}
```

När användaren klickar med musen kommer äpplena att skapas utan att cache-lagras. När användaren trycker på C-tangenten (tangentkod 67), cache-lagras äppelvektorerna som bitmappar och visas på skärmen. Den här tekniken innebär avsevärda prestandaförbättringar för återgivningen, både på datorer och mobilenheter när processorkapaciteten är låg.

Funktionen för bitmappscachning förbättrar återgivningsprestandan, men den kan snabbt lägga beslag på mycket av minneskapaciteten. Så snart ett objekt har cache-lagrats kommer dess yta att hämtas som en genomskinlig bitmapp och sparas i minnet, vilket visas i följande bild:

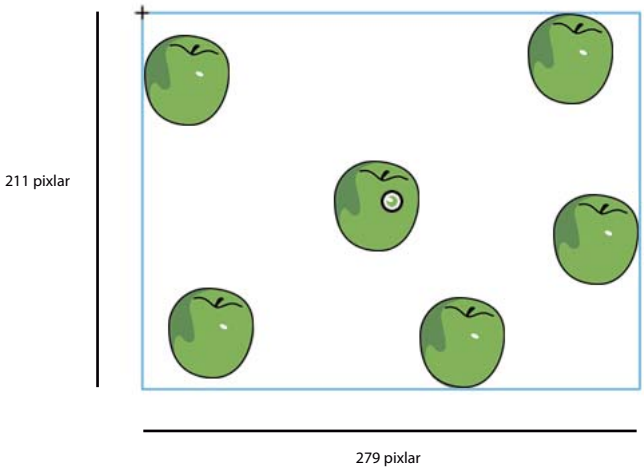


Objekt och dess bitmappsyta sparas i minnet.

I Flash Player 10.1 och AIR 2.5 har minnesanvändningen optimerats på samma sätt som beskrivs i ”Filter och borttagning av dynamiska bitmappar” på sidan 19. Om ett cachelagrat visningsobjekt är dolt eller finns utanför skärmen kommer dess bitmapp att släppas från minnet om den inte används på ett tag.

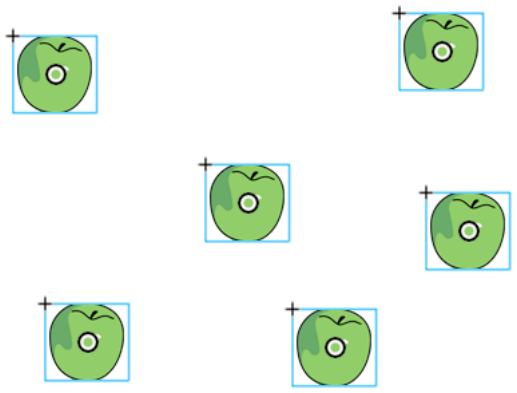
Obs! *Om egenskapen* `opaqueBackground` *för visningsobjektet har ställts in på en viss färg tolkar körningsmiljön visningsobjektet som ogenomskinligt. När det används med egenskapen* `cacheAsBitmap` *skapas en ogenomskinlig 32-bitars bitmapp i minnet. Alfakanalen anges som 0xFF, vilket förbättrar prestandan eftersom ingen genomskinlighet krävs för att rita bitmappen på skärmen. Genom att alfablandningen undviks kommer återgivningen att ske ännu snabbare. Om det aktuella skärmdjupet är begränsat till 16 bitar, kommer bitmappen i minnet att sparas som en 16-bitars bild. Om du använder egenskapen* `opaqueBackground` *innebär detta inte direkt att bitmappscachning aktiveras.*

Om du vill spara minne ska du använda egenskapen `cacheAsBitmap` och aktivera den för varje visningsobjekt i stället för i behållaren. Om bitmappscachning aktiveras i behållaren kommer den slutliga bitmappen att bli mycket större i minnet och en genomskinlig bild med måtten 211 x 279 pixlar kommer att skapas. Bilden använder cirka 229 kB av minnet:



Aktivera bitmappscachning för behållare

Dessutom riskerar du genom att cache-lagra behållaren att hela bitmappen uppdateras i minnet om äpplet börjar röra sig i en bildruta. Om du aktiverar bitmappscachning för enskilda instanser kommer sex 7-kB-ytor att cache-lagras i minnet, vilket innebär att endast 42 kB av minnet används:



Aktivera bitmappscachning för instanser

Genom att ha åtkomst till varje äppelinstans genom visningslistan och anropa metoden `getChildAt()`, kommer referenser att sparas i ett Vector-objekt och ge enklare åtkomst:

```
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);
        }
    }
}
```

Tänk på att bitmappscachning förbättrar återgivningen om det cache-lagrade innehållet inte roteras, skalförändras eller ändras i varje bildruta. Emellertid kommer alla omformningar, förutom de som gäller för förflyttningen på x- och y-axlarna, inte att förbättra återgivningen. I dessa fall kommer den cache-lagrade bitmappskopian att uppdateras för varje omformning som inträffar i visningsobjektet. Uppdatering av den cache-lagrade kopian kan resultera i hög processoranvändning, sämre prestanda och hög batteriförbrukning. Även här gäller att egenskapen `cacheAsBitmapMatrix`, som finns i AIR och Packager for iPhone, inte har denna begränsning.

I följande kod ändras alfavärdet i rörelsemetoden, vilket innebär att opaciteten ändras för äpplet i varje bildruta:

```
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();
}
```

När du använder bitmappscachning minskas prestandan. Genom att ändra alfavärdet tvingas körningsmiljön att uppdatera den cachelagrade bitmappen i minnet så fort alfavärdet ändras.

Filter förlitar sig på bitmappar som uppdateras när spelhuvudet för ett cache-lagrat filmklipp flyttas. Detta innebär att om du använder ett filter så kommer värdet för egenskapen `cacheAsBitmap` automatiskt att ställas in som `true`. I följande bild illustreras ett animerat filmklipp:

Animerat filmklipp

Undvik att använda filter för animerat innehåll eftersom detta kan leda till prestandaförsämringar. I bilden nedan har designern lagt till ett skuggfilter:

Animerat filmklipp med skuggfilter

Detta resulterar bland annat i att bitmappen måste genereras om, om tidslinjen i filmklippet spelas upp. Bitmappen måste även genereras om när innehållet ändras på något annat sätt än genom en enkel x- eller y-omformning. För varje bildruta måste bitmappen ritas om, vilket kräver ytterligare processorresurser, leder till sämre prestanda och förbrukar batterikapacitet.

I följande videofilm visar Paul Trani exempel på hur Flash Professional och ActionScript kan användas för att optimera grafik med bitmappar:

- Optimizing Graphics
- Optimizing Graphics with ActionScript

Omformningsmatriser för cachelagrade bitmappar i AIR

Ange egenskapen `cacheAsBitmapMatrix` när du använder cachelagrade bitmappar i AIR-program för mobiler.

I AIR-mobilprofilen kan du tilldela ett Matrix-objekt till egenskapen `cacheAsBitmapMatrix` för ett visningsobjekt. När du anger den här egenskapen kan du använda valfri tvådimensionell omformning på objektet utan att generera om den cachelagrade bitmappen. Du kan också ändra alpha-egenskapen utan att generera om den cachelagrade bitmappen. Egenskapen `cacheAsBitmap` måste också anges som `true` och objektet får inte ha några 3D-egenskaper.

När du anger egenskapen `cacheAsBitmapMatrix` genereras den cachelagrade bitmappen, även om visningsobjektet finns utanför skärmen, är dolt eller har egenskapen `visible` inställd på `false`. Om du återställer egenskapen `cacheAsBitmapMatrix` med ett matrix-objekt som innehåller en annan omformning genereras den cachelagrade bitmappen också om.

Den omformningsmatris du använder på egenskapen `cacheAsBitmapMatrix` används på visningsobjektet när det återges till bitmappscachen. Om omformningen innehåller en 2x-skalning blir bitmappsåtergivningen alltså två gånger så stor som vektoråtergivningen. Återgivningsfunktionen använder den omvända omformningen på den cachelagrade bitmappen så att den slutliga visningen ser likadan ut. Du kan skala den cachelagrade bitmappen till en mindre storlek för att minska minnesanvändningen, vilket dock kan försämra återgivningskvaliteten. Du kan också skala bitmappen till en större storlek för att öka återgivningskvaliteten i en del fall, vilket i så fall ökar minnesanvändningen. I allmänhet bör du dock använda en identitetsmatris, d.v.s. en matris som inte använder några omformningar, för att undvika förändringar i utseendet, som följande exempel visar:

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

När du väl har angett egenskapen `cacheAsBitmapMatrix` kan du skala, skeva, rotera och översätta objektet utan att utlösa någon omgenerering av bitmappen.

Du kan också ändra alfavärdet i intervallet 0 till 1. Om du ändrar alfavärdet via egenskapen `transform.colorTransform` med en färgomformning måste det alfavärde som används i omformningsobjektet vara mellan 0 och 255. Om du ändrar färgomformningen på något annat sätt genereras den cachelagrade bitmappen om.

Ange alltid egenskapen `cacheAsBitmapMatrix` när du anger `cacheAsBitmap` som `true` i innehåll som skapas för mobilenheter. Du bör dock ha följande möjliga nackdelar i åtanke. Efter att ett objekt har roterats, skalats eller skevats kan den slutliga återgivningen uppvisa skalnings- eller utjämningsdefekter jämfört med en normal vektoråtergivning.

Manuell bitmappscaching

Använd klassen BitmapData för att skapa ett eget bitmappscachningbeteende.

I följande exempel återanvänds en rastrerad bitmappsversion av ett visningsobjekt och referenser görs till samma BitmapData-objekt. När varje visningsobjekt skalas kommer det ursprungliga BitmapData-objektet i minnet varken att uppdateras eller att ritas om. Detta arbetssätt sparar processorresurser och gör att programmet körs snabbare. När ett visningsobjekt skalas kommer bitmappen däri att sträckas ut.

Här visas den uppdaterade `BitmapApple`-klassen:

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

Alfavärdet ändras fortfarande för varje bildruta. I följande kod skickas originalkällans buffert till varje BitmapApple-instans:

```
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);
}
```

Med den här tekniken används endast en liten del av minnet eftersom endast en cache-lagrad bitmapp används och den delas av alla `BitmapApple`-instanser. Dessutom kommer originalkällans bitmapp aldrig att uppdateras, trots ändringar som görs i `BitmapApple`-instanserna, som exempelvis i alfavärdet, rotering och skalning. Använd den här tekniken för att förhindra prestandaförsämringar.

För en mjuk slutlig bitmapp ska du ange att `smoothing`-egenskapen ska vara `true`:

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

    smoothing = true;

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

Du kan även få prestandaförbättringar genom att ändra scenkvaliteten. Ange att scenkvaliteten ska vara `HIGH` före rastreringen och ändra den sedan till låg `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 );
}
```

Att växla scenkvalitet före och efter det att vektorn för en bitmapp ritats upp, kan vara en användbar teknik för att få kantutjämnat innehåll på skärmen. Den här tekniken kan vara effektiv oberoende av den slutgiltiga scenkvaliteten. Du kan till exempel få en kantutjämnad bitmapp med kantutjämnad text även om scenkvaliteten är inställd på `LOW`. Denna teknik går inte att använda med egenskapen `cacheAsBitmap`. I detta fall kommer vektorkvaliteten att uppdateras när du anger att scenkvaliteten ska vara `LOW`, vilket leder till att bitmappsytan i minnet uppdateras och att den slutgiltiga kvaliteten uppdateras.

Isolera beteenden

Isolera om det är möjligt händelser som exempelvis `Event.ENTER_FRAME` i en enda hanterare.

Koden kan optimeras ytterligare genom att isolera händelsen `Event.ENTER_FRAME` i klassen`Apple` i en hanterare. Den här tekniken sparar processorresurser. I följande exempel visas detta annorlunda arbetssätt där klassen BitmapApple inte längre används för att hantera rörelsebeteendet:

```
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;
        }
}
```

I följande kod instansieras äpplena och deras beteenden hanteras i en enskild hanterare:

```
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;
            }
        }
}
```

Resultatet är att en `Event.ENTER_FRAME`-händelse hanterar rörelsen, i stället för att 200 hanterare flyttar varje äpple. Hela animeringen går enkelt att pausa vilket kan vara användbart i spel.

I ett enkelt spel går det att använda följande hanterare:

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

Näst steg blir att få äpplena att interagera med musen eller tangentbordet. För detta krävs förändringar i `BitmapApple`-klassen.

```
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);
        }
}
```

Resultatet är `BitmapApple`-instanser som är interaktiva, som exempelvis traditionella Sprite-objekt. Instanserna är emellertid länkade till en enskild bitmapp, som inte ändras när visningsobjekten omformas.

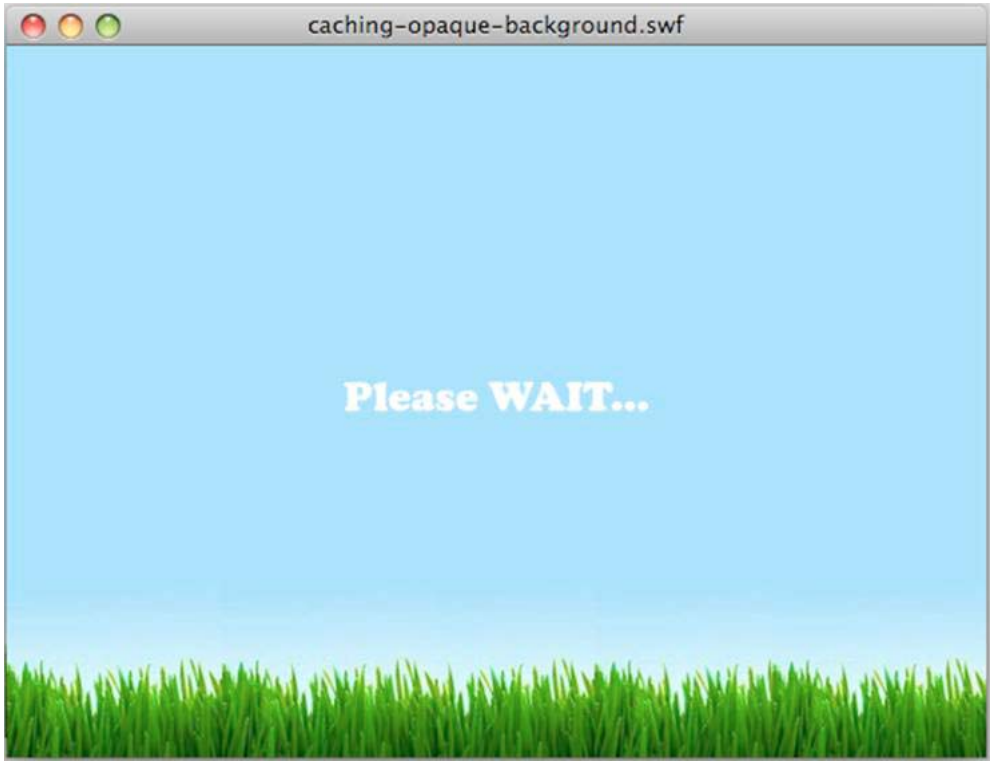
Återge textobjekt

Använd bitmappscachningfunktionen och egenskapen `opaqueBackground` för att få bättre textåtergivning.

Flash-textmotorn har ett antal mycket bra optimeringar. Det krävs emellertid flera klasser för att visa en rad med text. Detta medför att det krävs mycket minnekapacitet och många rader med ActionScript-kod när du skapar ett redigeringsbart textfält med klassen TextLine. Klassen TextLine är bäst lämpad för statisk och icke-redigeringsbar text, där den återges snabbare och är mindre minneskrävande.

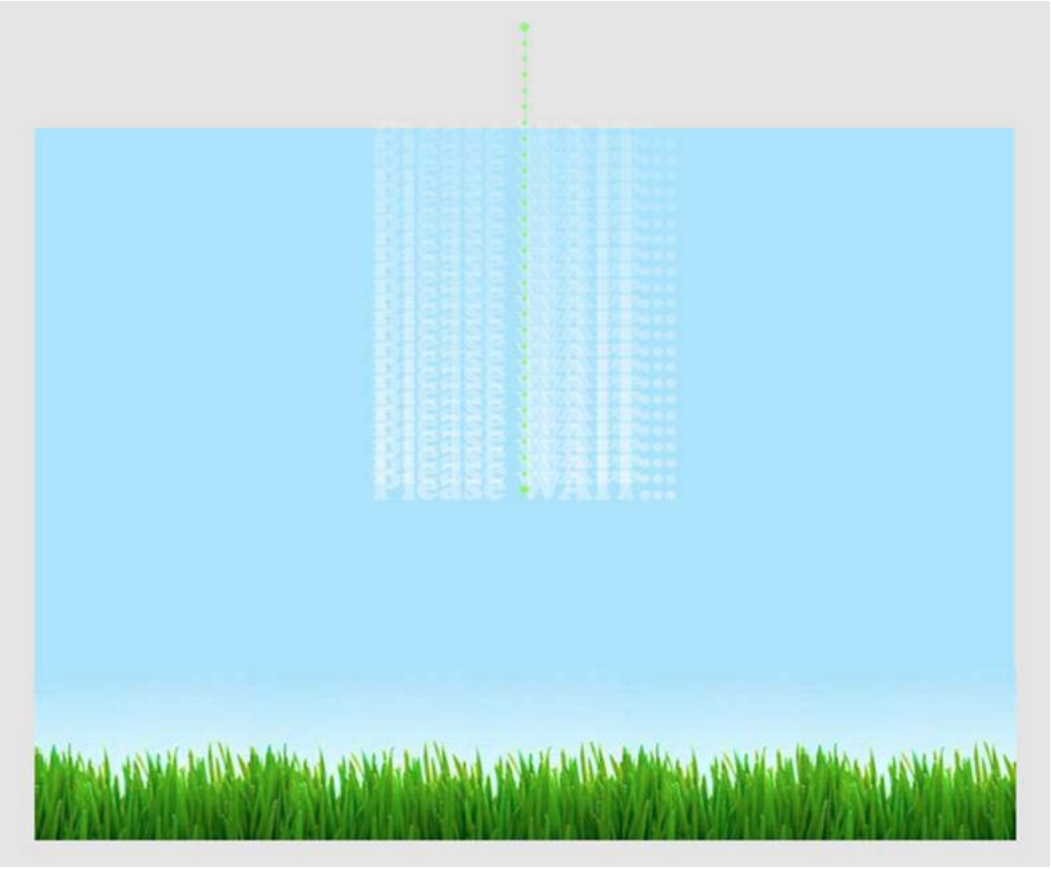
I funktionen för bitmappscachning kan du cache-lagra innehåll såsom bitmappar för att förbättra återgivningsprestandan. Den här funktionen är användbar för avancerat vektorinnehåll, men även för textinnehåll som kräver bearbetning ska återges.

I följande exempel visas hur bitmappscachningfunktionen och egenskapen `opaqueBackground` kan användas för att förbättra återgivningsprestandan. I bilden nedan visas en vanlig välkomstbild som kan visas när en användare väntar på att något ska läsas in:



Välkomstskärm

I nästa bild visas övergången som används för objektet TextField programmatiskt. Textövergången sker sakta från scenöverkanten mot mitten:



Textövergång

Med följande kod skapas övergången. I variabeln `preloader` lagras det aktuella målobjektet för att minimera egenskapssökningar, som kan försämra prestandan:

```
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 );
}
```

Funktionen `Math.abs()` kan göras textbunden och flyttas hit för att minska antalet funktionsanrop och för att få ytterligare prestandaförbättringar. Det bästa är att använda typen int för egenskaperna `destX`- och `destY` så att du får fasta punktvärden. När du använder typen int får du perfekt pixelfästning utan att behöva avrunda värden manuellt i långsamma metoder som `Math.ceil()` eller `Math.round()`. Denna kod avrundar inte koordinaterna till heltal eftersom objekten inte kommer att förflytta sig med mjuka rörelser när värdena avrundas på detta sätt. Rörelserna kan bli ryckiga eftersom koordinaterna fäster mot närmaste avrundade heltal i varje bildruta. Den här tekniken kan emellertid vara användbar när det gäller att fastställa den slutgiltiga positionen för ett visningsobjekt. Använd inte följande kod:

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

Följande kod är mycket snabbare:

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

Den föregående koden kan optimeras ytterligare med hjälp av operatorer för bitvis växling för att dividera värdet.

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

Med funktionen för bitmappscachning är det enklare att återge objekt med hjälp av dynamiska bitmappar. I det aktuella exemplet är filmklippet med TextField-objektet cache-lagrat:

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

Ytterligare ett sätt att förbättra prestandan är att ta bort alfagenomskinlighet. Alfagenomskinlighet belastar körningsmiljön ytterligare vid uppritning av genomskinliga bitmappsbilder, vilket visades i föregående kod. Du kan använda egenskapen `opaqueBackground` för att gå förbi detta och ange en speciell bakgrundsfärg.

När du använder egenskapen `opaqueBackground` kommer fortfarande 32 bitar att användas för bitmappsytan som skapades i minnet. Alfaförskjutningen är emellertid angiven som 255 och ingen genomskinlighet kommer att användas. Detta resulterar inte i att minnesanvändningen för `opaqueBackground`-egenskapen kommer att minska, men återgivningsprestandan kommer att förbättras när funktionen för bitmappscachning används. I följande kod finns alla optimeringarna:

```
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 );
}
```

Animeringen är nu optimerad och bitmappscachningen har optimerats genom att genomskinligheten tagits bort. På mobilenheter bör du överväga om du ska ändra scenkvaliteten till `LOW` och `HIGH` under olika lägen i animeringen när du använder funktionen för bitmappscachning:

```
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 );
    }
}
```

I det här fallet innebär emellertid en förändring av scenkvaliteten att bitmappsytan för TextField-objektet måste genereras om för att matcha den aktuella scenkvaliteten. Av denna anledning är det bäst att inte ändra scenkvaliteten när du använder funktionen för bitmappscachning.

En manuell bitmappscachning skulle ha använts här i stället. Om du vill simulera egenskapen `opaqueBackground` kan du rita upp filmklippet på ett icke-genomskinligt BitmapData-objekt, vilket medför att bitmappsytan inte behöver genereras om.

Den här tekniken fungerar bra för innehåll som inte förändras över tiden. Om emellertid innehållet i textfältet kan ändras ska du överväga att använda en annan strategi. Tänk dig exempelvis ett textfält som uppdateras kontinuerligt med en procentsats som visar hur mycket programmet har läst in. Om textfältet, eller dess visningsobjekt, har cache-lagrats som en bitmapp måste dess yta genereras om varje gång som innehållet ändras. Du kan inte använda manuell bitmappscachning här eftersom visningsobjektet ständigt förändras. Denna ständiga förändring tvingar dig till att manuellt anropa metoden `BitmapData.draw()` för att uppdatera den cache-lagrade bitmappen.

Tänk på att ett textfält, där återgivningen är inställd på Kantutjämna för läsbarhet (oavsett värdet på scenkvaliteten), förblir perfekt kantutjämnat från och med Flash Player 8 (och AIR 1.0). Detta arbetssätt leder till mindre minnesförbrukning men det kräver mer processorbearbetning och återgivningen sker något långsammare är om funktionen för bitmappscachning används.

I följande kod används detta arbetssätt:

```
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 );
    }
}
```

Att använda det här alternativet (Kantutjämna för läsbarhet) för text som rör sig är inte att rekommendera. När texten skalas medför detta alternativ att den försöker förbli justerad, vilket resulterar i en skiftningseffekt. Om visningsobjektets innehåll däremot ändras kontinuerligt, och du behöver skala texten, kan du förbättra prestandan i mobilprogram genom att ställa in kvaliteten på `LOW`. När rörelsen har avslutats återställer du kvaliteten till `HIGH`.

GPU

GPU-återgivning i Flash Player-program

En viktig ny funktion i Flash Player 10.1 är att det nu går att använda grafikprocessorn för att återge grafiskt innehåll på mobila enheter. Tidigare gick det endast att återge grafik med processorn. Genom att använda grafikprocessorn går det att optimera återgivningen av filter, bitmappar, video och text. Tänk på att återgivning med grafikprocessorn inte alltid är lika exakt som programvaruåtergivning. Innehållet kan bli lite otydligt när du använder maskinvaruåtergivning. Dessutom finns i Flash Player 10.1 en begränsning som kan förhindra att Pixel Bender-effekter återges på skärmen. Dessa effekter kan visas som svarta fyrkanter när maskinvaruacceleration används.

Även om Flash Player 10 innehöll en funktion för GPU-acceleration användes grafikprocessorn inte för grafikberäkningar. Den användes endast för att skicka all grafik till skärmen. I Flash Player 10.1 används grafikprocessorn även för grafikberäkningar, vilket kan öka återgivningshastigheten avsevärt. Dessutom minskar belastningen på processorn, vilket är bra för enheter med begränsade resurser som exempelvis mobila enheter.

GPU-läget ställs in automatiskt när innehåll körs på mobila enheter för att få bästa möjliga prestanda. Du behöver inte längre ställa in `wmode` på `gpu` för att få GPU-återgivning, men om du anger `opaque` eller `transparent` för `wmode` inaktiveras GPU-acceleration.

Obs! *Flash Player på stationära datorer använder fortfarande huvudprocessorn (CPU) för programvaruåtergivning. Programvaruåtergivning används eftersom drivrutiner varierar mycket på stationära datorer och eftersom de kan framhäva återgivningsskillnader. Det kan också finnas återgivningsskillnader mellan olika stationära datorer och vissa mobila enheter.*

GPU-återgivning i AIR-program för mobiler

Du kan aktivera maskinvaruacceleration för grafik i ett AIR-program genom att inkludera `<renderMode>gpu</renderMode>` i programbeskrivningsfilen. Du kan inte ändra återgivningsläge under körning. På stationära datorer ignoreras inställningen `renderMode`. Grafikacceleration via grafikprocessorn stöds inte för tillfället.

Begränsningar för GPU-återgivning

Följande begränsningar finns för GPU-återgivning i AIR 2.5:

- Om grafikprocessorn inte kan återge ett objekt visas det inte alls. Processoråtergivning (CPU-återgivning) används inte.
- Följande blandningslägen stöds inte: layer, alpha, erase, overlay, hardlight, lighten och darken.
- Filter stöds inte.
- PixelBender stöds inte.
- Många grafikprocessorer har en maximal texturstorlek på 1 024 x 1 024. I ActionScript blir detta den maximala, slutliga återgivna storleken på ett visningsobjekt efter alla omformningar.
- Du bör inte använda GPU-återgivning i AIR-program som spelar upp video.
- I GPU-återgivningsläget flyttas textfält inte alltid till synliga områden när det virtuella tangentbordet öppnas. Gör något av följande för att garantera att textfältet syns när användaren skriver in text. Placera textfältet på den övre halvan av skärmen eller flytta det till den övre halvan av skärmen när det får fokus.
- GPU-återgivning är inaktiverat på vissa enheter där det inte fungerar tillfredsställande. Läs den viktiga informationen för AIR-utvecklare om du vill veta mer.

Råd om god praxis för GPU-återgivning

Om du följer dessa riktlinjer kan GPU-återgivningen gå snabbare:

- Begränsa antalet objekt som visas på scenen. Det tar tid att återge ett objekt och kombinera det med andra objekt som placerats intill det. Om du inte längre vill visa ett visningsobjekt anger du dess `visible`-egenskap som `false`. Undvik att flytta det utanför scenen, dölja det bakom ett annat objekt eller ange dess `alpha`-egenskap som 0. Om du inte behöver visningsobjektet alls tar du bort det från scenen med `removeChild()`.
- Återanvänd objekt i stället för att skapa och ta bort dem.
- Gör bitmappar som är ungefär 2^n gånger 2^m bitar stora, men inte större. Dimensionerna måste inte vara av tvåpotens, men bör vara i närheten av tvåpotens, utan att vara större. En bild med 31 x 15 pixlar återges till exempel snabbare än en bild med 33 x 17 pixlar. (31 och 15 är precis under tvåpotenserna 2: 32 och 16.)
- Om det är möjligt anger du parametern repeat till false vid anrop till metoden `Graphic.beginBitmapFill()`.
- Undvik att rita för mycket. Använd bakgrundsfärgen som bakgrund. Använd inte stora former i olika lager ovanpå varandra. Varje pixel som måste ritas utgör en belastning.
- Undvik former med långa, tunna spetsar, kanter som korsar sig själva eller många detaljer vid kanterna. Dessa former tar längre tid att återge än visningsobjekt med jämna kanter.
- Begränsa storleken på visningsobjekt.
- Aktivera cacheAsBitMap och cacheAsBitmapMatrix för visningsobjekt vars grafik inte uppdateras så ofta.
- Undvik rit-API:t i ActionScript (klassen Graphics) när du skapar grafik. Om det är möjligt bör du i stället skapa de objekten statiskt från början.
- Skala bitmappsresurser till deras slutliga storlek innan du importerar dem.

GPU-återgivning i AIR 2.0.3 för mobiler

GPU-återgivning är mer begränsat i AIR-program för mobiler som skapas med Packager for iPhone. Grafikprocessorn är bara effektiv för bitmappar, fyllda former och visningsobjekt för vilka egenskapen `cacheAsBitmap` har angetts. Dessutom kan grafikprocessorn återge objekt som roteras och skalas, för objekt där `cacheAsBitmap` och `cacheAsBitmapMatrix` har angetts. Grafikprocessorn används samtidigt för andra visningsobjekt, vilket vanligtvis medför sämre återgivningsprestanda.

Tips om optimering av prestanda för GPU-återgivning

Även om GPU-återgivning kan förbättra prestanda för SWF-innehåll avsevärt, spelar innehållets utformning en viktig roll. Tänk på att inställningar som tidigare har fungerat bra vid programvaruåtergivning ibland inte fungerar så bra med GPU-återgivning. Använd följande tips för att få bra prestanda med GPU-återgivning, utan att programvaruåtergivningen påverkas negativt.

Obs! *Mobilenheter med stöd för maskinvaruåtergivning öppnar ofta SWF-innehåll från webben. Därför är det en bra idé att ta hänsyn till följande råd när du skapar SWF-innehåll för att försäkra dig om att resultatet blir optimalt på alla skärmar.*

- Undvik att använda `wmode=transparent` eller `wmode=opaque` i HTML-parametrar av typen embed. Dessa lägen kan orsaka sämre prestanda. De kan också ge upphov till mindre fel i synkroniseringen av ljud och video vid både programvaru- och maskinvaruåtergivning. Dessutom har många plattformar inte stöd för GPU-återgivning när de här lägena används, vilket försämrar prestanda avsevärt.
- Använd bara det normala blandningsläget och alfablandningsläget. Undvik att använda andra blandningslägen, särskilt lagerblandningsläget. Det är inte alla blandningslägen som kan återges korrekt med GPU-återgivning.
- När en grafikprocessor återger vektorgrafik delas grafiken upp i nät, som består av små trianglar, innan den ritas upp. Den här processen kallas tesselering. Tesselering medför en viss prestandaförlust, som ökar i takt med formens komplexitet. För att minimera prestandapåverkan bör du undvika övergångsformer, eftersom dessa tesseleras vid varje bildruta med GPU-återgivning.
- Undvik kurvor som skär sig själva, mycket tunna böjda områden (som t.ex. en månskära) och komplicerade detaljer längs kanterna på en form. De här formerna är svåra för grafikprocessorn att tesselera till triangulära nät. För att förtydliga kan vi använda två vektorer: en fyrkant på 500 × 500 och en månskära på 100 × 10. Grafikprocessorn kan enkelt återge den stora fyrkanten, eftersom den bara består av två trianglar. Däremot krävs det ett stort antal trianglar för att beskriva månskärans kurva. Det är därför mer komplicerat att återge den formen, även om det handlar om ett färre antal pixlar.
- Undvik stora skaländringar, eftersom dessa också kan medföra att grafikprocessorn tesselerar om grafiken.
- Undvik överritning om det går. Överritning är när flera grafiska element placeras i lager så att de skymmer varandra. Med programvaruåtergivning ritas varje pixel bara en gång. Därför försämras programmets prestanda inte vid programvaruåtergivning, oavsett hur många grafiska element som täcker varandra vid en viss pixelposition. Vid maskinvaruåtergivning däremot ritas alla pixlar för alla element, vare sig de skyms av andra element eller inte. Om två rektanglar överlappar varandra ritas det överlappande området två gånger vid maskinvaruåtergivning, medans det vid programvaruåtergivning bara ritas en gång.

  På en stationär dator, som använder programvaruåtergivning, märker du därför oftast inte någon prestandapåverkan trots överritningen. Men många överlappande former kan påverka prestanda negativt på enheter som använder GPU-återgivning. Det är därför alltid en bra idé att ta bort objekt från visningslistan i stället för att dölja dem.
- Undvik att använda stora, fyllda rektanglar som bakgrunder. Ange i stället bakgrundsfärgen för scenen.
- Undvik i största möjliga mån att använda standardläget för bitmappsfyllning för bitmappsupprepning. Använd i stället läget för bitmappsbegränsning (clamp) för att få bättre prestanda.

Asynkrona åtgärder

Använd asynkrona versioner av åtgärder i stället för synkrona om möjligt.

Synkrona åtgärder körs så fort koden anger det och koden väntar tills de är klara innan den går vidare. Det innebär att de körs i programkodsfasen av bildrutans slinga. Om en synkron åtgärd tar för lång tid, tänjer den bildruteslingans storlek vilket eventuellt kan leda till att visningen stoppas eller ser ryckig ut.

När koden verkställer en asynkron åtgärd behöver den inte nödvändigtvis köras direkt. Din kod och annan programkod i den aktuella körningstråden fortsätter att verkställas. Körtidsmodulen verkställer sedan åtgärden så fort som möjligt samtidigt som den försöker förhindra återgivningsproblem. I vissa fall verkställs den i bakgrunden och körs inte som en del av körtidsbildrutan alls. När åtgärden är klar skickar körtidsmodulen en händelse och din kod kan lyssna efter händelsen för att utföra andra arbeten.

Asynkrona åtgärder schemaläggs och indelas för att undvika återgivningsproblem. Därför är det mycket enklare att skapa ett svarsvilligt program med synkrona åtgärder. Mer information finns i ”Upplevda prestanda jämfört med verkliga prestanda” på sidan 2.

Vissa kompromisser sker dock när åtgärder körs asynkront. Den verkliga körningstiden kan bli längre för asynkrona åtgärder, speciellt åtgärder som slutförs snabbt.

I körningsmiljön är många åtgärder i sig synkrona eller asynkrona, och du kan inte välja hur de ska köras. I Adobe AIR finns det dock tre typer av åtgärder som du välja att utföra synkront eller asynkront:

- Åtgärder i klasserna File och FileStream

  Många åtgärder i File-klassen kan utföras synkront eller asynkront. Metoderna för att kopiera eller ta bort en fil eller katalog och visa innehållet i en katalog har t.ex. alla asynkrona versioner. Metoderna har suffixet ”Async” som läggs till i namnet för den asynkrona versionen. Om du t.ex. vill ta bort en fil asynkront ska du anropa metoden `File.deleteFileAsync()` i stället för metoden `File.deleteFile()`.

  Hur du öppnar ett FileStream-objekt avgör om åtgärderna utförs asynkront eller inte när ett FileStream-objekt används för att läsa från eller skriva till en fil. Använd metoden `FileStream.openAsync()` för asynkrona åtgärder. Skrivning av data utförs asynkront. Läsning av data utförs i segment vilket innebär att data är tillgängliga en del i taget. I synkront läge läser FileStream-objektet däremot hela filen innan koden verkställs.

- Lokala SQL-databasåtgärder

  Alla åtgärder som verkställs via ett SQLConnection-objekt kan verkställas i synkront eller asynkront läge när du arbetar med en lokal SQL-databas. För att ange att åtgärder ska verkställas asynkront ska du öppna anslutningen till databasen med metoden `SQLConnection.openAsync()` i stället för metoden `SQLConnection.open()`. Databasåtgärder verkställs i bakgrunden när de körs asynkront. Databasmotorn körs inte i körtidsmodulens bildruteslinga alls vilket innebär att det är mycket mindre troligt att databasåtgärder orsakar återgivningsproblem.

  Information om andra strategier för att förbättra prestandan med en lokal SQL-databas finns i ”SQL-databasprestanda” på sidan 89.

- Fristående Pixel Bender-skuggningar

  Med klassen ShaderJob kan du köra en bild eller datauppsättning genom en Pixel Bender-skuggning och få åtkomst till rådataresultatet. Skuggningen verkställs asynkront som standard när du anropar metoden `ShaderJob.start()`. Det sker i bakgrunden utan att körtidsmodulens bildruteslinga används. För att tvinga ett ShaderJob-objekt att verkställas synkront (rekommenderas inte) ska du skicka värdet `true` till den första parametern i metoden `start()`.

I tillägg till dessa inbyggda mekanismer för att köra kod asynkront kan du även strukturera din egen kod så att den körs asynkront i stället för synkront. Om du skriver kod som ska utföra en potentiellt långvarig åtgärd kan du strukturera koden så att den verkställs i delar. Genom att dela upp koden kan körtidsmodulen utföra återgivning mellan kodkörningsblocken vilket minskar eventuella återgivningsproblem.

Flera tekniker för att dela upp kod anges nedan. Huvudtanken bakom alla dessa tekniker är att koden enbart skrivs för att utföra en del av arbetet vid ett visst tillfälle. Du kan spåra vad koden gör och när den slutar att arbeta. Du kan t.ex. använda en mekanism som ett Timer-objekt för att upprepade gånger kontrollera om det finns arbete kvar att utföra och utföra ytterligare delar av arbetet tills det är klart.

Det finns några etablerade mönster för att strukturera kod så att arbetet delas upp på det här sättet. Följande artiklar och kodbibliotek beskriver mönstren och innehåller kod som hjälper dig att implementera dem i dina program:

- Asynkrona körningar i ActionScript (artikel av Trevor McCauley med mer bakgrundsinformation samt flera exempel på implementering)
- Tolka och återge stora mängder data i Flash Player (artikel av Jesse Warden med bakgrundsinformation samt exempel på två sätt, ”builder pattern” och ”green threads”)
- Green Threads (artikel av Drew Cummins som beskriver tekniken ”green threads” med exempel på källkod)
- greenthreads (bibliotek med öppen källkod av Charlie Hubbard för implementering av ”green threads” i ActionScript. Se greenthreads, snabbstart för mer information.)
- Trådar i ActionScript 3 på http://www.adobe.com/go/learn_fp_as3_threads_se (artikel av Alex Harui, inklusive ett exempel på implementering av tekniken ”pseudo threading”)

Genomskinliga fönster

Överväg att använda ett ogenomskinligt, rektangulärt programfönster i stället för ett genomskinligt fönster i AIR-program för datorer.

Om du vill använda ett ogenomskinligt inledande fönster i ett AIR-program för datorer anger du följande värde i XML-programbeskrivningsfilen:

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

För fönster som skapas med programkod ska du skapa ett NativeWindowInitOptions-objekt med egenskapen `transparent` inställd på `false` (standardvärdet). Skicka det till NativeWindow-konstruktorn när NativeWindow-objektet skapas:

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

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

För en fönsterkomponent i Flex ska du kontrollera att komponentens egenskap för genomskinlighet är inställd på false (standardvärdet) innan Window-objektets `open()`-metod anropas.

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

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

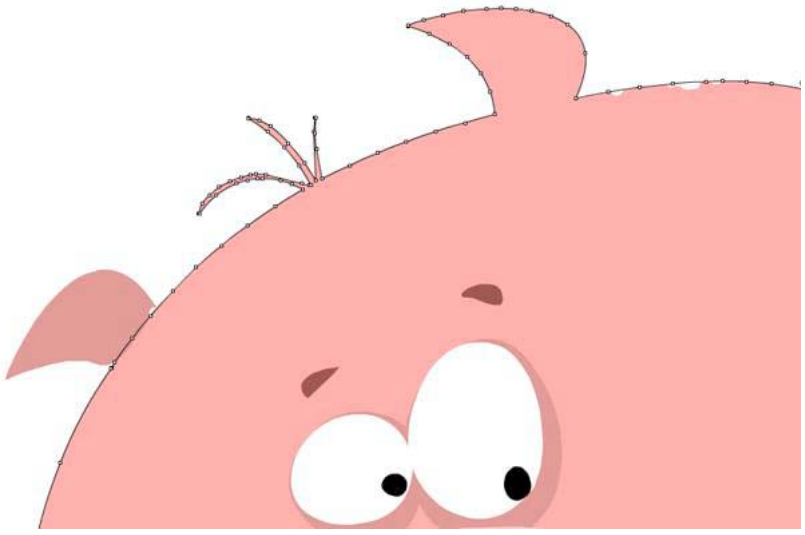
Ett genomskinligt fönster visar eventuellt en del av användarens skrivbord eller andra programfönster bakom ditt programfönster. Det innebär att körtidsmodulen använder mer resurser när ett genomskinligt fönster återges. Ett rektangulärt ogenomskinligt fönster, oavsett om det använder operativsystemets färg eller en anpassad färg medför inte samma återgivningsarbete.

Använd bara ett genomskinligt fönster om det är viktigt att en annan form än rektangulärt används eller om bakgrundsinnehåll ska kunna ses genom programfönstret.

Utjämning av vektorformer

Utjämna former för att förbättra återgivningsprestanda.

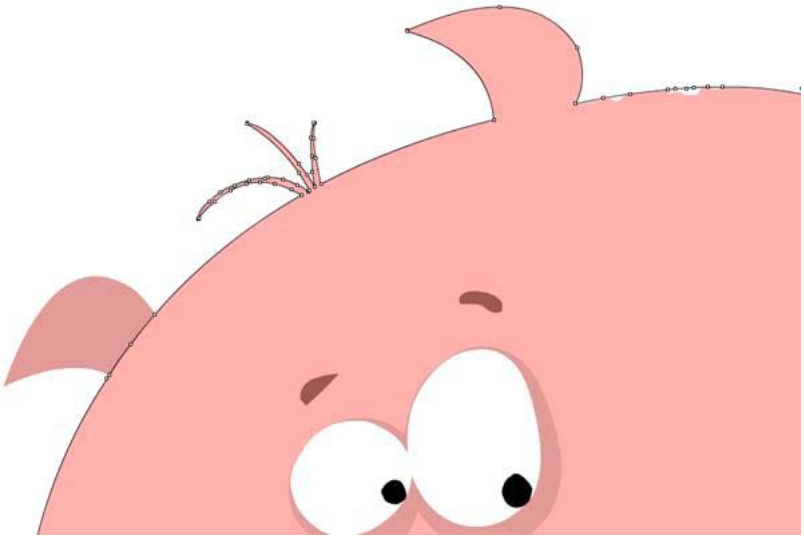
Till skillnad från bitmappar kräver återgivning av vektorinnehåll många beräkningar, särskilt om det rör övertoningar och komplexa banor med många kontrollpunkter. Som designer eller utvecklare måste du försäkra dig om att formerna optimeras tillräckligt. Följande bild visar ej förenklade banor med många kontrollpunkter:



Ej optimerade banor

Om du använder verktyget Utjämna i Flash Professional kan du ta bort extra kontrollpunkter. Det finns ett motsvarande verktyg i Adobe® Illustrator®, och det totala antalet punkter och banor visas på panelen Dokumentinformation.

Utjämning tar bort extra kontrollpunkter, vilket minskar den slutliga storleken på SWF-filen och förbättrar återgivningsprestanda. Nästa bild visar samma banor efter utjämning:



Optimerade banor

Förutsatt att du inte förenklar banorna alltför mycket påverkar den här optimeringen inte det visuella intrycket. Du kan däremot förbättra den genomsnittliga bildrutefrekvensen i det slutliga programmet avsevärt genom att förenkla komplicerade banor.

Kapitel 6: Optimera nätverksinteraktion

Förbättringar för nätverksinteraktion

I Flash Player 10.1 och AIR 2.5 introduceras en uppsättning nya funktioner för nätverksoptimering på alla plattformar, inklusive cirkulär bufferthantering och smart sökning.

Cirkulär bufferthantering

När du överför medieinnehåll till mobilenheter kan du träffa på problem som du nästan aldrig förväntar dig på vanliga datorer. Det är exempelvis vanligare att du får slut på diskutrymmet eller minnet. Vid inläsning av video hämtas och cachelagras hela FLV-filen (eller MP4-filen) på hårddisken i datorversionerna av Flash Player 10.1 och AIR 2.5. Därefter spelas videon upp från cachefilen. Det är ovanligt att diskutrymmet tar slut. Om detta skulle inträffa avbryts videouppspelningen på datorn.

Det är betydligt vanligare att en mobilenhet får slut på diskutrymmet. Om diskutrymmet tar slut avbryts dock inte uppspelningen, som den gör på datorn. I stället återanvänds cachefilen genom att körningsmiljön börjar skriva till den igen från början. Användaren kan fortsätta att titta på videon. Användaren kan inte söka i det område som skrivs om, förutom i början av filen. Cirkulär bufferthantering startas inte som standard. Den kan startas under uppspelningen och även i början av uppspelningen om filmen är större än diskutrymmet eller RAM-minnet. För körningsmiljön krävs minst 4 MB RAM-minne eller 20 MB ledigt utrymme på hårddisken för att det ska gå att använda cirkulär bufferthantering.

Obs! *Om enheten har tillräckligt med diskutrymme fungerar mobilversionen av miljön på samma sätt som på datorn. Tänk på att en buffert i RAM-minnet används som reserv om det inte finns tillräckligt med minne i enheten eller om disken är full. Ett gränsvärde för storleken på cache-filen och RAM-bufferten kan ställas in vid kompileringen. Vissa MP4-filer har en struktur som kräver att hela filen måste laddas ned innan uppspelningen kan börja. Sådana filer identifieras och laddas inte ned om det inte finns tillräckligt med diskutrymme, och MP4-filen kan i så fall inte spelas upp. Det bästa kan vara att inte begära nedladdning av dessa filer över huvud taget.*

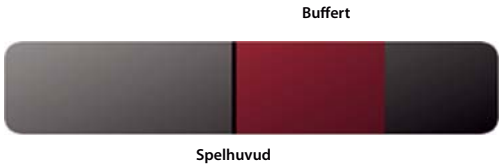
Som utvecklare ska du tänka på att endast söka efter arbeten inom gränserna för den cache-lagrade strömmen. `NetStream.seek()` kan ibland misslyckas om förskjutningen ligger utanför intervallet och i detta fall skickas en `NetStream.Seek.InvalidTime`-händelse.

Smart sökning

Obs! *För funktionen för smart sökning krävs Adobe® Flash® Media Server 3.5.3.*

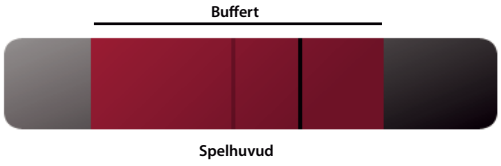
I Flash Player 10.1 och AIR 2.5 introduceras ett nytt beteende, som kallas smart sökning och som förbättrar prestanda för direktuppspelad video. Om användaren söker ett mål inom buffertens gränser återanvänds bufferten så att sökningen blir omedelbar. I tidigare versioner av körningsmiljön återanvändes inte bufferten. Om en användare till exempel spelade upp en video från en direktspelande server, med bufferttiden var inställd på 20 sekunder (`NetStream.bufferTime`), och användaren försökte söka 10 sekunder framåt, skulle alla buffertdata kastas och de 10 sekunder som redan lästs in skulle inte återanvändas. Detta beteende innebar att miljön var tvungen begära nya data från servern mycket oftare, vilket resulterade i dålig uppspelningsprestanda på långsamma anslutningar.

Bilden nedan visar hur bufferten fungerade i tidigare versioner av körningsmiljön. Med egenskapen `bufferTime` anges antalet sekunder som ska laddas in i förväg, så att bufferten kan användas för att fortsätta spela upp videon om anslutningshastigheten skulle försämras:

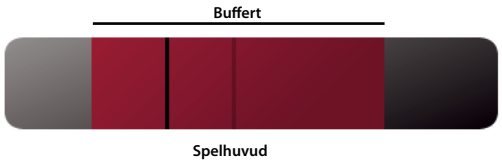


Buffertbeteende innan funktionen smart sökning fanns

Med funktionen för smart sökning används nu bufferten för att ge direkt framåt- eller bakåtsökning när användaren drar videon framåt eller bakåt. I följande bild visas det nya beteendet:



Framåtsökning med funktionen smart sökning



Bakåtsökning med funktionen smart sökning

Vid smart sökning återanvänds bufferten när användaren söker framåt eller bakåt vilket gör uppspelningsupplevelsen snabbare och enklare. En av fördelarna med det nya beteendet är att de som publicerar videon kan spara på bandbredden. Om sökningen däremot går utanför buffertgränserna används standardbeteendet, och nya data begärs från servern.

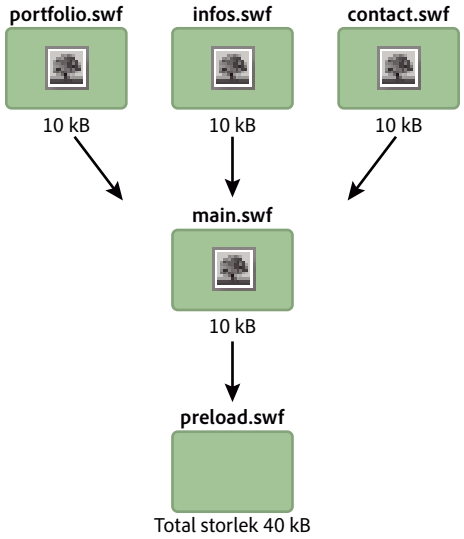
Obs! *Detta beteende gäller dock inte för hämtning av progressiv video.*

Om du vill använda smart sökning anger du `NetStream.inBufferSeek` som `true`.

Externt innehåll

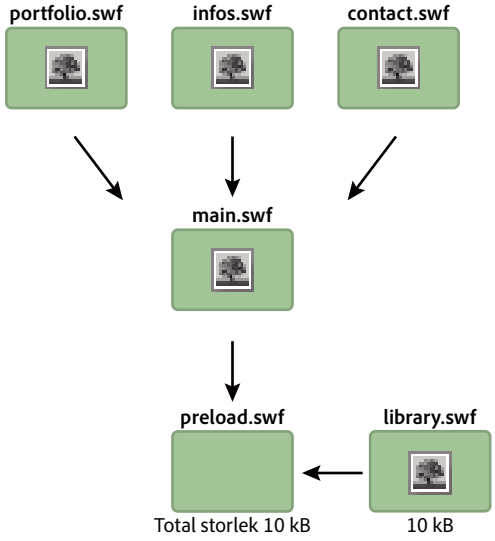
Dela upp ditt program i flera SWF-filer.

Mobilenheter kan ha begränsad åtkomst till nätverket. Om du vill läsa in ditt innehåll snabbt ska du dela upp programmet i flera SWF-filer. Försök genomgående att återanvända kodlogik och resurser i hela programmet. I bilden nedan visas ett exempel på ett program som delats upp i flera SWF-filer:



Program som delats upp i flera SWF-filer

I detta exempel innehåller varje SWF-fil en egen kopia av samma bitmapp. Denna dubblering kan undvikas om du använder ett delat bibliotek under körningen. Se bilden nedan.



Använda en RSL-fil (bibliotek som delas vid körning)

När du använder denna teknik kommer ett delat bibliotek under körningen att läsas in för att bitmappen ska bli tillgänglig för andra SWF-filer. I klassen ApplicationDomain sparas alla definitioner som har lästs in och de görs tillgängliga under körningen med metoden `getDefinition()`.

Ett bibliotek som delas vid körningen kan även innehålla all kodlogik. Hela programmet kan uppdateras under körningen utan att omkompilering krävs. I följande kod läses ett delat bibliotek in under körningen och definitionen som finns i SWF-filen hämtas under körningen. Denna teknik kan användas för teckensnitt, bitmappar, ljud eller för andra ActionScript-klasser:

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

Det blir enklare att hämta definitionen om du läser in klassdefinitionerna i den inlästa SWF-filens programdomän:

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

Nu kan klasserna som är tillgängliga i den inlästa SWF-filen användas när du anropar metoden `getDefinition()` i den aktuella programdomänen. Du får även tillgång till klasserna genom att anropa metoden `getDefinitionByName()`. Med den här tekniken sparar du bandbredd eftersom teckensnitt och stora resurser endast läses in en gång. Resurser exporteras aldrig i någon annan SWF-fil. Den enda begränsningen är att programmet måste testas och köras genom filen loader.swf. Den här filen läser först in resursen och därefter de olika SWF-filerna som programmet består av.

Indata- och utdatafel

Se till att det finns händelsehanterare och felmeddelanden för in-/utdatafel.

För mobilenheter kan nätverket vara mindre tillförlitligt än det för datorer som är anslutna till Internet med en hög hastighet. När det gäller åtkomst till externt innehåll på mobilenheter finns det två begränsningar; tillgänglighet och hastighet. Du ska därför se till att resurserna är enkla och att det finns hanterare för varje IO_ERROR-händelse så att du får återkoppling från användaren.

Tänk dig till exempel en användare som besöker din webbplats via en mobilenhet och plötsligt tappar nätverksanslutningen mellan två tunnelbanestationer. En dynamisk resurs har då lästs in när anslutningen bröts. För en dator kan du använda en tom händelsehanterare för att förhindra att ett körningsfel visas, eftersom detta scenario nästan aldrig inträffar. På en mobilenhet måste du emellertid hantera situationen med mer än bara en tom händelseavslyssnare.

Följande kod hanterar inte ett in-/utdatafel. Använd den inte så som den visas.

```
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;
}
```

Ett bättre sätt är att hantera sådana fel och ge användaren ett felmeddelande. Följande kod hanterar detta korrekt:

```
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
}
```

Ett tips är att komma ihåg att erbjuda användaren att läsa in innehållet på nytt. Detta beteende kan implementeras i hanteraren `onIOError()`.

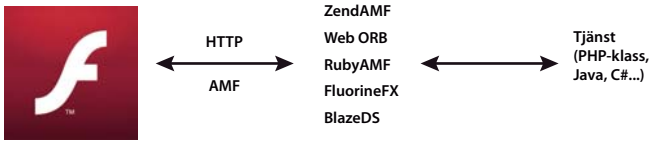
Flash Remoting

Använd Flash Remoting och AMF för att optimera kommunikationen för klient-server-data.

Du kan använda XML för att läsa in fjärrinnehåll till SWF-filer. XML är emellertid vanliga textfiler, som läses in och tolkas av körningsmiljön. XML fungerar bäst för program där en begränsad mängd innehåll läses in. Om du utvecklar ett program som läser in stora mängder data ska du överväga om du inte ska använda Flash Remoting-tekniken och AMF (Action Message Format) i stället.

AMF är ett binärt format som används för att dela data mellan en server och körningsmiljön. När du använder AMF minskas mängden data och hastigheten för överföringen förbättras. Eftersom AMF är ett ursprungligt format för körningsmiljön undviker du, genom att skicka AMF-data till miljön, minnesintensiv serialisering och avserialisering på klienten. Den fjäranslutna gatewayen hanterar dessa uppgifter. När du skickar en ActionScript-datatyp till en server, kommer serialiseringen att hanteras i fjärrgatewayen på serversidan. Gatewayen skickar dessutom motsvarande datatyp till dig. Den här datatypen är en klass som skapas på servern. Den innehåller en uppsättning metoder som kan anropas från körningsmiljön. Flash Remoting-gatewayer innehåller ZendAMF, FluorineFX, WebORB och BlazeDS, en officiell Java Flash Remoting-gateway från Adobe med öppen källkod.

I följande bild illustreras begreppet Flash Remoting:



Flash Remoting

I följande exempel används NetConnection-klassen för att ansluta till en 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 ?");
```

Att ansluta till en fjärrgateway är relativt enkelt. Att använda Flash Remoting kan emellertid göras ännu enklare om du använder RemoteObject-klassen som finns i Adobe® Flex® SDK.

Obs! *Externa SWC-filer, som exempelvis de från Flex-ramverket, kan användas i Adobe® Flash® Professional-projekt. När du använder SWC-filer kan du använda RemoteObject-klassen och dess beroenden utan att använda Flex SDK för övrigt. Om du är en avancerad utvecklare kan du även kommunicera med en fjärrgateway direkt genom raw-Socket-klassen om det skulle behövas.*

Onödiga nätverksåtgärder

Cachelagra resurser lokalt när de har lästs in i stället för att läsa in dem från nätverket varje gång de behövs.

Om programmet läser in resurser som media eller data kan du cachelagra dessa genom att spara dem på den lokala enheten. För resurser som ändras ofta kan du överväga att regelbundet uppdatera cachen. Programmet kan t.ex. kontrollera efter en ny version av en bildfil en gång per dag eller varannan timme.

Du kan cachelagra resurser på flera sätt beroende på resursens typ och egenskaper:

- Medieresurser som bilder och video: Spara filen i filsystemet med klasserna File och FileStream
- Enskilda datavärden eller små uppsättningar av data: spara värdena som lokala, delade objekt med klassen SharedObject
- Större uppsättningar av data: spara data i en lokal databas eller serialisera data och spara dem som en fil

För cachelagring av datavärden finns en ResourceCache-klass, som utför inläsningen och cachningen, hos AS3CoreLib-projektet med öppen källkod.

Kapitel 7: Arbeta med media

Video

Information om hur du optimerar prestanda för video på mobilenheter finns i Optimize web content for mobile delivery på webbplatsen Adobe Developer Connection.

Läs särskilt följande avsnitt (på engelska):

- *Playing video on mobile devices*
- *Code samples*

De här avsnitten innehåller information om hur du utvecklar videospelare för mobilenheter, som:

- Riktlinjer för videokodning
- God praxis
- Hur du profilerar videospelarens prestanda
- En videospelarimplementering

StageVideo

Använd klassen StageVideo för att utnyttja maskinvaruacceleration för att presentera video.

Information om hur du använder StageVideo-objektet finns i Använda klassen StageVideo för maskinvaruaccelererad presentation i Utvecklarhandbok för ActionScript 3.0.

Ljud

Från och med Flash Player 9.0.115.0 och AIR 1.0 kan körningsmiljön spela upp AAC-filer (AAC Main, AAC LC och SBR). En enkel optimering är att använda AAC-filer i stället för MP3-filer. Med AAC-formatet får du bättre kvalitet och mindre filstorlekar än om du använder MP3-formatet med samma bithastighet. Genom att minska filstorleken minskas bandbredden vilket är en viktig faktor för mobila enheter som inte har tillgång till Internet-anslutningar med hög hastighet.

Maskinvaruavkodning av ljud

Precis som för videoavkodning krävs för ljudavkodning höga processorcykler och den kan optimeras genom att utnyttja tillgänglig maskinvara på enheten. Flash Player 10.1 och AIR 2.5 kan identifiera och använda maskinvaruljuddrivrutiner för att förbättra prestandan vid avkodning av AAC-filer (LC-, HE/SBR-profiler) eller MP3-filer (PCM stöds inte). Processoranvändningen minskar avsevärt vilket resulterar i bättre batteriutnyttjande och dessutom frigörs processorn för andra operationer.

Obs! *När AAC-formatet används stöds inte huvudprofilen för AAC på enheter eftersom de flesta enheter saknar stöd för den.*

Maskinvaruavkodning av ljud sker automatiskt för användaren och utvecklaren. När en ljudström börjar spelas upp i körningsmiljön kontrolleras maskinvaran först, precis som vid en videouppspelning. Om det finns en maskinvarudrivrutin, och om ljudformatet stöds, börjar maskinvaruavkodningen av ljudet. Även om avkodningen av den inkommande AAC- eller MP3-strömmen hanteras av maskinvaran är det inte alltid den klarar av alla effekter. Det händer bland annat att maskinvaran inte kan bearbeta ljudmixningar och omsamplingar på grund av interna begränsningar.

Kapitel 8: SQL-databasprestanda

Programdesign för databasprestanda

Ändra inte ett SQLStatement-objekts `text`*-egenskap när det har verkställts. Använd i stället en SQLStatement-instans för varje SQL-sats och satsparametrar för att ange olika värden.*

Innan en SQL-sats körs förbereds (kompileras) den i körningen för att avgöra vilka steg som ska användas internt för att utföra satsen. När du anropar `SQLStatement.execute()` på en SQLStatement-instans som inte har körts tidigare förbereds satsen automatiskt innan den körs. För efterföljande anrop till metoden `execute()` förbereds satsen fortfarande, såvida inte egenskapen `SQLStatement.text` har ändrats. Därmed körs den snabbare.

För att dra störst nytta av att återanvända satser bör du använda satsparametrar för att anpassa din sats om värden ändras mellan satskörningar. (Satsparametrar anges med egenskapen `SQLStatement.parameters` (associativ array).) Om du ändrar värdena för satsparametrarna behöver inte satsen förberedas på nytt i körningen, till skillnad från när du ändrar SQLStatement-instansens `text`-egenskap.

När du återanvänder en SQLStatement-instans måste en referens till SQLStatement-instansen sparas i programmet när den har förberetts. Detta gör du genom att deklarera variabeln som en klassomfångsvariabel i stället för en funktionsomfångsvariabel. Ett bra sätt att göra SQLStatement till en klassomfångsvariabel är att strukturera programmet så att en SQL-sats placeras i en enskild klass. En grupp satser som körs i kombination kan också placeras i en enskild klass. (Tekniken kallas för ett Command-designmönster.) Om du definierar instanserna som medlemsvariabler i klassen finns de kvar så länge instansen av klassen wrapper finns i programmet. Du kan, som ett minimum, definiera en variabel som innehåller SQLStatement-instansen utanför en funktion så att instansen finns kvar i minnet. Du kan till exempel deklarera SQLStatement-instansen som en medlemsvariabel i en ActionScript-klass eller som en icke-funktionsvariabel i en JavaScript-fil. Du kan sedan ange satsens parametervärden och anropa dess `execute()`-metod när du vill köra frågan.

Använd databasindex för att förbättra körhastigheten när data jämförs och sorteras.

När du skapar ett index för en kolumn lagrar databasen en kopia av data i kolumnen. Kopian hålls sorterad i numerisk eller alfabetisk ordning. Det gör att databasen snabbt kan matcha värden (t.ex. när likhetsoperatorn används) och sortera resulterande data med `ORDER BY`-satsen.

Databasindex uppdateras kontinuerligt vilket gör att dataändringar (INSERT eller UPDATE) utförs en aning långsammare i tabellen. Hastigheten vid datahämtning kan dock öka avsevärt. På grund av denna prestandakompromiss bör du inte indexera varenda kolumn i alla tabeller. Använd i stället en strategi när du definierar index. Använd riktlinjerna nedan när du planerar en strategi för indexering:

- Indexera kolumner som används i kopplade tabeller efter WHERE-satser eller ORDER BY-satser
- Indexera kolumner som ofta används tillsammans som ett index
- Ange COLLATE NOCASE-kollation för ett index när en kolumn som innehåller textdata hämtas sorterad i alfabetisk ordning

Överväg att kompilera SQL-satser i förväg när programmet är inaktivt.

Den första gången som en SQL-sats verkställs, tar det längre tid eftersom SQL-texten förbereds (kompileras) av databasmotorn. Eftersom det kan vara krävande att förbereda och köra en sats är en strategi att läsa in initiala data i förväg och sedan köra övriga satser i bakgrunden:

1. Läs först in data som behövs i programmet.
2. Kör de övriga satserna när de initiala startåtgärderna för programmet har slutförts eller vid något inaktivt läge i programmet.

Anta t.ex. att programmet inte alls behöver åtkomst till databasen för att visa den inledande skärmen. I sådana fall kan du vänta tills skärmen visas innan databasanslutningen öppnas. Skapa till sist SQLStatement-instanser och verkställ alla som du kan.

Eller anta att vissa data visas direkt när programmet startar, till exempel resultatet av en viss fråga. I så fall är det bara att köra SQLStatement-instansen för den frågan. När initiala data har lästs in och visats kan du skapa SQLStatement-instanser för andra databasåtgärder och, om möjligt, köra andra satser som behövs senare.

I praktiken är den extra tiden som krävs att förbereda satsen en engångskostnad när SQLStatement-instanser återanvänds. Det har förmodligen inte någon större inverkan på den övergripande prestandan.

Gruppera flera SQL dataändringsåtgärder som en transaktion.

Anta att du kör många SQL-satser som innefattar tillägg eller ändringar av data (`INSERT`- eller `UPDATE`-satser). I så fall kan du öka prestanda markant genom att köra alla satser i en explicit transaktion. Om du inte börjar en transaktion explicit körs var och en av satserna i en egen transaktion som skapas automatiskt. När körningen av varje transaktion (varje sats) har slutförts skrivs alla resulterande data till databasfilen på disken.

Å andra sidan bör du även tänka på vad som händer om du skapar en transaktion explicit och kör satserna i kontexten för den transaktionen. Vid körningen görs alla ändringar i minnet och alla ändringar skrivs sedan på en gång till databasfilen när transaktionen har verkställts. Att skriva data till disk är vanligtvis den mest tidskrävande delen av åtgärden. Därför förbättras prestanda markant om skrivningen till disk endast sker en gång i stället för en gång per SQL-sats.

Bearbeta stora `SELECT`-frågeresultat i delar med SQLStatement-klassens `execute()`-metod (med parametern `prefetch`) och `next()`-metoden.

Anta att du verkställer en SQL-sats som hämtar en stor resultatuppsättning. Programmet bearbetar sedan varje datarad i en slinga. Data formateras eller objekt skapas till exempel. Att bearbeta dessa data kan ta lång tid vilket kan leda till återgivningsproblem, t.ex. att skärmen fryser eller inte svarar. Så som beskrivs i ”Asynkrona åtgärder” på sidan 75, är en lösning att dela upp arbetet i segment. SQL-databasens API gör det lätt att dela upp databearbetningen.

SQLStatement-klassens `execute()`-metod har en valfri `prefetch`-parameter (den första parametern). Om du tillhandahåller ett värde anger den det maximala antalet resultatrader som databasen returnerar när körningen är klar:

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

När den första uppsättningen resultatdata returneras kan du anropa `next()`-metoden för att fortsätta att verkställa satsen och hämta ytterligare en uppsättning resultatrader. Precis som `execute()`-metoden, accepterar `next()`-metoden en `prefetch`-parameter som anger maximalt antal rader som ska returneras:

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

Du kan fortsätta att anropa `next()`-metoden tills alla data har lästs in. Så som visades tidigare kan du avgöra när alla data har lästs in. Kontrollera egenskapen `complete` för SQLResult-objektet som skapas varje gång metoden `execute()` eller `next()` avslutas.

Obs! *Använd parametern `prefetch` och metoden `next()` när du ska dela upp bearbetning av resultatdata. Använd inte parametern och metoden när du ska begränsa resultatet av en fråga till en del av resultatuppsättningen. Om du bara vill hämta en delmängd av rader i resultatuppsättningen för en sats ska du använda `LIMIT`-satsen i satsen `SELECT`. Om resultatuppsättningen är stor kan du fortfarande använda `prefetch`-parametern och `next()`-metoden för att dela upp resultatbearbetningen.*


Överväg att använda flera asynkrona SQLConnection-objekt med en databas för att verkställa flera satser samtidigt.

När ett SQLConnection-objekt ansluts till en databas med metoden `openAsync()`, körs det i bakgrunden i stället för i den huvudsakliga körningstråden. Dessutom kör varje SQLConnection sin egen bakgrundstråd. Genom att använda flera SQLConnection-objekt kan du effektivt köra flera SQL-satser samtidigt.

Det finns vissa potentiella nackdelar med denna metod. Den viktigaste är att varje extra SQLStatement-objekt kräver mer minne. Samtidig körning kan även innebära mer arbete för processorn, speciellt på maskiner som bara har en CPU eller CPU-kärna. På grund av detta rekommenderas inte metoden för användning på mobila enheter.

Ytterligare en nackdel är att den potentiella fördelen med att återanvända SQLStatement-objekt kan förloras eftersom ett SQLStatement-objekt är länkat till ett enda SQLConnection-objekt. Det innebär att SQLStatement-objektet inte kan återanvändas om dess associerade SQLConnection-objekt redan används.

Om du väljer att använda flera SQLConnection-objekt anslutna till en databas bör du tänka på att varje objekt verkställer satser i sin egen transaktion. Kom ihåg att ta dessa separata transaktioner med i beräkningen i kod som ändrar data, t.ex. när data läggs till, ändras eller tas bort.

Paul Robertson har skapat ett bibliotek med öppen källkod som hjälper dig att dra nytta av fördelarna med att använda flera SQLConnection-objekt samtidigt som de potentiella nackdelarna minimeras. Biblioteket använder en pool av SQLConnection-objekt och hanterar associerade SQLStatement-objekt. Det ser till att SQLStatement-objekt återanvänds och flera SQLConnection-objekt är tillgängliga för att verkställa flera satser samtidigt. Gå till http://probertson.com/projects/air-sqlite/ om du vill ha mer information eller vill hämta biblioteket.

Databasfiloptimering

 *Undvik schemaändringar i databaser.*

Undvik, om möjligt, att göra ändringar i schemat (tabellstrukturen) för en databas när du har lagt till data i databasens tabeller. I en databasfil finns vanligtvis tabelldefinitionerna i början av filen. När du öppnar en anslutning till en databas läses dessa definitioner in. När du lägger till data i databastabeller läggs dessa data till i filen efter tabelldefinitionsdatan. Om du gör schemaändringar kommer dock tabellens nya definitionsdata att blandas med tabelldata i databasfilen. Om du t.ex. lägger till en kolumn i en tabell eller lägger till en ny tabell kan det leda till att datatyperna blandas. Om alla definitionsdata för tabellen inte är samlade i början av databasfilen tar det längre tid att öppna en anslutning till databasen. Det tar längre tid att öppna anslutningen eftersom det tar längre tid för körtidsmodulen att läsa tabellens definitionsdata från olika delar av filen.

 *Använd metoden `SQLConnection.compact()` för att optimera en databas efter schemaändringar.*

Om du måste göra schemaändringar kan du anropa metoden `SQLConnection.compact()` när ändringarna har slutförts. Med den här åtgärden omstruktureras databasfilen så att tabelldefinitionsdata samlas i början av filen. Åtgärden `compact()` kan dock vara tidskrävande, speciellt när databasfilen har blivit lite större.

Onödig databasbearbetning i körtid

 *Använd ett helt kvalificerat tabellnamn (inklusive databasnamn) i SQL-satsen.*

Specificera alltid databasnamnet explicit tillsammans med varje tabellnamn i en sats. (Använd ”main” om det är huvuddatabasen). Följande kod inkluderar t.ex. ett explicit databasnamn, `main`:

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

Genom att ange databasnamnet explicit förhindrar du att körningen måste kontrollera varje ansluten databas för att hitta den matchande tabellen. Det förhindrar även möjligheten att fel databas väljs i körningen. Bakom scenerna är SQLConnection-instansen även ansluten till en temporär databas som är tillgänglig via SQL-satser. Därför bör du följa den här regeln även om en SQLConnection-instans bara är ansluten till en enskild databas.

 *Använd explicita kolumnnamn SQL `INSERT`- och `SELECT`-satser.*

Exemplen nedan visar hur du använder explicita kolumnnamn:

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

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

Jämför de föregående exemplen med de som följer. Undvik den här typen av kod:

```
-- 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
```

Utan explicita kolumnnamn måste körtidsmodulen utföra extra arbete för att ta reda på kolumnnamnen. Om en `SELECT`-sats använder ett jokertecken i stället för explicita kolumner kommer körtidsmodulen att hämta extra data. Dessa extra data kräver ytterligare bearbetning och skapar objektinstanser som inte behövs.

Undvik att koppla samma tabell flera gånger i en sats om du inte jämför tabellen med sig självt.

När SQL-satser blir stora, kan du av misstag koppla en databastabell till en fråga flera gånger. Ofta kan samma resultat uppnås om tabellen bara används en gång. Det är mer troligt att samma tabell kopplas flera gånger om du använder en eller flera vyer i frågan. Du kanske t.ex. kopplar en tabell till en fråga samt en vy som inkluderar data från samma tabell. De två åtgärderna leder till mer än en koppling.

Effektiv SQL-syntax

Använd `JOIN` (i `FROM`-satsen) när du ska inkludera en tabell i en fråga i stället för en underfråga i `WHERE`-satsen. Detta fungerar även om du bara behöver data i en tabell för filtrering, inte för resultatet.

Att koppla flera tabeller i `FROM`-satsen fungerar bättre än att använda en underfråga i en `WHERE`-sats.

Undvik SQL-satser som inte kan dra nytta av index. Sådana satser inkluderar de som använder sammanställningsfunktioner i en underfråga, en `UNION`-sats i en underfråga och en `ORDER BY`-sats med en `UNION`-sats.

Ett index kan avsevärt öka bearbetningshastigheten för en `SELECT`-fråga. En del SQL-syntax hindrar dock databasen från att använda index och tvingar den att använda verkliga data för söknings- och sorteringsåtgärder.

Undvik att använda operatören `LIKE`, speciellt med ett inledande jokertecken, t.ex. `LIKE('%XXXX%')`.

Eftersom `LIKE`-åtgärden har stöd för sökningar med jokertecken tar det längre tid än exakta jämförelser. Om du inleder söksträngen med jokertecken kan databasen inte använda index alls vid sökningen. I stället måste databasen söka igenom den fullständiga texten på varje rad i tabellen.

Undvik att använda operatören `IN`. Om de möjliga värdena är kända på förhand, kan `IN`-åtgärden skrivas med `AND` eller `OR` för snabbare körning.

Den andra av följande två satser verkställs snabbare. Den är snabbare eftersom den använder enkla likhetsuttryck i kombination med `OR` i stället för `IN()`- eller `NOT IN()`-satser:

```
-- 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
```


Överväg alternativa former av en SQL-sats för att förbättra prestandan.

Så som visades i de föregående exemplen kan databasprestandan påverkas av hur en SQL-sats är skriven. Det finns ofta flera olika sätt att skriva en `SELECT`-sats för att hämta en viss resultatmängd. I vissa fall körs en metod avsevärt snabbare än en annan. I tillägg till de föregående förslagen kan du lära dig mer om olika SQL-satser och deras prestanda via särskilda resurser om SQL-språket.

SQL-satsens prestanda


Jämför alternativa SQL-satser för att avgöra vilken som är snabbast.

Det bästa sättet att jämföra prestandan för flera versioner av en SQL-sats är att testa dem direkt med databasen och dina data.

Följande utvecklingsverktyg anger körtider när SQL-satser körs. Använd dem när du jämför hastigheten för alternativa versioner av satser:

- Run! (redigerings- och testningsverktyg för AIR SQL-frågor av Paul Robertson)
- Lita (SQLite Administration Tool av David Deraedt)

Kapitel 9: Testning och distribution

Testning

Det finns ett antal verktyg som kan användas för att testa programmen. Du kan använda klasserna Stats och PerformanceTest, som utvecklats av Flash-användare. Du kan även använda profileraren i Adobe® Flash® Builder™ och FlexPMD-verktyget.

Klassen Stats

Om du vill profilera koden under körningen med den officiella versionen av körningsmiljön, utan något externt verktyg, kan du använda klassen Stats, som utvecklats av en Flash-användare med namnet "mr. doob". Du kan hämta klassen Stats på följande adress: https://github.com/mrdoob/Hi-ReS-Stats.

Klassen Stats använder du för att spåra följande:

- Bildruteåtergivning per sekund (ju högre värde desto bättre).
- Antal millisekunder som används för att återge en bildruta (ju lägre värde desto bättre).
- Hur mycket av minnet som används för koden. Om det ökas för varje bildruta är det möjligt att programmet har ett minnesläckage. Det är viktigt att undersöka orsaken till minnesläckaget.
- Hur mycket av minnet som programmet utnyttjat maximalt.

När du laddat ned klassen Stats kan den används i följande kompakta kod:

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

Du kan aktivera Stats-objektet genom att använda villkorlig kompilering i Adobe® Flash® Professional eller Flash Builder:

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

Genom att ändra värdet för `DEBUG`-konstanten kan du aktivera eller inaktivera komplieringen av Stats-objektet. Samma arbetssätt kan användas för att ersätta all kodlogik som du inte vill kompilera i programmet.

Klassen PerformanceTest

För dig som vill profilera ActionScript-kodkörningen har Grant Skinner utvecklat ett verktyg som kan integreras i ett testarbetsflöde. Du kan skicka en anpassad klass till PerformanceTest-klassen där ett antal tester av koden sedan utförs. Klassen PerformanceTest gör att du enkelt kan testa olika arbetssätt. Du kan hämta klassen PerformanceTest på följande plats: http://www.gskinner.com/blog/archives/2009/04/as3_performance.html.

Flash Builder-profileraren

Flash Builder levereras med en profilerare som du kan använda för att testa din kod med stor detaljrikedom.

Obs! *Använd felsökningsversionen av Flash Player för att använda profileraren eftersom du annars kommer att få ett felmeddelande.*

Profileraren kan även användas med innehåll som producerats i Adobe Flash Professional. Du gör detta genom att ladda den kompilerade SWF-filen från ett ActionScript- eller Flex-projekt till Flash Builder för att sedan köra profileraren. Mer information om profileraren finns i ”Profiling Flex applications” i Using Flash Builder 4.

FlexPMD

Adobe Technical Services har utvecklat ett verktyg med namnet FlexPMD, vilket du använder för att granska ActionScript 3.0-kodens kvalitet. FlexPMD är ett ActionScript-verktyg som påminner om JavaPMD. Med FlexPMD kan du förbättra kodkvaliteten genom att granska en ActionScript 3.0- eller Flex-källkatalog. Här upptäcks dåliga kodmönster som exempelvis oanvänd kod, alltför komplex kod, alltför lång kod och felaktig användning av livscykeln för Flex-komponenten.

FlexPMD är ett Adobe-projekt med öppen källkod som finns på följande adress: http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD. Eclipse är ett plugin-program, som finns på följande adress: http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD+Eclipse+plugin.

FlexPMD gör det enklare att granska och kontrollera att koden är ren och optimerad. Den stora fördelen med FlexPMD är dess omfattning. Du kan som utvecklare skapa en egen uppsättning med regler för att granska all kod. Du kan till exempel skapa regler som upptäcker överanvändning av filter eller regler för att upptäcka dåligt skriven kod.

Distribution

När du exporterar den sista version av programmet i Flash Builder är det viktigt att kontrollera att du exporterar den version som ska släppas. När du exporterar den versionen tas all felsökningsinformation i SWF-filen bort. När du tar bort felsökningsinformationen blir SWF-filen mycket mindre vilket gör att programmet körs snabbare.

Om du vill köra den version av projektet som ska släppas ska du använda projektpanelen i Flash Builder och välja alternativet Export Release Build (Exportera slutversion).

Obs! *När du kompilerar projektet i Flash Professional kan du inte välja mellan versionen som ska släppas och felsökningsversionen. Den kompilerade SWF-filen är som standard en släppt version.*