Ottimizzazione delle prestazioni per la PIATTAFORMA ADOBE® FLASH®

Note legali

Sommario

Capitolo 1: Introduzione

Le applicazioni Adobe® AIR® e Adobe® Flash® Player sono eseguibili su molte piattaforme, compresi desktop, dispositivi mobili, tavolette e apparecchi TV. Mediante esempi di codice e casi di utilizzo, questo documento delinea le procedure ottimali che gli sviluppatori dovrebbero seguire per la distribuzione di queste applicazioni. Gli argomenti trattati sono:

- Risparmio di memoria
- Riduzione dell'uso della CPU
- Miglioramento delle prestazioni di ActionScript 3.0
- Aumento della velocità di rendering
- Ottimizzazione dell'interazione di rete
- Operazioni con audio e video
- Ottimizzazione delle prestazioni del database SQL
- Benchmarking e distribuzione delle applicazioni

La maggior parte di queste ottimizzazioni è utilizzabile nelle applicazioni su tutti i dispositivi, sia per il runtime AIR che per il runtime Flash Player. Vengono discusse anche aggiunte ed eccezioni per dispositivi particolari.

Alcune di queste ottimizzazioni riguardano le funzionalità introdotte in Flash Player 10.1 e AIR 2.5, tuttavia molte sono applicabili anche alle versioni precedenti di AIR e Flash Player.

Elementi fondamentali dell'esecuzione del codice runtime

Per comprendere come è possibile migliorare le prestazioni di un'applicazione, è fondamentale conoscere i meccanismi di esecuzione del codice runtime nella piattaforma Flash. Il runtime segue un andamento ciclico, con determinate azioni che vengono eseguite su ogni "fotogramma". In questo caso, per fotogramma si intende semplicemente un blocco di tempo determinato dalla frequenza dei fotogrammi specificata per l'applicazione. La quantità di tempo assegnata a ciascun fotogramma corrisponde direttamente alla frequenza dei fotogrammi. Ad esempio, se specificate una frequenza fotogrammi di 30 fotogrammi al secondo, il runtime tenta di assegnare a ciascun fotogramma un trentesimo di secondo.

La frequenza fotogrammi iniziale di un'applicazione viene specificata in fase di creazione (authoring). Potete specificare la frequenza fotogrammi mediante le impostazioni di Adobe® Flash® Builder™ o di Flash Professional. Oppure, potete anche specificare la frequenza fotogrammi iniziale nel codice. Per impostare la frequenza fotogrammi in un'applicazione basata esclusivamente su ActionScript, applicate il tag di metadati `[SWF(frameRate="24")]` alla classe documento principale. In MXML, impostate l'attributo `frameRate` nel tag Application o WindowedApplication.

Ogni ciclo (loop) di fotogramma consiste di due fasi, divise in tre parti: gli eventi, l'evento `enterFrame` e il rendering.

La prima fase comprende due parti (gli eventi e l'evento `enterFrame`), che possono entrambe determinare una chiamata al codice. Nella prima parte della prima fase, arrivano e vengono inviati gli eventi runtime. Questi eventi possono rappresentare il completamento o l'avanzamento di operazioni asincrone, ad esempio una risposta dal caricamento di dati su una rete. Includono inoltre gli eventi generati dall'input dell'utente. Quando gli eventi vengono inviati, il runtime esegue il codice dell'applicazione nei listener che avete registrato. Se non si verifica alcun evento, il runtime completa questa fase senza eseguire alcuna azione. Il runtime non accelera mai la frequenza fotogrammi a causa di una mancanza di attività. Se gli eventi si verificano durante altre parti del ciclo di esecuzione, il runtime li inserisce in coda e li invia nel fotogramma successivo.

La seconda parte della prima fase è costituita dall'evento `enterFrame`. Questo evento si distingue da tutti gli altri perché viene inviato una volta per ogni fotogramma.

Una volta che tutti gli eventi sono stati inviati, ha inizio la fase di rendering del ciclo di fotogramma. A questo punto il runtime calcola lo stato di tutti gli elementi visibili e li disegna sullo schermo. Il processo viene quindi ripetuto come in una corsa su pista.

Nota: *per gli eventi che includono una proprietà* `updateAfterEvent`*, potete impostare forzatamente l'esecuzione immediata del rendering anziché attendere la fase di rendering. Evitate tuttavia di utilizzare* `updateAfterEvent` *se è spesso causa di problemi di prestazioni.*

Per semplificare, possiamo immaginare che le due fasi del ciclo di fotogramma abbiano la stessa durata. In questo caso, la prima metà del ciclo di fotogramma sarebbe occupata dall'esecuzione dei gestori di eventi e del codice dell'applicazione e l'altra metà verrebbe utilizzata per il rendering. Nella realtà, tuttavia, le due fasi hanno spesso una durata diversa. A volte il codice dell'applicazione utilizza più della metà del tempo disponibile nel fotogramma, riducendo il tempo a disposizione per il rendering. In altri casi, soprattutto in presenza di contenuti visivi complessi come filtri e metodi di fusione, il rendering occupa più della metà del tempo del fotogramma. Poiché il tempo effettivamente occupato dalle due fasi è flessibile, il ciclo di fotogramma viene detto anche “pista elastica”.

Se le due operazioni del ciclo di fotogramma (esecuzione del codice e rendering) richiedono troppo tempo, il runtime non è in grado di rispettare la frequenza fotogrammi. Il fotogramma si espande, occupando un intervallo di tempo superiore a quello assegnato, e determina un ritardo nell'attivazione del fotogramma successivo. Se, ad esempio un ciclo di fotogramma impiega più di un trentesimo di secondo, il runtime non è in grado di aggiornare lo schermo a 30 fotogrammi al secondo. La riduzione della frequenza fotogrammi influisce negativamente sulle prestazioni. Nella migliore delle ipotesi, le animazioni vengono riprodotte in maniera discontinua. Nel peggiore dei casi, l'applicazione si blocca e il contenuto della finestra scompare.

Per ulteriori informazioni sul modello di rendering e sull'esecuzione del codice runtime nella piattaforma Flash, vedete le seguenti risorse:

- Articolo di Ted Patrick: Flash Player Mental Model - The Elastic Racetrack (Modello mentale di Flash Player - La pista elastica)
- Articolo di Trevor McCauley: Asynchronous ActionScript Execution (Esecuzione asincrona del codice ActionScript)
- Per informazioni sull'ottimizzazione di Adobe AIR per esecuzione di codice, memoria e rendering, vedete http://www.adobe.com/go/learn_fp_air_perf_tv_it (Video della presentazione alla MAX Conference a cura di Sean Christmann)

Differenza tra prestazioni percepite e prestazioni effettive

In ultima analisi, il giudizio sulle prestazioni di un'applicazione spetta agli utenti. Gli sviluppatori possono misurare le prestazioni in termini di tempo necessario per l'esecuzione delle operazioni o di numero di istanze di oggetti che vengono create. Queste metriche, tuttavia, sono poco significative per gli utenti finali. A volte gli utenti misurano le prestazioni in base a criteri diversi. Ad esempio, l'applicazione funziona velocemente e senza problemi e risponde rapidamente all'input? Ha un impatto negativo sulle prestazioni del sistema? Per valutare le prestazioni percepite di un'applicazione, provate a porvi le seguenti domande:

- Le animazioni vengono riprodotte in maniera fluida o discontinua?
- I contenuti video vengono riprodotti in maniera fluida o discontinua?
- I clip audio vengono riprodotti senza interruzioni o in maniera frammentaria?
- La finestra è soggetta a problemi di sfarfallio o scomparsa del contenuto durante le operazioni più lunghe?
- Il testo digitato viene visualizzato in tempo reale o con ritardo?
- I clic del mouse producono un effetto immediato o si verifica un ritardo?
- La ventola della CPU produce un rumore più forte durante l'esecuzione dell'applicazione?
- Su un laptop o un dispositivo mobile, la batteria si scarica rapidamente quando viene eseguita l'applicazione?
- Le prestazioni delle altre applicazioni peggiorano durante l'esecuzione dell'applicazione?

La differenza tra prestazioni percepite ed effettive è importante. Il modo di ottenere prestazioni percepite ottimali non corrisponde sempre a quello che consente di raggiungere le prestazioni più rapide in assoluto. Verificate che l'applicazione non esegua mai una quantità di codice tale da impedire al runtime di aggiornare lo schermo e acquisire l'input dell'utente con la frequenza prevista. In alcuni casi, per raggiungere un equilibrio ottimale dovrete suddividere un'attività del programma in più parti in modo che il runtime aggiorni lo schermo tra una parte e l'altra. Per istruzioni specifiche, vedete “Prestazioni di rendering” a pagina 49.

Le tecniche e i suggerimenti riportati di seguito consentono di migliorare sia le prestazioni percepite dagli utenti che le prestazioni effettive a livello di esecuzione del codice.

Ottimizzazioni mirate

Alcuni miglioramenti delle prestazioni non si riflettono in un miglioramento evidente per gli utenti. È importante concentrare gli interventi per l'ottimizzazione delle prestazioni su aree problematiche di un'applicazione specifica. Alcune ottimizzazioni delle prestazioni sono procedure generalmente consigliate e possono essere sempre applicate. Per altre ottimizzazioni, l'effettiva utilità dipende dalle esigenze dell'applicazione e dalla base di utenti prevista. Le applicazioni forniscono sempre prestazioni migliori, ad esempio, se non utilizzate animazioni, video o filtri grafici ed effetti. D'altra parte, la piattaforma Flash viene utilizzata proprio perché offre capacità grafiche e multimediali che consentono di creare applicazioni sofisticate e incisive. Valutate se il livello di complessità desiderato è adeguato alle caratteristiche prestazionali dei computer e dei dispositivi su cui viene eseguita l'applicazione.

A livello generale, evitate di eseguire le ottimizzazioni in una fase troppo iniziale. In alcuni casi, per ottimizzare le prestazioni dovrete modificare il codice in una maniera che lo rende più difficile da leggere o meno flessibile. Una volta ottimizzato, questo codice è più difficile da gestire. Per queste ottimizzazioni è opportuno verificare se una determinata sezione del codice presenta prestazioni scadenti prima di decidere di ottimizzarla.

Per migliorare le prestazioni dovrete accettare alcuni compromessi. Idealmente, la riduzione della quantità di memoria consumata da un'applicazione comporta anche l'aumento della velocità con cui l'applicazione esegue una specifica operazione. In una situazione reale, tuttavia, questo tipo di miglioramento non è sempre possibile. Se un'applicazione si blocca durante un'operazione, ad esempio, spesso è necessario suddividere l'esecuzione dell'attività tra più fotogrammi. Poiché il lavoro viene suddiviso, è probabile che il completamento del processo richieda complessivamente più tempo. È tuttavia possibile che l'utente non noti questo tempo aggiuntivo se l'applicazione continua a rispondere all'input e non si blocca.

Per individuare gli elementi da ottimizzare, e stabilire se le ottimizzazioni sono effettivamente utili, è importante eseguire dei test delle prestazioni. In "Benchmarking e distribuzione" a pagina 97 vengono riportati tecniche e suggerimenti utili per testare le prestazioni.

Per ulteriori informazioni su come individuare le aree di un'applicazione da ottimizzare, vedete le seguenti risorse:

- Per informazioni sulle applicazioni per AIR ad alte prestazioni, vedete http://www.adobe.com/go/learn_fp_goldman_tv_it (Video della presentazione alla MAX Conference a cura di Oliver Goldman)
- Per informazioni sulle applicazioni Adobe AIR ad alte prestazioni, vedete http://www.adobe.com/go/learn_fp_air_perf_devnet_it (Articolo su Adobe Developer Connection basato sulla presentazione a cura di Oliver Goldman)

Capitolo 2: Risparmio di memoria

Risparmiare memoria è sempre importante quando si sviluppano applicazioni, anche nel caso di applicazioni desktop. Se poi si sviluppa per i dispositivi mobili, la memoria è un bene ancora più prezioso ed è quindi opportuno limitare la quantità di memoria che l'applicazione consuma.

Oggetti di visualizzazione

Scegliete un oggetto di visualizzazione appropriato.

ActionScript 3.0 include un'ampia gamma di oggetti di visualizzazione. Uno dei consigli di ottimizzazione più semplici al fine di limitare l'utilizzo della memoria è quello di usare il tipo appropriato di oggetti di visualizzazione. Per forme semplici e non interattive, usate oggetti Shape. Per gli oggetti interattivi che non necessitano di una linea temporale, scegliete oggetti Sprite. Per animazioni con una linea temporale, usate oggetti MovieClip. Scegliete sempre il tipo di oggetto più efficiente per l'applicazione specifica che state sviluppando.

Il codice seguente mostra l'utilizzo della memoria associato a vari oggetti di visualizzazione:

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

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

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

Il metodo `getSize()` indica quanti byte di memoria vengono consumati da un oggetto. Potete vedere che utilizzando più oggetti MovieClip anziché semplici oggetti Shape, si può sprecare memoria se le funzionalità di un oggetto MovieClip non sono realmente necessarie.

Tipi di base

Utilizzate il metodo `getSize()` per valutare il codice e determinare l'oggetto più efficiente per un'operazione specifica.

Tutti i tipi di base eccetto String usano 4 – 8 byte di memoria. Non esiste un modo per ottimizzare la memoria utilizzando un tipo specifico per un tipo di base:

```
// 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
```

A un oggetto Number, che rappresenta un valore a 64 bit, vengono assegnati 8 byte dall'AVM (ActionScript Virtual Machine), se non viene assegnato un valore specifico. Tutti gli altri tipi di base vengono memorizzati in 4 byte.

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

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

Il comportamento è diverso per il tipo String. La quantità di memoria allocata dipende dalla lunghezza della stringa:

```
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
```

Utilizzate il metodo `getSize()` per valutare il codice e determinare l'oggetto più efficiente per un'operazione specifica.

Riutilizzo degli oggetti

Riutilizzate gli oggetti quando è possibile, anziché ricrearli.

Un altro modo semplice per ottimizzare la memoria consiste nel riutilizzare gli oggetti ed evitare di ricrearli, quando è possibile. Ad esempio, in un ciclo, non utilizzate il codice seguente:

```
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);
}
```

Ricreando l'oggetto Rectangle in ogni iterazione di ciclo si utilizza più memoria e i tempi si allungano perché a ogni iterazione viene creato un nuovo oggetto. Adottate l'approccio seguente:

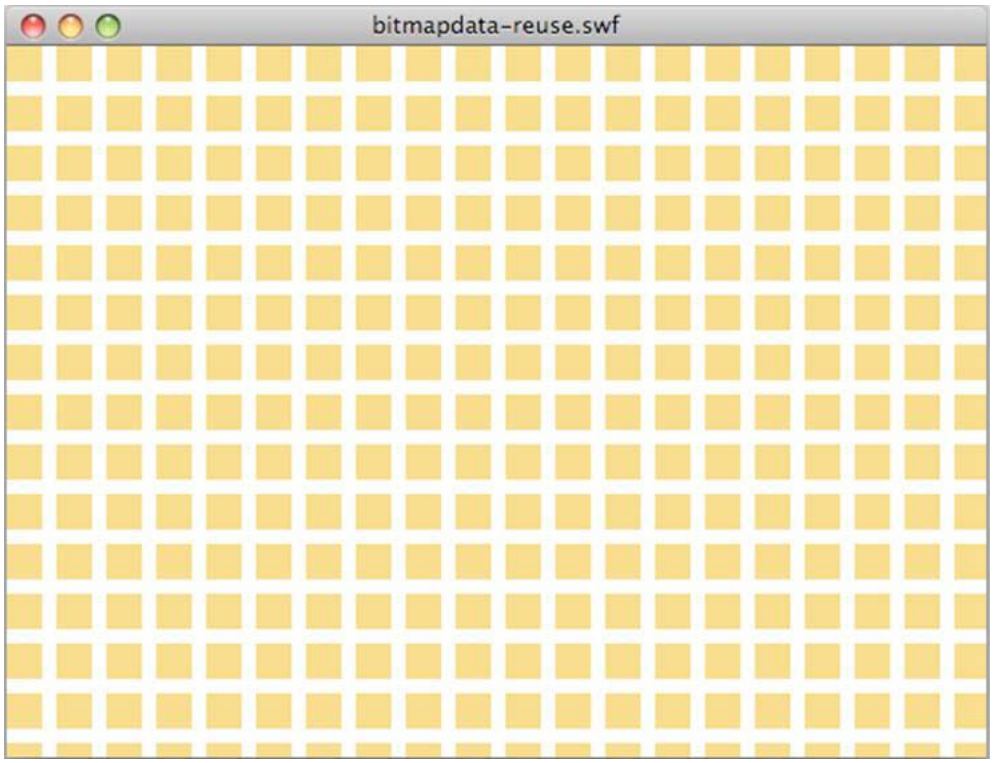
```
const MAX_NUM:int = 18;
const COLOR:uint = 0xCCCCCC;

// 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);
}
```

Nell'esempio precedente è stato utilizzato un oggetto che ha un impatto relativamente modesto sulla memoria. L'esempio seguente invece illustra un caso di risparmio di memoria più consistente grazie al riutilizzo di un oggetto BitmapData. Il codice seguente usato per creare un effetto "a piastrelle" determina uno spreco di memoria:

```
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);
}
```

Nota: *quando si usano valori positivi, l'inserimento del valore arrotondato in int è molto più veloce rispetto all'uso del metodo* `Math.floor()`.

L'immagine seguente mostra il risultato dell'effetto a piastrelle sulla bitmap:



Risultato dell'effetto a piastrelle su una bitmap

Una versione ottimizzata crea un'istanza BitmapData singola alla quale fanno riferimento più istanze Bitmap, producendo lo stesso risultato:

```
// 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);
}
```

Questa soluzione permette di risparmiare circa 700 KB di memoria, ovvero una quantità significativa per un dispositivo mobile tradizionale. Ogni contenitore bitmap può essere modificato senza alterare l'istanza BitmapData originale utilizzando le proprietà Bitmap:

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

L'immagine seguente mostra il risultato delle trasformazioni bitmap:

Risultato delle trasformazioni bitmap

Altri argomenti presenti nell'Aiuto

"Caching bitmap" a pagina 56

Pooling di oggetti

Usate il pooling degli oggetti quando potete.

Un'altra importante tecnica di ottimizzazione è il "pooling degli oggetti" e prevede il riutilizzo degli oggetti nel tempo. Durante l'inizializzazione della vostra applicazione, dovete creare un numero definito di oggetti e memorizzarli in un "pool", ad esempio un oggetto Array o Vector. Quando avete finito di utilizzare un oggetto, lo disattivate in modo che non consumi risorse della CPU e rimuovete tutti i riferimenti reciproci. Tuttavia, non impostate i riferimenti su `null`, per non rendere idoneo l'oggetto al processo di garbage collection. Limitatevi a reinserire l'oggetto nel pool, in modo da poterlo recuperare quando avete bisogno di un nuovo oggetto.

Riutilizzando gli oggetti è possibile ridurre la necessità di crearne nuove istanze, attività che può rivelarsi onerosa. Inoltre, si riducono le probabilità di esecuzione del processo di garbage collection, che può rallentare l'applicazione. Il codice riportato di seguito illustra la tecnica di pooling degli oggetti:

```
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;
        }
    }
}
```

La classe SpritePool crea un pool di nuovi oggetti durante l'inizializzazione dell'applicazione. Il metodo `getSprite()` restituisce le istanze di questi oggetti e il metodo `disposeSprite()` le rilascia. Il codice consente al pool di crescere dopo che è stato consumato completamente. È anche possibile creare un pool di dimensioni fisse nel quale non vengano allocati nuovi oggetti quando il pool è esaurito. Cercate di evitare di creare nuovi oggetti, quando potete. Per ulteriori informazioni, vedete "Liberare spazio in memoria" a pagina 12. Il codice seguente utilizza la classe SpritePool per recuperare nuove istanze:

```
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 );
    }
}
```

Il codice seguente rimuove tutti gli oggetti di visualizzazione dall'elenco di visualizzazione quando viene fatto clic sul mouse e li riutilizza in seguito per un'altra operazione:

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

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

Nota: *il vettore del pool fa sempre riferimento agli oggetti Sprite. Se volete rimuovere completamente l'oggetto dalla memoria, dovete utilizzare il metodo* `dispose()` *sulla classe SpritePool per eliminare tutti i riferimenti rimanenti.*

Liberare spazio in memoria

Eliminate tutti i riferimenti agli oggetti per assicurarvi che venga attivato il processo di garbage collection.

Non potete avviare il garbage collector direttamente nella versione release di Flash Player. Per assicurarvi che un oggetto venga incluso nel processo di garbage collection, rimuovete tutti i riferimenti a tale oggetto. Tenete presente che il vecchio operatore `delete` utilizzato in ActionScript 1.0 e 2.0 si comporta diversamente in ActionScript 3.0. Può infatti essere utilizzato solo per eliminare le proprietà dinamiche di un oggetto dinamico.

Nota: *potete avviare il garbage collector direttamente in Adobe® AIR® e nella versione debug di Flash Player.*

Ad esempio, il codice seguente imposta un riferimento Sprite su `null`:

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

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

Ricordate che quando un oggetto è impostato su `null`, esso non viene necessariamente rimosso dalla memoria. A volte il garbage collector non viene eseguito, se la memoria disponibile non viene considerata sufficientemente ridotta. Il processo di garbage collection non è prevedibile. È l'allocazione della memoria, e non l'eliminazione dell'oggetto, ad attivare il processo di garbage collection. Quando viene eseguito, il garbage collector trova i grafici degli oggetti che non sono ancora stati raccolti e rileva gli oggetti inattivi nei grafici individuando gli oggetti che fanno riferimento gli uni agli altri ma non sono più utilizzati dall'applicazione. Gli oggetti inattivi rilevati in questo modo vengono eliminati.

Nelle applicazioni di grandi dimensioni, questo processo può consumare molte risorse della CPU e condizionare le prestazioni, producendo un rallentamento sensibile dell'applicazione. Cercate di limitare i passaggi del processo di garbage collection riutilizzando il più possibile gli oggetti. Inoltre, impostate i riferimenti su null quando potete, in modo che il garbage collector necessiti di meno tempo di elaborazione per individuarli. Gestite sempre direttamente la vita operativa degli oggetti, quando è possibile, e considerate il processo di garbage collection come una specie di polizza assicurativa.

Nota: *l'impostazione di un riferimento a un oggetto di visualizzazione su null non garantisce che l'oggetto venga bloccato. L'oggetto continua a consumare cicli di CPU fino a quando non viene incluso nel processo di garbage collection. Accertatevi di disattivare correttamente l'oggetto prima di impostare il suo riferimento su* `null`.

Il garbage collector può essere avviato mediante il metodo `System.gc()`, disponibile in Adobe AIR e nella versione di debug di Flash Player. Il profiler fornito con Adobe® Flash® Builder™ permette di avviare manualmente il garbage collector. L'esecuzione del garbage collector vi consente di vedere come la vostra applicazione risponde e se gli oggetti vengono eliminati correttamente dalla memoria.

Nota: *se un oggetto è stato utilizzato come listener di eventi, un altro oggetto può farvi riferimento. In tal caso, rimuovete i listener di eventi chiamando il metodo* `removeEventListener()` *prima di impostare i riferimenti su* `null`.

Fortunatamente, la quantità di memoria utilizzata dalle bitmap può essere ridotta istantaneamente. Ad esempio, la classe BitmapData include il metodo `dispose()`. Nell'esempio seguente viene creata un'istanza BitmapData di 1,8 MB. La memoria corrente in uso cresce fino a 1,8 MB e la proprietà `System.totalMemory` restituisce un valore inferiore:

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

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

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

Quindi, la classe BitmapData viene rimossa manualmente dalla memoria e la memoria in uso viene nuovamente verificata:

```
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
```

Anche se il metodo `dispose()` rimuove i pixel dalla memoria, il riferimento deve comunque essere impostato su `null` per rilasciarlo completamente. Chiamate sempre il metodo `dispose()` e impostate il riferimento su `null` quando non avete più bisogno di un oggetto BitmapData, in modo da liberare immediatamente la memoria.

Nota: *Flash Player 10.1 e AIR 1.5.2 introducono un nuovo metodo denominato `disposeXML()` nella classe System. Questo metodo vi consente di rendere un oggetto XML immediatamente disponibile per il processo di garbage collection, passando la struttura XML come parametro.*

Altri argomenti presenti nell'Aiuto

"Blocco e sblocco degli oggetti" a pagina 28

Uso delle bitmap

Utilizzare i vettori al posto delle bitmap è un buon modo per risparmiare memoria, tuttavia l'uso dei vettori, specialmente se in numero elevato, aumenta in modo sensibile il consumo di risorse della CPU o GPU. Le bitmap consentono di ottimizzare il rendering, poiché il runtime richiede meno risorse di elaborazione per disegnare i pixel sullo schermo di quante non ne servano per eseguire il rendering del contenuto vettoriale.

Altri argomenti presenti nell'Aiuto

"Caching manuale delle bitmap" a pagina 63

Downsampling delle bitmap

Per ottenere un uso più efficiente della memoria, le immagini a 32 bit opache vengono ridotte a 16 bit quando Flash Player rileva la presenza di uno schermo a 16 bit. Questa operazione di downsampling consente di dimezzare il consumo di risorse della memoria e di ottenere un rendering più veloce delle immagini. Questa funzione è disponibile solo in Flash Player 10.1 per Windows Mobile.

Nota: *prima di Flash Player 10.1, tutti i pixel creati nella memoria venivano memorizzati a 32 bit (4 byte). Un logo semplice di 300 x 300 pixel consumava 350 KB di memoria (300*300*4/1024). Grazie a questo nuovo comportamento, lo stesso logo opaco ora consuma solo 175 KB. Se il logo è trasparente, il downsampling a 16 bit non viene eseguito e le dimensioni in memoria rimangono le stesse. Questo comportamento riguarda solo le bitmap incorporate e le immagini caricate in runtime (PNG, GIF, JPG).*

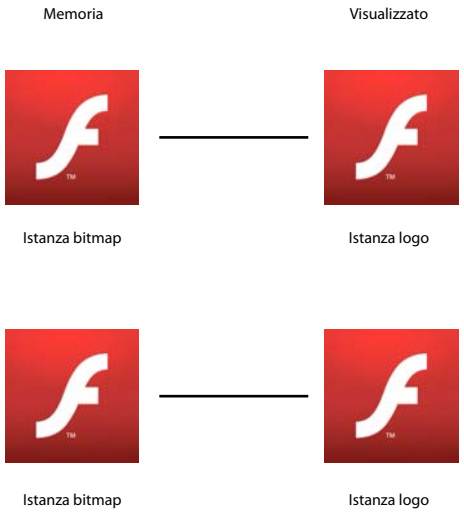
Sui dispositivi mobili può essere difficile notare la differenza tra un'immagine a 16 bit e la stessa immagine riprodotta a 32 bit. Nel caso di un'immagine semplice con pochi colori, la differenza è praticamente impercettibile. Anche per un'immagine più complessa, risulta comunque difficile rilevare le differenze. Tuttavia, si può verificare una certa degradazione del colore quando si effettua uno zoom in avvicinamento sull'immagine, e un gradiente a 16 bit può apparire meno fluido della versione a 32 bit.

Riferimento singolo BitmapData

È importante ottimizzare l'uso della classe BitmapData riutilizzando le istanze il più possibile. Flash Player 10.1 e AIR 2.5 introducono una nuova funzionalità per tutte le piattaforme, denominata "riferimento singolo BitmapData". Quando si creano istanze BitmapData da un'immagine incorporata, viene utilizzata un'unica versione della bitmap per tutte le istanze BitmapData. Solo se una bitmap viene modificata in un secondo momento, viene assegnato ad essa un riferimento bitmap specifico nella memoria. L'immagine incorporata può provenire dalla libreria o da un tag [Embed].

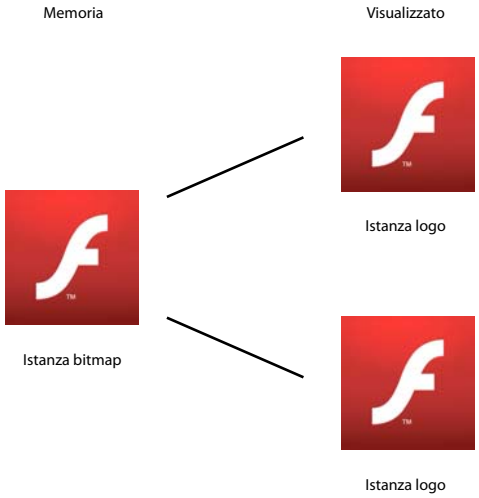
Nota: *anche i contenuti esistenti possono sfruttare questa nuova funzionalità, poiché Flash Player 10.1 e AIR 2.5 riutilizzano automaticamente le bitmap.*

Quando si crea un'istanza di un'immagine incorporata, viene creata in memoria una bitmap associata. Prima di Flash Player 10.1 e AIR 2.5, a ogni istanza veniva assegnata una bitmap separata in memoria, come mostra il diagramma seguente:



Bitmap in memoria prima di Flash Player 10.1 e AIR 2.5

In Flash Player 10.1 e AIR 2.5, quando si creano più istanze della stessa immagine, viene utilizzata un'unica versione della bitmap per tutte le istanze BitmapData. Il diagramma seguente illustra questo concetto:

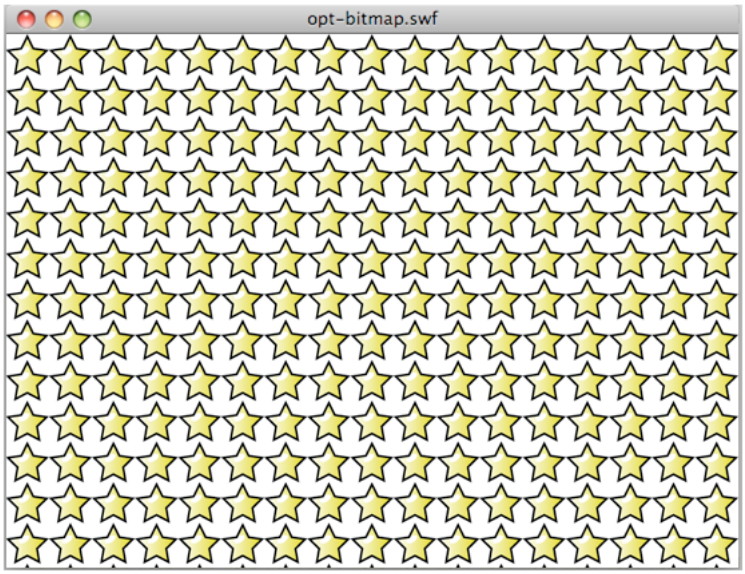


Bitmap in memoria in Flash Player 10.1 e in AIR 2.5

Questo approccio riduce sensibilmente la quantità di memoria utilizzata da un'applicazione che deve gestire molte bitmap. Il codice seguente crea più istanze di un simbolo `Star`:

```
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)
    }
}
```

L'immagine seguente mostra il risultato del codice:



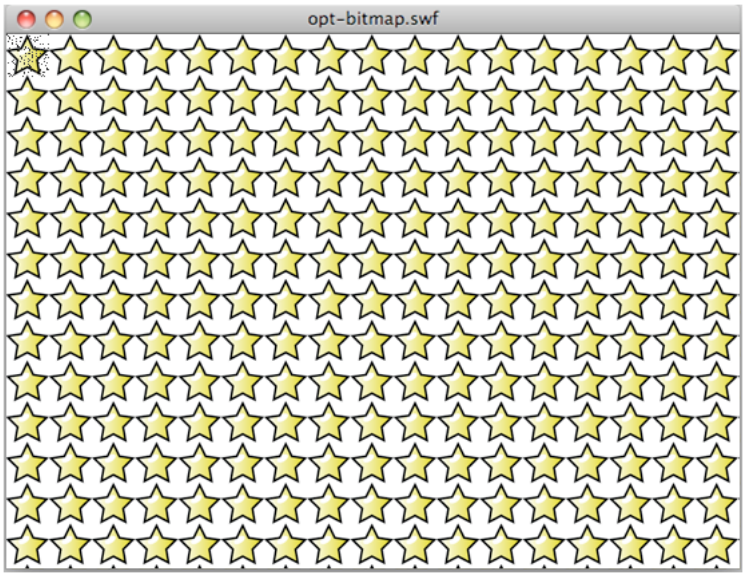
Risultato del codice che crea più istanze di un simbolo

Con Flash Player 10, ad esempio, l'animazione precedente utilizza circa 1008 KB di memoria. Con Flash Player 10.1, sia su un PC desktop che su un dispositivo mobile, l'animazione utilizza solo 4 KB.

Il codice seguente modifica una singola istanza BitmapData:

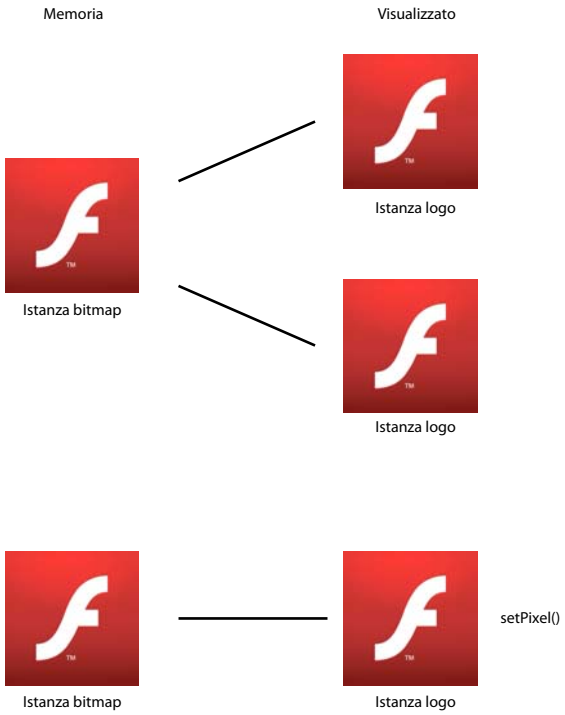
```
const MAX_NUM:int = 18;

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);
```

L'immagine seguente mostra il risultato della modifica di un'istanza `Star`:



Risultato della modifica di un'istanza

Internamente, il runtime assegna automaticamente e crea una bitmap in memoria per gestire le modifiche dei pixel. Quando viene chiamato un metodo della classe BitmapData, con conseguenti modifiche ai pixel, viene creata una nuova istanza in memoria e non vengono aggiornate altre istanze. La figura seguente illustra questo concetto:



Risultato in memoria della modifica di una bitmap

Se si modifica una stella, ne viene creata una nuova copia in memoria. L'animazione risultante utilizza circa 8 KB in memoria con Flash Player 10.1 e AIR 2.5.

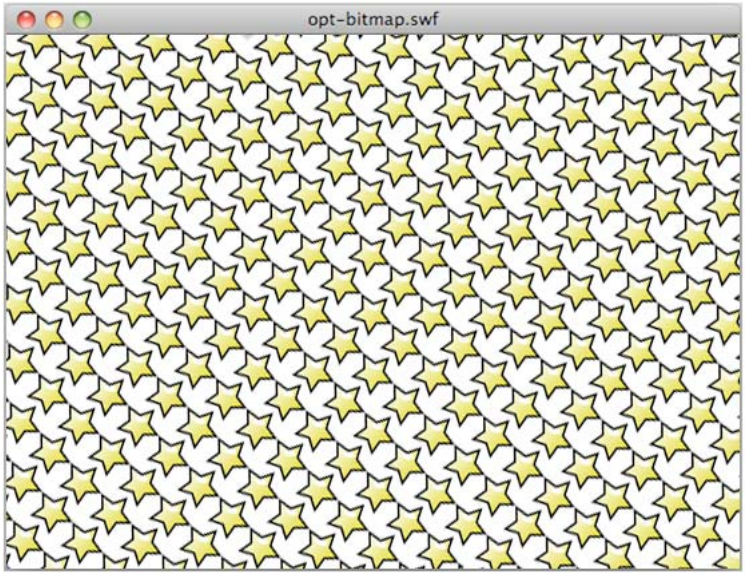
Nell'esempio precedente, ogni bitmap è disponibile per la trasformazione. Per creare solo l'effetto a piastrelle, il metodo `beginBitmapFill()` è quello più appropriato:

```
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);
```

Questo approccio produce lo stesso risultato con una singola istanza BitmapData creata. Per ruotare le stelle continuamente, anziché accedere a ogni istanza Star, usate un oggetto Matrix che viene ruotato su ogni fotogramma. Passate questo oggetto Matrix al metodo `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);
}
```

Se si usa questa tecnica, non è necessario un ciclo ActionScript per creare l'effetto. Tutte le operazioni vengono eseguite internamente dal runtime. L'immagine seguente mostra il risultato della trasformazione delle stelle:



Risultato della rotazione delle stelle

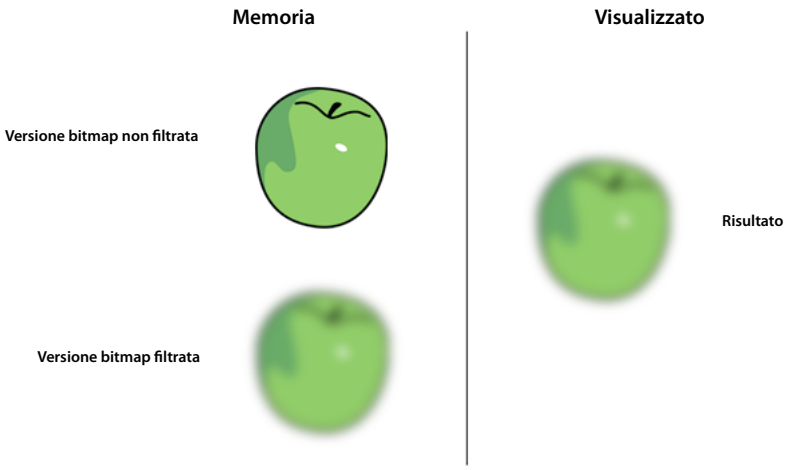
Con questo approccio, l'aggiornamento dell'oggetto di origine BitmapData originale aggiorna automaticamente le istanze di tale oggetto in altre parti dello stage, in base a una tecnica che si presta ad applicazioni molti sofisticate. Tuttavia, questo tipo di approccio non consentirebbe il ridimensionamento di ogni singola stella, come nell'esempio precedente.

Nota: *quando si utilizzano più istanze della stessa immagine, il modo in cui le immagini vengono disegnate dipende dal fatto che una classe sia associata o meno alla bitmap originale in memoria. Se nessuna classe è associata alla bitmap, le immagini vengono disegnate come oggetti Shape con riempimenti bitmap.*

Filtri e scaricamento dinamico delle bitmap

Evitate l'uso dei filtri, compresi quelli elaborati tramite Pixel Bender.

Cercate di limitare l'uso di effetti quali i filtri, compresi i filtri elaborati nei dispositivi mobili tramite Pixel Bender. Quando un filtro viene applicato a un oggetto di visualizzazione, il runtime crea due bitmap in memoria. Ciascuna di queste bitmap ha le dimensioni dell'oggetto di visualizzazione. La prima viene creata come versione rasterizzata dell'oggetto di visualizzazione, che a sua volta viene utilizzata per produrre una seconda bitmap con il filtro applicato:



Due bitmap in memoria quando è applicato un filtro

Quando si modifica una delle proprietà di un filtro, entrambe le bitmap vengono aggiornate in memoria per creare la bitmap risultante. Questo processo comporta una certa attività di elaborazione per la CPU e le due bitmap possono utilizzare una quantità significativa di memoria.

Flash Player 10.1 e AIR 2.5 introducono un nuovo comportamento di filtraggio per tutte le piattaforme. Se il filtro non viene modificato entro 30 secondi, oppure è nascosto o fuori dello schermo, la memoria utilizzata dalla bitmap non filtrata viene liberata.

Questa funzione consente di risparmiare metà della memoria utilizzata da un filtro su tutte le piattaforme. Ad esempio, considerate un oggetto di testo al quale sia applicato un filtro di sfocatura. Il testo in questo caso viene utilizzato per una semplice decorazione e non subisce modifiche. Dopo 30 secondi, la bitmap non filtrata in memoria viene liberata. Lo stesso risultato si verifica se il testo rimane nascosto o fuori schermo per 30 secondi. Quando una delle proprietà del filtro viene modificata, la bitmap non filtrata viene ricreata in memoria. Questa funzione prende il nome di "scaricamento dinamico delle bitmap". Anche con queste ottimizzazioni, usate i filtri con parsimonia poiché richiedono comunque un'intensa attività di elaborazione da parte della CPU o GPU quando vengono modificati.

Di norma, ogni volta che è possibile, utilizzate le bitmap create con un software di authoring come Adobe® Photoshop® per emulare i filtri. Evitate di utilizzare le bitmap dinamiche create in runtime in ActionScript. L'uso di bitmap create esternamente consente al runtime di ridurre il carico di lavoro per la CPU o la GPU, specialmente quando le proprietà del filtro non cambiano nel corso del tempo. Se possibile, create gli effetti che volete aggiungere a una bitmap in un programma di authoring. Potete quindi visualizzare la bitmap in runtime senza ulteriori elaborazioni, ottenendo un risultato potenzialmente molto più rapido.

Mipmapping diretto

Usate il mipmapping per ridimensionare le immagini di grandi dimensioni, se necessario.

Un'altra nuova funzione disponibile in Flash Player 10.1 e AIR 2.5 su tutte le piattaforme riguarda il mipmapping. In Flash Player 9 e AIR 1.0 era stata introdotta una funzione di mipmapping che migliorava la qualità e le prestazioni delle bitmap ridimensionate.

Nota: *la funzione di mipmapping può essere applicata solo alle immagini caricate dinamicamente oppure alle bitmap incorporate, mentre non è disponibile per gli oggetti di visualizzazione che sono stati filtrati o memorizzati nella cache. Il mipmapping può essere elaborato solo se la larghezza e l'altezza della bitmap sono numeri pari. Se viene rilevata una larghezza o un'altezza espressa con un numero dispari, il mipmapping si interrompe. Ad esempio, un'immagine 250 x 250 può essere ridimensionata tramite mipmapping a 125 x 125, ma non oltre. In questo caso, almeno una delle dimensioni è un numero dispari. Le bitmap con dimensioni corrispondenti a potenze di due producono i risultati migliori; ad esempio: 256 x 256, 512 x 512, 1024 x 1024 e così via.*

Ad esempio, immaginate che venga caricata un'immagine 1024 x 1024 e che uno sviluppatore desideri ridimensionarla per creare una miniatura per una galleria di immagini. La funzione di mipmapping esegue correttamente il rendering dell'immagine quando questa viene ridimensionata utilizzando le versioni intermedie di downsampling della bitmap come texture. Nelle versioni precedenti del runtime venivano create versioni ridotte intermedie della bitmap in memoria. Se un'immagine 1024 x 1024 veniva caricata e visualizzata a 64 x 64, le versioni precedenti del runtime creavano tutte le bitmap con dimensioni dimezzate. Ad esempio, in questo caso sarebbero state create bitmap 512 x 512, 256 x 256, 128 x 128 e 64 x 64.

Flash Player 10.1 e AIR 2.5 ora supportano il mipmapping direttamente dall'immagine originale fino alle dimensioni di destinazione richieste. Facendo riferimento all'esempio precedente, verrebbero create solo la bitmap originale da 4 MB (1024 x 1024) e la bitmap da 16 KB (64 x 64) ridotta tramite mipmapping.

La logica di mipmapping funziona anche con la funzione di scaricamento dinamico delle bitmap. Se viene utilizzata solo la bitmap da 64 x 64, la bitmap originale da 4 MB viene liberata dalla memoria. L'originale viene ricaricato solo se la mipmap deve essere ricreata. Inoltre, se sono richieste altre bitmap di mipmapping di varie dimensioni, per creare le bitmap viene utilizzata la catena di mipmap delle bitmap. Ad esempio, se deve essere creata una bitmap 1:8, vengono esaminate le bitmap 1:4, 1:2 e 1:1 per determinare quale deve essere caricata per prima in memoria. Se non vengono trovate altre versioni, la bitmap originale 1:1 viene caricata dalla risorsa e utilizzata.

Il decompressore JPEG può eseguire il mipmapping nel proprio formato. Questo mipmapping diretto consente di decomprimere una bitmap di grandi dimensioni direttamente ottenendo un formato mipmap senza caricare l'intera immagine non compressa. La generazione della mipmap risulta sensibilmente più veloce e la memoria utilizzata dalle bitmap di grandi dimensioni non viene allocata e poi liberata. La qualità delle immagini JPEG è paragonabile a quella ottenuta con la tecnica di mipmapping generale.

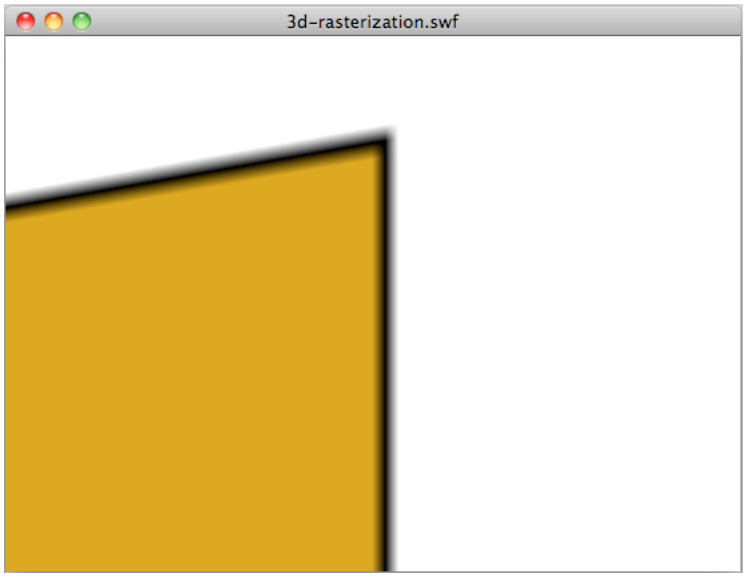
Nota: *utilizzate il mipmapping con moderazione. Anche se contribuisce a migliorare la qualità delle bitmap ridimensionate, il mipmapping influisce sull'ampiezza di banda, sulla memoria e sulla velocità. In alcuni casi può essere più opportuno utilizzare una versione pre-ridimensionata della bitmap creata con un tool esterno e importarla nell'applicazione. Non cominciate con bitmap di grandi dimensioni se intendete soltanto ridimensionarle in riduzione.*

Uso degli effetti 3D

Valutate la possibilità di creare manualmente gli effetti 3D.

In Flash Player 10 e AIR 1.5 è stato introdotto un motore 3D che consente di applicare una trasformazione prospettica agli oggetti di visualizzazione. Potete applicare queste trasformazioni utilizzando le proprietà `rotationX` e `rotationY` oppure il metodo `drawTriangles()` della classe Graphics. È anche possibile applicare la profondità mediante la proprietà `z`. Tenete presente che ogni oggetto di visualizzazione con trasformazione prospettica viene rasterizzato come bitmap e di conseguenza richiede una quantità maggiore di memoria.

La figura seguente illustra l'effetto di antialiasing prodotto dalla rasterizzazione quando si utilizza la trasformazione prospettica:



Antialiasing risultante dalla trasformazione prospettica

L'antialiasing è il risultato della rasterizzazione dinamica di un contenuto vettoriale come bitmap. Si verifica quando utilizzate effetti 3D nella versione desktop di AIR e Flash Player, oltre che in AIR 2.0.1 e AIR 2.5 per dispositivi mobili, mentre non viene applicato in Flash Player per dispositivi mobili.

Se create manualmente un effetto 3D senza avvalervi dell'API nativa, potete ridurre la quantità di memoria utilizzata. D'altra parte, le nuove funzioni 3D introdotte in Flash Player 10 e AIR 1.5 facilitano il mapping delle texture, grazie all'uso di metodi come `drawTriangles()`, che gestisce il mapping delle texture in modo nativo.

In qualità di sviluppatore, dovete decidere se l'effetto 3D che desiderate ottenere fornisce prestazioni migliori se elaborato con l'API nativa oppure manualmente. Valutate le prestazioni di rendering e di esecuzione di ActionScript, così come l'utilizzo della memoria.

Nelle applicazioni per dispositivi mobili AIR 2.0.1 e AIR 2.5 in cui impostate la proprietà dell'applicazione `renderMode` su `GPU`, le trasformazioni 3D vengono eseguite dalla GPU. Se però impostate la proprietà `renderMode` su `CPU`, sarà la CPU, e non la GPU, a eseguire le trasformazioni 3D. Nelle applicazioni Flash Player 10.1, le trasformazioni 3D vengono eseguite dalla CPU.

In questo caso, tenete presente che l'applicazione di una qualsiasi trasformazione 3D a un oggetto di visualizzazione richiede due bitmap in memoria: una per la bitmap di origine e una seconda per la versione risultante dalla trasformazione prospettica. In questo modo, le trasformazioni 3D funzionano in maniera simile ai filtri. Di conseguenza, è consigliabile utilizzare le proprietà 3D con moderazione quando le trasformazioni 3D vengono eseguite dalla CPU.

Oggetti di testo e memoria

Utilizzate il nuovo motore di testo di Adobe® Flash® per il testo di sola lettura e gli oggetti TextField per il testo di input.

In Flash Player 10 e AIR 1.5 è stato introdotto un nuovo e sofisticato motore di testo, Adobe Flash Text Engine (FTE), che permette di risparmiare memoria di sistema. Tuttavia, FTE è un'API di basso livello che richiede un livello ActionScript 3.0 aggiuntivo, fornito con il pacchetto flash.text.engine.

Per il testo di sola lettura è meglio utilizzare Flash Text Engine, che garantisce un risparmio di memoria e un rendering migliore. Per il testo di input, invece, gli oggetti TextField sono più indicati, perché richiedono meno codice ActionScript per la creazione dei comportamenti tipici quali la gestione dell'input e l'invio a capo del testo.

Altri argomenti presenti nell'Aiuto

"Rendering di oggetti di testo" a pagina 69

Modello a eventi e callback a confronto

Valutate la possibilità di utilizzare semplici callback anziché il modello a eventi.

Il modello a eventi di ActionScript 3.0 si basa sul concetto di invio degli oggetti. È orientato agli oggetti e ottimizzato per il riutilizzo del codice. Il metodo `dispatchEvent()` scorre ciclicamente l'elenco dei listener e chiama il metodo del gestore di eventi per ogni oggetto registrato. Tuttavia, uno degli svantaggi del modello a eventi consiste nel fatto che possono essere creati molti oggetti nell'arco della vita operativa dell'applicazione.

Immaginate di dover inviare un evento dalla linea temporale per indicare la fine di una sequenza di animazione. Per eseguire la notifica, potete inviare un evento da un fotogramma specifico della linea temporale, come illustrato dal codice seguente:

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

La classe Document può intercettare l'evento con la seguente riga di codice:

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

Per quanto questo tipo di approccio sia corretto, l'uso del modello a eventi nativo può risultare più lento e consumare più memoria rispetto al ricorso a una funzione di callback tradizionale. Gli oggetti evento devono essere creati e allocati in memoria, con un conseguente rallentamento delle prestazioni. Ad esempio, quando si intercetta l'evento Event.ENTER_FRAME, viene creato un nuovo oggetto evento su ciascun fotogramma per il gestore di eventi. Le prestazioni possono risultare particolarmente lente con gli oggetti di visualizzazione, a causa delle fasi di cattura e bubbling, che possono appesantire l'elaborazione se l'elenco di visualizzazione è complesso.

Capitolo 3: Riduzione dell'uso della CPU

Un'altra area importante di ottimizzazione è l'uso della CPU. L'ottimizzazione dei processi di elaborazione della CPU migliora le prestazioni e, di conseguenza, consente di estendere la durata della batteria nei dispositivi mobili.

Miglioramenti di Flash Player 10.1 per l'uso della CPU

In Flash Player 10.1 sono state introdotte due nuove funzioni che permettono di ridurre l'elaborazione della CPU. Mettono infatti in pausa e riprendono l'esecuzione del contenuto SWF quando esce dallo schermo e limitano il numero di istanze di Flash Player in una pagina.

Pause, throttle, resume

Nota: *la funzione "pause, throttle, resume" (pausa, rallentamento e ripresa) non è utilizzabile per le applicazioni Adobe® AIR®.*

Per ottimizzare l'uso della CPU e della batteria, Flash Player 10.1 introduce una nuova funzione relativa alle istanze inattive. Lo scopo di questa funzione è quello di limitare l'uso della CPU mettendo in pausa e quindi riprendendo il file SWF quando il contenuto esce e quindi rientra nello schermo. Grazie a questa funzione, Flash Player rilascia più memoria possibile rimuovendo gli oggetti che possono essere ricreati alla ripresa del contenuto. Il contenuto viene considerato "fuori schermo" quando esce completamente dallo schermo.

Due scenari comportano l'uscita del contenuto SWF dallo schermo:

- L'utente fa scorrere la pagina facendo uscire il contenuto SWF dallo schermo.

  In questo caso, se non è in corso la riproduzione di audio o video, il contenuto continua a essere riprodotto, ma il rendering viene interrotto. Se non viene riprodotto audio o video, per avere la certezza che la riproduzione o l'esecuzione di codice ActionScript non venga sospesa, impostate il parametro HTML `hasPriority` su true. Tenete tuttavia presente che il rendering del contenuto SWF viene messo in pausa quando il contenuto è fuori schermo o nascosto, indipendentemente dal valore del parametro HTML `hasPriority`.

- Viene aperta una scheda nel browser e di conseguenza il contenuto SWF viene portato in secondo piano.

  In questo caso, a prescindere dal valore del tag HTML `hasPriority`, il contenuto SWF viene rallentato (operazione denominata *throttle*) fino a un valore compreso tra 2 e 8 fps. La riproduzione di audio e video viene interrotta e il rendering del contenuto non viene elaborato, a meno che il contenuto SWF non diventi nuovamente visibile.

Per Flash Player 11.2 e versioni successive eseguite nei browser desktop Windows e Mac, potete usare l'evento ThrottleEvent nell'applicazione. Flash Player invia un evento ThrottleEvent quando Flash Player esegue una pausa, un rallentamento o una ripresa (pause, throttle o resume) della riproduzione.

ThrottleEvent è un evento di trasmissione e ciò significa che viene inviato da tutti gli oggetti EventDispatcher con un listener registrato per questo evento. Per ulteriori informazioni sugli eventi di trasmissione, vedete la classe DisplayObject.

Gestione delle istanze

Nota: *la funzione di gestione delle istanze non è utilizzabile nelle applicazioni Adobe® AIR®.*

Utilizzate il parametro HTML `hasPriority` per ritardare il caricamento dei file SWF fuori schermo.

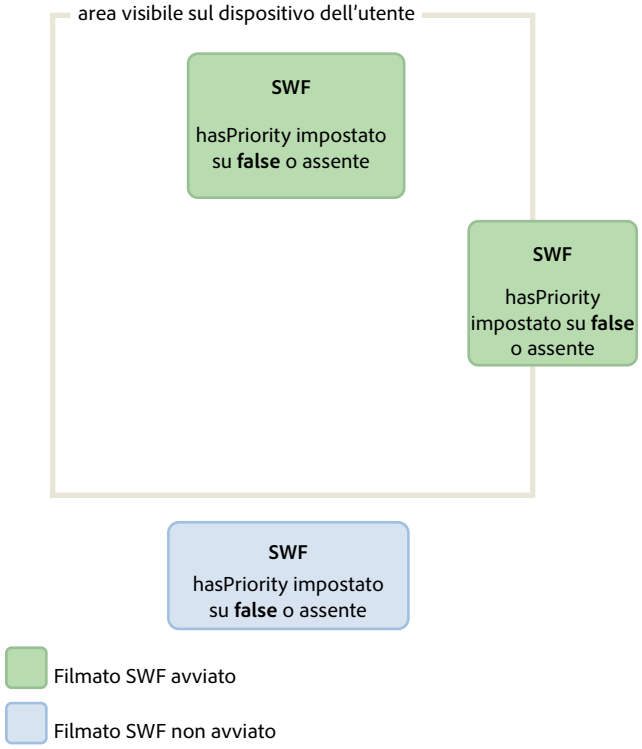
Flash Player 10.1 introduce un nuovo parametro HTML chiamato `hasPriority`:

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

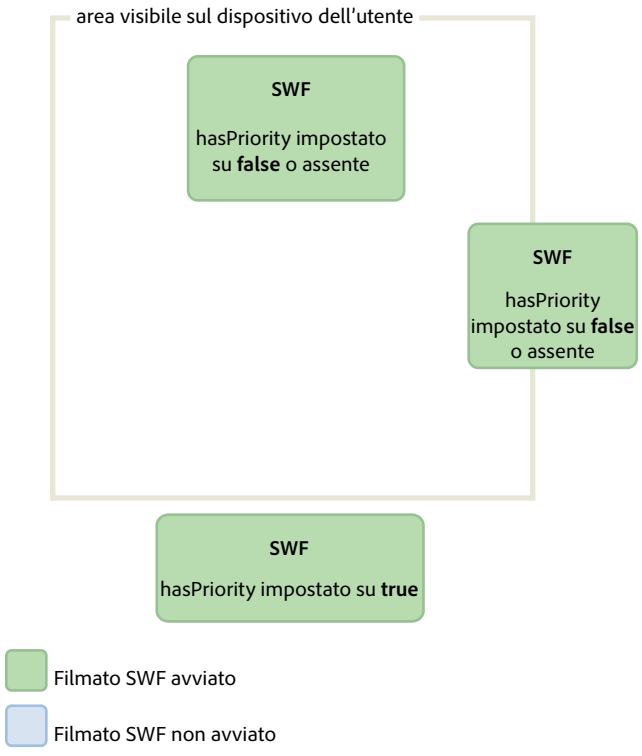
Questa funzione permette di limitare il numero di istanze di Flash Player che vengono avviate su una pagina, riducendo il consumo della CPU e risparmiando le risorse della batteria. L'idea è quella di assegnare una priorità specifica al contenuto SWF, dando a un contenuto particolare di una pagina una priorità maggiore rispetto a un altro contenuto della stessa pagina. Considerate un semplice esempio: un utente sta navigando in un sito web e la pagina dell'indice contiene tre file SWF differenti. Uno di essi è visibile, un altro è solo parzialmente visibile sullo schermo e l'ultimo è fuori schermo, quindi richiede lo scorrimento della pagina per essere visualizzato. Le prime due animazioni vengono avviate normalmente ma l'ultima viene ritardata fino al momento in cui il contenuto diventa visibile. Questo scenario corrisponde al comportamento predefinito quando il parametro `hasPriority` non è presente o è impostato su `false`. Per fare in modo che un file SWF venga avviato anche se è fuori schermo, impostate il parametro `hasPriority` su `true`. Tuttavia, indipendentemente dal valore del parametro `hasPriority`, il rendering di un file SWF che non è visibile all'utente viene sempre messo in pausa.

Nota: *se le risorse della CPU disponibili cominciano a esaurirsi, le istanze di Flash Player non vengono più avviate automaticamente, anche se il parametro `hasPriority` è impostato su `true`. Se vengono create nuove istanze tramite JavaScript dopo che la pagina è stata caricata, tali istanze ignoreranno la proprietà `hasPriority`. Eventuali contenuti da 1x1 pixel o 0x0 pixel vengono avviati, impedendo il differimento dei file di supporto SWF se il webmaster non ha incluso la proprietà `hasPriority`. È comunque ancora possibile avviare i file SWF con un clic. Questo comportamento viene chiamato "click to play" (riproduzione tramite clic).*

I diagrammi seguenti mostrano gli effetti dell'impostazione del parametro `hasPriority` su valori differenti:



Effetti di valori differenti del parametro hasPriority



Effetti di valori differenti del parametro hasPriority

Modalità sleep

Flash Player 10.1 e AIR 2.5 dispongono di una nuova funzione per i dispositivi mobili che consente di ridurre l'attività di elaborazione della CPU e, di conseguenza, aumentare l'autonomia della batteria. Questa funzione sfrutta la retroilluminazione disponibile in molti dispositivi mobili. Se un utente sta utilizzando, ad esempio, un'applicazione mobile e a un certo punto viene interrotto e smette di utilizzare il dispositivo, il runtime rileva il momento in cui la retroilluminazione entra in modalità sleep. Abbassa quindi la frequenza dei fotogrammi a 4 fotogrammi al secondo (fps) e sospende il rendering. Per le applicazioni AIR, la modalità sleep viene attivata anche quando l'applicazione passa all'esecuzione in background.

il codice ActionScript continua a essere eseguito in modalità sleep, analogamente a quanto avviene quando la proprietà `Stage.frameRate` è impostata su 4 fps. La fase di rendering però viene saltata, quindi l'utente non nota che il lettore sta funzionando a 4 fps. È stata scelta una frequenza di 4 fps anziché zero perché consente di mantenere aperte tutte le connessioni (NetStream, Socket e NetConnection), mentre una frequenza pari a zero le chiuderebbe. È stata scelta una frequenza di aggiornamento di 250 ms (4 fps) perché molti produttori di dispositivi mobili usano lo stesso valore per la frequenza di aggiornamento del dispositivo. In questo modo la frequenza di aggiornamento del runtime corrisponde esattamente a quella del dispositivo.

Nota: *quando il runtime è in modalità sleep, la proprietà* `Stage.frameRate` *ritorna alla frequenza fotogrammi del file SWF originale, invece che a 4 fps.*

Quando la retroilluminazione viene riattivata, il rendering viene ripreso e la frequenza dei fotogrammi ritorna all'impostazione originale. Immaginate un'applicazione come un lettore multimediale utilizzata da un utente per ascoltare della musica. Se lo schermo entra in modalità sleep, il runtime risponde in base al tipo di contenuto in corso di riproduzione. Ecco alcune possibili situazioni con il comportamento del runtime corrispondente:

- La retroilluminazione entra in modalità sleep ed è in corso di riproduzione un contenuto non audiovisivo: il rendering viene messo in pausa e la frequenza fotogrammi impostata su 4 fps.
- La retroilluminazione entra in modalità sleep ed è in corso di riproduzione un contenuto audiovisivo: il runtime forza la retroilluminazione a rimanere attiva per non interrompere le attività dell'utente.
- La retroilluminazione passa dalla modalità sleep alla modalità attiva: il runtime imposta la frequenza fotogrammi sullo stesso valore di quella del file SWF originale e riprende il rendering.
- Flash Player viene messo in pausa durante la riproduzione di un contenuto audiovisivo: il comportamento di sistema predefinito della retroilluminazione viene ripristinato perché non è più in esecuzione un contenuto audiovisivo.
- Il dispositivo mobile riceve una chiamata mentre è in corso di riproduzione un contenuto audiovisivo: il rendering viene messo in pausa e la frequenza fotogrammi impostata su 4 fps.
- La modalità sleep della retroilluminazione viene disattivata su un dispositivo mobile: il runtime continua a funzionare normalmente.

Quando la retroilluminazione entra in modalità sleep, li rendering viene sospeso e la frequenza fotogrammi rallenta. Questa funzione consente di ridurre il carico di lavoro della CPU ma non è affidabile per creare una vera pausa, come in un videogioco.

Nota: *nessun evento ActionScript viene inviato quando il runtime entra o esce dalla modalità sleep.*

Blocco e sblocco degli oggetti

Per bloccare e sbloccare correttamente gli oggetti, utilizzate gli eventi `REMOVED_FROM_STAGE` *e* `ADDED_TO_STAGE`*.*

Per ottimizzare il codice, bloccate e sbloccate sempre gli oggetti. È importante farlo per tutti gli oggetti, ma in modo particolare per gli oggetti di visualizzazione. Anche se degli oggetti di visualizzazione non sono più presenti nell'elenco di visualizzazione e sono in attesa di essere eliminati dal processo di garbage collection, potrebbero comunque continuare a utilizzare codice caratterizzato da un uso intensivo della CPU. Ad esempio, potrebbero ancora utilizzare l'evento Event.ENTER_FRAME. Di conseguenza, è importantissimo bloccare e sbloccare correttamente gli oggetti utilizzando gli eventi `Event.REMOVED_FROM_STAGE` and `Event.ADDED_TO_STAGE`. L'esempio seguente mostra un clip filmato in riproduzione sullo stage che interagisce con la tastiera:

```
// 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;
    }
}
```

Clip filmato che interagisce con la tastiera

Quando viene fatto clic sul pulsante Remove, il clip filmato viene rimosso dall'elenco di visualizzazione:

```
// 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);
}
```

Ciononostante, il clip filmato continua a inviare l'evento `Event.ENTER_FRAME`. Il clip filmato è ancora in esecuzione, ma non ne viene eseguito il rendering. Per gestire correttamente questa situazione, intercettate gli eventi appropriati e rimuovete i listener di eventi, per impedire l'esecuzione del codice che fa un uso intensivo della CPU:

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

Quando si preme il pulsante Show, il clip filmato viene riavviato, gli eventi `Event.ENTER_FRAME` vengono intercettati di nuovo e la tastiera controlla correttamente il clip filmato.

Nota: *se un oggetto di visualizzazione viene rimosso dall'elenco di visualizzazione, l'impostazione del suo riferimento su `null` dopo la rimozione non garantisce che l'oggetto venga bloccato. Se il garbage collector non viene eseguito, l'oggetto continua a consumare memoria e risorse della CPU anche se non è più visualizzato. Per fare in modo che l'oggetto consumi la quantità minima possibile di risorse della CPU, bloccatelo completamente quando lo rimuovete dall'elenco di visualizzazione.*

A partire da Flash Player 10 e AIR 1.5, è presente anche il seguente comportamento. Se l'indicatore di riproduzione incontra un fotogramma vuoto, l'oggetto di visualizzazione viene bloccato automaticamente anche se non avete implementato alcun comportamento di blocco.

Il concetto di blocco è importante anche quando caricate contenuto tramite la classe Loader. Quando si usava la classe Loader con Flash Player 9 e AIR 1.0, era necessario bloccare manualmente il contenuto intercettando l'evento `Event.UNLOAD` inviato dall'oggetto LoaderInfo. Ogni oggetto doveva essere bloccato manualmente e il carico di lavoro non era indifferente. Flash Player 10 e AIR 1.5 hanno introdotto un importante nuovo metodo nella classe Loader, ovvero `unloadAndStop()`. Questo metodo vi consente di scaricare un file SWF, bloccare automaticamente ogni oggetto nel file SWF caricato e imporre l'esecuzione del garbage collector.

Nel codice seguente, il file SWF viene caricato e quindi scaricato con il metodo `unload()`, che richiede più lavoro di elaborazione e il blocco manuale:

```
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();
}
```

È buona norma utilizzare il metodo `unloadAndStop()`, che gestisce il blocco in modo nativo e impone l'esecuzione del processo di garbage collection:

```
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();
}
```

Le azioni seguenti vengono eseguite quando si chiama il metodo `unloadAndStop()`:

- I suoni vengono interrotti.
- I listener registrati nella linea temporale principale del file SWF vengono rimossi.
- Gli oggetti Timer vengono interrotti.
- Le periferiche hardware (ad esempio la fotocamera e il microfono) vengono rilasciate.
- Ogni clip filmato viene interrotto.
- L'invio degli eventi `Event.ENTER_FRAME`, `Event.FRAME_CONSTRUCTED`, `Event.EXIT_FRAME`, `Event.ACTIVATE` e `Event.DEACTIVATE` viene interrotto.

Eventi activate e deactivate

Utilizzate gli eventi `Event.ACTIVATE` e `Event.DEACTIVATE` per rilevare l'inattività in background e ottimizzare l'applicazione di conseguenza.

Due eventi, (`Event.ACTIVATE` e `Event.DEACTIVATE`), consentono di regolare l'applicazione in modo che utilizzi il minor numero possibile di cicli della CPU e di rilevare quando il runtime viene attivato o disattivato. Il codice può, di conseguenza, essere ottimizzato per reagire alle modifiche del contesto. Il codice seguente intercetta entrambi gli eventi e modifica dinamicamente la frequenza fotogrammi impostandola su zero quando l'applicazione viene disattivata. L'applicazione non è più attiva, ad esempio, quando l'utente passa a un'altra scheda o esegue l'applicazione in background:

```
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;
}
```

Quando l'applicazione viene riattivata, la frequenza fotogrammi ritorna al valore originale. Anziché modificare la frequenza fotogrammi dinamicamente, potete considerare altre modalità di ottimizzazione, ad esempio bloccare o sbloccare gli oggetti.

Gli eventi activate e deactivate consentono di implementare un meccanismo simile alla funzione "Pausa e ripresa" disponibile a volte sui dispositivi mobili e sui netbook.

Altri argomenti presenti nell'Aiuto

"Frequenza fotogrammi dell'applicazione" a pagina 54

"Blocco e sblocco degli oggetti" a pagina 28

Interazioni con il mouse

Valutate la possibilità di disattivare l'interazione con il mouse, quando è possibile.

Quando usate un oggetto interattivo, ad esempio un oggetto MovieClip o Sprite, il runtime esegue il codice nativo per rilevare e gestire le interazioni con il mouse. Il rilevamento dell'interazione con il mouse può imporre un notevole carico di lavoro alla CPU quando sullo schermo sono visualizzati molti oggetti interattivi, specialmente se sovrapposti. Un modo facile per evitare questo lavoro di elaborazione è quello di disattivare l'interazione con il mouse per gli oggetti per i quali non è richiesta. Il codice dell'esempio seguente illustra l'uso delle proprietà `mouseEnabled` e `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;
```

Quando è possibile, disattivate l'interazione con il mouse al fine di limitare l'uso della CPU da parte dell'applicazione e conseguentemente ridurre il consumo della batteria sui dispositivi mobili.

Timer ed eventi ENTER_FRAME

Scegliete i timer oppure gli eventi `ENTER_FRAME`, a seconda che il contenuto sia animato o meno.

I timer sono da preferire agli eventi `Event.ENTER_FRAME` per i contenuti non animati che vengono eseguiti per un tempo lungo.

In ActionScript 3.0 sono disponibili due modi per chiamare una funzione a intervalli specifici. Il primo è quello di utilizzare l'evento `Event.ENTER_FRAME` inviato dagli oggetti di visualizzazione (DisplayObject). Il secondo approccio prevede l'uso di un timer. Gli sviluppatori ActionScript ricorrono spesso all'uso dell'evento `ENTER_FRAME`. L'evento `ENTER_FRAME` viene inviato su ogni fotogramma. Di conseguenza, l'intervallo con il quale la funzione viene chiamata è correlato alla frequenza fotogrammi corrente. La frequenza fotogrammi è accessibile tramite la proprietà `Stage.frameRate`. Tuttavia, in alcuni casi, l'uso di un timer può rivelarsi una scelta migliore rispetto all'uso dell'evento `ENTER_FRAME`. Ad esempio, se non usate animazioni ma volete che il codice venga chiamato a intervalli specifici, l'impiego di un timer può essere l'opzione migliore.

Un timer può comportarsi in modo analogo all'evento `ENTER_FRAME`, ma un evento può essere inviato indipendentemente dalla frequenza dei fotogrammi. Questo comportamento può offrire una certa ottimizzazione. Prendete come esempio un'applicazione lettore video. In questo caso, non avete bisogno di una frequenza fotogrammi elevata, perché solo i controlli dell'applicazione si muovono.

***Nota:** la frequenza fotogrammi non influisce sul video, poiché il video non è incorporato nella linea temporale, bensì viene caricato dinamicamente tramite streaming o download progressivo.*

In questo esempio, la frequenza fotogrammi è impostata su un valore basso, 10 f/s. Il timer aggiorna i controlli a una frequenza di un aggiornamento al secondo. Questa frequenza di aggiornamento più elevata è resa possibile dal metodo `updateAfterEvent()`, disponibile sull'oggetto TimerEvent. Questo metodo esegue un aggiornamento forzato dello schermo ogni volta che il timer invia un evento, se necessario. Il codice seguente illustra questo concetto:

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

Una chiamata al metodo `updateAfterEvent()` non modifica la frequenza fotogrammi, bensì forza il runtime ad aggiornare il contenuto a video che è cambiato. La linea temporale viene comunque riprodotta a 10 f/s. Tenete presente che i timer e gli eventi `ENTER_FRAME` non sono perfettamente precisi sui dispositivi a basse prestazioni o nel caso in cui le funzioni dei gestori di eventi contengano codice che richiede una notevole elaborazione. Come la frequenza fotogrammi di un file SWF, anche la frequenza di aggiornamento dei fotogrammi del timer può variare in alcune situazioni.


Riducete al minimo il numero di oggetti Timer e di gestori `enterFrame` registrati nell'applicazione.

A ogni fotogramma, il runtime invia un evento `enterFrame` a tutti gli oggetti di visualizzazione inclusi nel suo elenco di visualizzazione. È possibile registrare i listener dell'evento `enterFrame` in più oggetti di visualizzazione, ma in questo caso a ogni fotogramma verrà eseguita una quantità superiore di codice. In alternativa potete utilizzare un unico gestore `enterFrame` centralizzato che gestisce tutto il codice che deve essere eseguito a ogni fotogramma. Centralizzando questo codice potrete gestire più facilmente tutto il codice che viene eseguito di frequente.

Analogamente, quando utilizzate gli oggetti Timer si verifica un sovraccarico di elaborazione dovuto alla creazione e all'invio di eventi da più oggetti Timer. Se dovete attivare diverse operazioni a intervalli diversi, fate riferimento alle alternative suggerite di seguito:

- Riducete al minimo il numero di oggetti Timer utilizzati e raggruppate le operazioni in base alla frequenza di esecuzione.

  Ad esempio, potete utilizzare un oggetto Timer per le operazioni frequenti e impostare la frequenza di attivazione su 100 millisecondi. Potete utilizzare un altro oggetto Timer per le operazioni meno frequenti o che vengono eseguite in background e impostare la frequenza di attivazione su 2000 millisecondi.

- Utilizzate un unico oggetto Timer e fate in modo che la frequenza di attivazione delle operazioni corrisponda a multipli dell'intervallo della proprietà `delay` dell'oggetto Timer.

  Supponiamo, ad esempio, che alcune operazioni debbano essere eseguite ogni 100 millisecondi e altre ogni 200 millisecondi. In questo caso, utilizzate un unico oggetto Timer con un valore `delay` di 100 millisecondi. Nel gestore di eventi `timer`, aggiungete un'istruzione condizionale che attiva l'esecuzione delle operazioni da 200 millisecondi una volta sì e una volta no. Questa tecnica viene dimostrata nell'esempio seguente:

```
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;
}
```

Arrestate gli oggetti Timer quando non sono in uso.

Se il gestore di eventi `timer` di un oggetto Timer esegue le operazioni solo se sussistono determinate condizioni, chiamate il metodo `stop()` dell'oggetto Timer quando non si verifica alcuna delle condizioni richieste.

Nei gestori di eventi `enterFrame` o Timer, riducete al minimo il numero di modifiche dell'aspetto degli oggetti di visualizzazione che determinano il ridisegno dello schermo.

A ogni fotogramma, la fase di rendering ridisegna la parte dello stage che è stata modificata durante il fotogramma. Per le aree di ridisegno di grandi dimensioni, o le aree di ridisegno di piccole dimensioni che contengono un numero elevato di oggetti di visualizzazione complessi, il runtime impiega più tempo per eseguire il rendering. Per verificare la quantità di ridisegno necessaria, utilizzate la funzionalità "Mostra aree ridisegno" nel debugger di Flash Player o AIR.

Per ulteriori informazioni su come migliorare le prestazioni per le azioni ripetute, vedete il seguente articolo:

- Articolo e applicazione di esempio pubblicati da Arno Gourdol: Writing well-behaved, efficient, AIR applications (Come scrivere applicazioni AIR efficienti e ad alte prestazioni)

Altri argomenti presenti nell'Aiuto

"Isolamento dei comportamenti" a pagina 66

Sindrome dell'interpolazione

Per risparmiare la CPU, limitate l'uso delle interpolazioni: consumerete meno in termini di elaborazione della CPU, memoria e autonomia della batteria.

I designer e gli sviluppatori che producono contenuti desktop per Flash tendono a utilizzare molte interpolazioni di movimento nelle loro applicazioni. Quando invece create contenuto per dispositivi mobili a basse prestazioni, limitate il più possibile l'uso delle interpolazioni di movimento, in modo da consentire una riproduzione più rapida del contenuto nei dispositivi meno potenti.

Capitolo 4: Prestazioni di ActionScript 3.0

Classe Vector e classe Array a confronto

Utilizzate la classe Vector invece della classe Array quando è possibile.

La classe Vector consente un accesso in lettura e scrittura più rapido rispetto alla classe Array.

Una prova molto semplice permette di evidenziare i vantaggi della classe Vector sulla classe Array. Il codice seguente mostra un benchmark per la classe Array:

```
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
```

Il codice seguente mostra un benchmark per la classe Vector:

```
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
```

Questo esempio può essere ulteriormente ottimizzato assegnando una lunghezza specifica al vettore e inizializzando la lunghezza:

```
// 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
```

Se le dimensioni del vettore non vengono specificate in anticipo, esse aumentano quando lo spazio del vettore si esaurisce. Ogni volta che le dimensioni del vettore aumentano, viene allocato un nuovo blocco di memoria. Il contenuto corrente del vettore viene copiato nel nuovo blocco di memoria di memoria. Questo allocazione e copiatura di dati supplementare appesantisce le prestazioni. Il codice dell'esempio precedente viene ottimizzato per le prestazioni specificando la dimensione iniziale del vettore. Tuttavia, il codice non è ottimizzato per la gestibilità. Per migliorarne la gestibilità, memorizzate in una costante il valore riutilizzato:

```
// 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
```

Cercate di utilizzare le API che utilizzano gli oggetti Vector, quando è possibile, poiché è probabile che assicurino un'esecuzione più rapida.

API di disegno

Utilizzate l'API di disegno per ottenere un'esecuzione più rapida del codice.

Flash Player 10 e AIR 1.5 offrono una nuova API di disegno che consente di ottenere prestazioni migliori nell'esecuzione del codice. La nuova API non migliora le prestazioni di rendering, ma consente di ridurre anche sensibilmente il numero di righe di codice da scrivere. Meno righe di codice assicurano un'esecuzione più efficiente del codice ActionScript.

La nuova API di disegno comprende i seguenti metodi:

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

Nota: *questa sezione non prende in esame il metodo* `drawTriangles()`*, che riguarda gli oggetti 3D. Tuttavia, questo metodo può migliorare le prestazioni del codice ActionScript, perché gestisce la mappatura delle texture in modo nativo.*

Il codice seguente chiama esplicitamente il metodo appropriato per ogni riga che viene disegnata:

```
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 );
```

Il codice seguente viene seguito più rapidamente di quello dell'esempio precedente perché è costituito da meno righe. Più complesso è il tracciato, maggiore è il guadagno in termini di prestazioni derivante dall'uso del metodo `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 );
```

Il metodo `drawGraphicsData()` offre un simile miglioramento delle prestazioni.

Cattura e bubbling degli eventi

Utilizzate la cattura e il bubbling degli eventi per ridurre l'uso dei gestori di eventi.

Il modello di eventi di ActionScript 3.0 ha introdotto i concetti di cattura e bubbling degli eventi. Sfruttando correttamente il bubbling di un evento è possibile ottimizzare i tempi di esecuzione del codice ActionScript. Potete registrare un gestore di eventi in un solo oggetto anziché in più oggetti, ottenendo un miglioramento delle prestazioni.

Per esempio, immaginate di creare un videogioco in cui l'utente deve fare clic su delle mele il più velocemente possibile per distruggerle. Ogni volta che si fa clic su una mela, questa viene rimossa dallo schermo e vengono aggiunti dei punti al punteggio dell'utente. Per intercettare l'evento `MouseEvent.CLICK` inviato da ogni mela, potreste essere tentati di scrivere il codice seguente:

```
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 );
}
```

Il codice chiama il metodo `addEventListener()` su ogni istanza Apple. Inoltre rimuove ogni listener quando viene fatto clic su una mela, utilizzando il metodo `removeEventListener()`. Tuttavia, nel modello di eventi di ActionScript 3.0 sono disponibili le fasi di cattura e di bubbling per alcuni eventi, quindi è possibile intercettarli tramite un InteractiveObject principale. Di conseguenza, è possibile ottimizzare il codice precedente e ridurre al minimo il numero di chiamate ai metodi `addEventListener()` e `removeEventListener()`. Il codice seguente usa la fase di cattura per intercettare gli eventi dell'oggetto principale:

```
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 );
}
```

Questo codice è più semplice e molto più ottimizzato, poiché contiene una singola chiamata al metodo `addEventListener()` sul contenitore principale. I listener non vengono più registrati nelle istanze Apple, quindi non è necessario rimuoverli quando viene fatto clic su una mela. Il gestore `onAppleClick()` può essere ulteriormente ottimizzato interrompendo la propagazione dell'evento:

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

È anche possibile utilizzare la fase di bubbling per intercettare l'evento, passando `false` come terzo parametro al metodo `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 );
```

Il valore predefinito per il parametro della fase di cattura è `false`, quindi potete ometterlo:

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

Operazioni con i pixel

Disegno dei pixel con il metodo `setVector()`.

Quando si disegnano dei pixel, alcune semplici ottimizzazioni possono essere effettuate utilizzando i metodi appropriati della classe BitmapData. Un modo veloce per disegnare i pixel consiste nell'usare il metodo `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 );
```

Quando usate metodi lenti, ad esempio `setPixel()` o `setPixel32()`, servitevi dei metodi `lock()` e `unlock()` per ottenere un'esecuzione più rapida. Nel codice seguente, i metodi `lock()` e `unlock()` vengono utilizzati per migliorare le prestazioni:

```
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
```

Il metodo `lock()` della classe BitmapData blocca un'immagine e impedisce agli oggetti che vi fanno riferimento di essere aggiornati quando l'oggetto BitmapData cambia. Ad esempio, se un oggetto Bitmap fa riferimento a un oggetto BitmapData, potete bloccare l'oggetto BitmapData, modificarlo e quindi sbloccarlo. L'oggetto Bitmap non viene modificato finché non sbloccate l'oggetto BitmapData. Per migliorare le prestazioni, utilizzate questo metodo in combinazione con `unlock()` prima e dopo numerose chiamate al metodo `setPixel()` o `setPixel32()`. Le chiamate a `lock()` e `unlock()` consentono di evitare l'aggiornamento non richiesto dello schermo.

Nota: *quando si elaborano i pixel di una bitmap che non si trova nell'elenco di visualizzazione (doppio buffering), questa tecnica a volte non migliora le prestazioni. Se l'oggetto bitmap non fa riferimento a un buffer bitmap, l'uso di `lock()` e `unlock()` non migliora le prestazioni. Flash Player rileva che il buffer non è associato a un riferimento e non esegue il rendering della bitmap sullo schermo.*

I metodi che eseguono iterazioni sui pixel, come `getPixel()`, `getPixel32()`, `setPixel()` e `setPixel32()`, risultano facilmente lenti, specialmente nei dispositivi mobili. Se possibile, utilizzate metodi che recuperano tutti i pixel in una sola chiamata. Per leggere i pixel, usate il metodo `getVector()`, che è più veloce di `getPixels()`. Inoltre, ricordatevi di usare le API che utilizzano gli oggetti Vector, quando è possibile, poiché è probabile che assicurino un'esecuzione più rapida.

Espressioni regolari

Utilizzate i metodi della classe String, ad esempio `indexOf()`, `substr()` o `substring()`, anziché un'espressione regolare per le operazioni di ricerca ed estrazione di base delle stringhe.

Per eseguire alcune operazioni è possibile utilizzare sia un'espressione regolare che i metodi della classe String. Per determinare se una stringa contiene un'altra stringa, ad esempio, potete utilizzare il metodo `String.indexOf()` o un'espressione regolare. Il metodo della classe String, tuttavia, viene eseguito più rapidamente dell'espressione regolare equivalente e non richiede la creazione di un altro oggetto.

Utilizzate un gruppo di non cattura ("`(?:xxxx)`") anziché un gruppo ("`(xxxx)`") in un'espressione regolare se desiderate raggruppare gli elementi senza isolare i contenuti del gruppo nel risultato.

Nelle espressioni regolari di moderata complessità, spesso alcune parti dell'espressione vengono raggruppate. Nel seguente modello di espressione regolare, ad esempio, le parentesi creano un gruppo per il testo "ab". Di conseguenza, il quantificatore "+" viene applicato al gruppo anziché a un solo carattere:

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

Per impostazione predefinita, i contenuti di ogni gruppo vengono "catturati". I contenuti di ogni gruppo del modello possono essere inclusi nel risultato prodotto dall'esecuzione dell'espressione regolare. La cattura dei risultati dei gruppi richiede una quantità superiore di tempo e memoria perché vengono creati degli oggetti per contenere questi risultati. In alternativa potete utilizzare la sintassi dei gruppi di non cattura includendo un punto interrogativo e i due punti dopo la parentesi aperta. Questa sintassi specifica che i caratteri si comportano come un gruppo ma non vengono catturati nei risultati:

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

La sintassi dei gruppi di non cattura è più rapida e utilizza una quantità di memoria inferiore rispetto alla sintassi dei gruppi standard.

Utilizzate un modello di espressione regolare alternativo se un'espressione regolare ha prestazioni scadenti.

A volte è possibile utilizzare più di un modello di espressione regolare per testare o individuare lo stesso modello di testo. Alcuni modelli vengono eseguiti più rapidamente di altri per vari motivi. Se determinate che un'espressione regolare rallenta l'esecuzione del codice più di quanto sia necessario, considerate la possibilità di utilizzare modelli di espressione regolare alternativi che consentano di ottenere lo stesso risultato. Provate questi modelli alternativi per individuare quello più rapido.

Ottimizzazioni di vario tipo

Per un oggetto TextField, usate il metodo `appendText()` invece dell'operatore `+=`.

Quando usate la proprietà `text` della classe TextField, utilizzate il metodo `appendText()` anziché l'operatore `+=`. L'uso del metodo `appendText()` garantisce prestazioni migliori.

Ad esempio, il codice seguente utilizza l'operatore `+=` e l'esecuzione completa del ciclo richiede 1120 ms:

```
addChild ( myTextField );

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

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

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

Nell'esempio seguente, l'operatore `+=` è stato sostituito dal metodo `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
```

L'esecuzione del nuovo codice richiede 847 ms.


Aggiornate i campi di testo esternamente ai cicli, quando potete.

Questo codice può essere ottimizzato ulteriormente grazie a una tecnica molto semplice. L'aggiornamento del campo di testo all'interno di ciascun ciclo consuma molte risorse di elaborazione interne. Concatenando semplicemente una stringa e assegnandola al campo di testo all'esterno del ciclo, si riduce sensibilmente il tempo necessario per l'esecuzione del codice. Ora l'esecuzione del codice richiede 2 ms:

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

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

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

myTextField.text = content;

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

Quando si lavora con il testo HTML, il primo approccio è così lento che in alcuni casi può essere generata un'eccezione `Timeout` in Flash Player. Ad esempio, può essere generata un'eccezione se l'hardware sottostante è troppo lento.

***Nota:** Adobe® AIR® non genera questa eccezione.*

```
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 );
```

Assegnando il valore a una stringa esterna al ciclo, il codice richiede solo 29 ms per essere eseguito:

```
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
```

Nota: *in Flash Player 10.1 e AIR 2.5, la classe String è stata migliorata e le stringhe usano meno memoria.*

Evitate di usare l'operatore parentesi quadra, quando è possibile.

L'uso dell'operatore parentesi quadra può rallentare le prestazioni. Potete evitare di utilizzarlo memorizzandone il riferimento in una variabile locale. L'esempio seguente mostra un uso inefficiente dell'operatore parentesi quadra:

```
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
```

La seguente versione ottimizzata riduce l'uso dell'operatore parentesi quadra:

```
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
```

Usate il codice in linea, quando è possibile, per ridurre il numero di chiamate di funzione all'interno del codice.

Le chiamate di funzione possono richiedere molte risorse. Cercate di ridurre il numero di chiamate di funzione spostando il codice in linea. Lo spostamento in linea del codice è un buon modo per ottimizzare le prestazioni. Tuttavia, tenete presente che il codice in linea può rendere più difficile il riutilizzo del codice e incrementare le dimensioni del file SWF. Alcune chiamate di funzione, come quelle ai metodi della classe Math, sono più facili da spostare in linea. Il codice seguente utilizza il metodo `Math.abs()` per calcolare dei valori assoluti:

```
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
```

Il calcolo eseguito da `Math.abs()` può essere eseguito manualmente e spostato in linea:

```
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
```

Lo spostamento in linea della chiamata di funzione produce un codice oltre quattro volte più veloce. Questo approccio è utile in molte situazioni, ma valutate bene gli effetti che può avere sulla possibilità di riutilizzare il codice e sulla sua gestibilità.

Nota: *le dimensioni del codice hanno un forte impatto sulle prestazioni generali di riproduzione nel lettore. Se l'applicazione contiene una grande quantità di codice ActionScript, la macchina virtuale necessita di molto tempo solo per verificare il codice ed eseguire la compilazione JIT. La ricerca delle proprietà può risultare lenta, a causa dei vari livelli di gerarchie di ereditarietà e perché le cache locali tendono a eliminare più dati. Per ridurre le dimensioni del codice, evitate di usare il framework Adobe® Flex®, la libreria del framework TLF o eventuali librerie ActionScript di terze parti di grandi dimensioni.*


Evitate di valutare le istruzioni nei cicli.

Un'altra ottimizzazione può essere ottenuta evitando di valutare un'istruzione in un ciclo. Il codice seguente esegue un'iterazione su un array, ma non viene ottimizzato perché la lunghezza dell'array viene valutata a ogni iterazione:

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

È meglio memorizzare il valore e riutilizzarlo:

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

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


Utilizzate un ordine inverso per i cicli while.

Un ciclo while in ordine inverso è più veloce di uno in ordine standard:

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

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

Questi suggerimenti indicano alcuni modi per ottimizzare il codice ActionScript, spiegando come una singola riga di codice può influire sulle prestazioni e sulla memoria. Sono possibili molte altre ottimizzazioni ActionScript. Per ulteriori informazioni, accedete al collegamento seguente: http://www.rozengain.com/blog/2007/05/01/some-actionscript-30-optimizations/.

Capitolo 5: Prestazioni di rendering

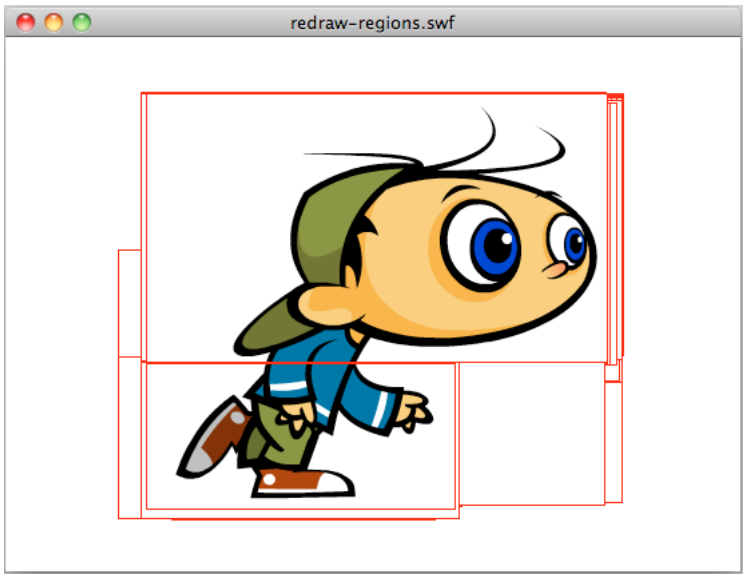
Aree di ridisegno

Quando create un progetto, utilizzate sempre l'opzione relativa alle aree di ridisegno.

Per migliorare il rendering, è importante utilizzare le aree di ridisegno durante la creazione di un progetto. L'utilizzo di questa opzione consente di vedere le aree durante il rendering e l'elaborazione da parte di Flash Player. Potete attivare questa opzione selezionando Mostra aree ridisegno nel menu di scelta rapida della versione di debug di Flash Player.

Nota: *l'opzione Mostra aree di ridisegno non è disponibile in Adobe AIR o nella versione release di Flash Player. (In Adobe AIR, il menu di scelta rapida è disponibile solo nelle applicazioni desktop, tuttavia non include voci incorporate o standard quale Mostra aree di ridisegno).*

L'immagine sottostante illustra l'opzione attivata con un semplice clip filmato animato sulla linea temporale:



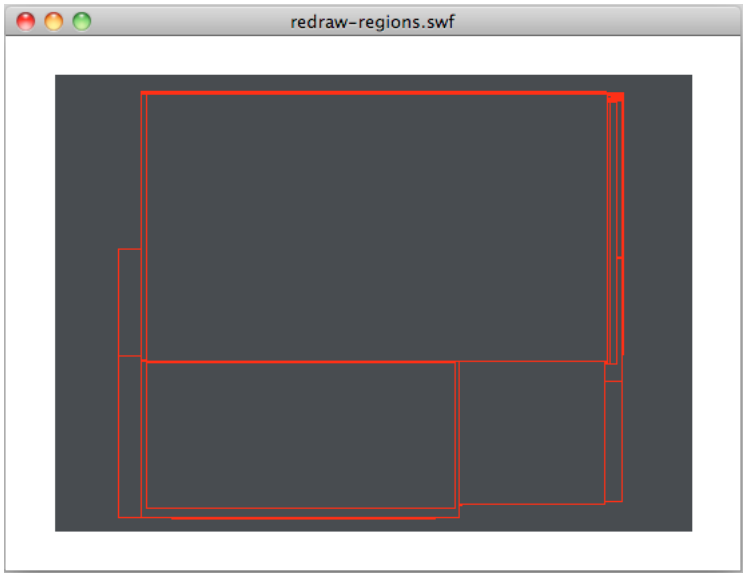
Opzione relativa alle aree di ridisegno attivata

Potete inoltre abilitare questa opzione a livello di codice utilizzando il metodo `flash.profiler.showRedrawRegions()`.

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

Nelle applicazioni Adobe AIR, questo metodo è il solo modo disponibile per abilitare l'opzione per il ridisegno delle aree.

Utilizzate le aree di ridisegno per identificare le opportunità di ottimizzazione. Tenete presente che anche se alcuni oggetti di visualizzazione non sono visibili, consumano comunque cicli della CPU perché sono ancora in fase di rendering. L'immagine seguente illustra questa idea. Una forma vettoriale nera copre il personaggio animato che corre. L'immagine mostra che l'oggetto di visualizzazione non è stato rimosso dall'elenco di visualizzazione ed è ancora in fase di rendering. Questa operazione utilizza cicli della CPU:



Aree ridisegnate

Per migliorare le prestazioni, impostate la proprietà `visible` del personaggio nascosto che corre su `false` o rimuovetelo completamente dall'elenco di visualizzazione e arrestate la relativa linea temporale Queste operazioni consentono di bloccare l'oggetto di visualizzazione, utilizzando di conseguenza una quantità minima di potenza della CPU.

È consigliabile utilizzare le aree di ridisegno durante l'intero ciclo di sviluppo. L'utilizzo di questa opzione evita di ritrovarsi alla fine del progetto aree di ridisegno non necessarie e aree di ottimizzazione che sono state omesse.

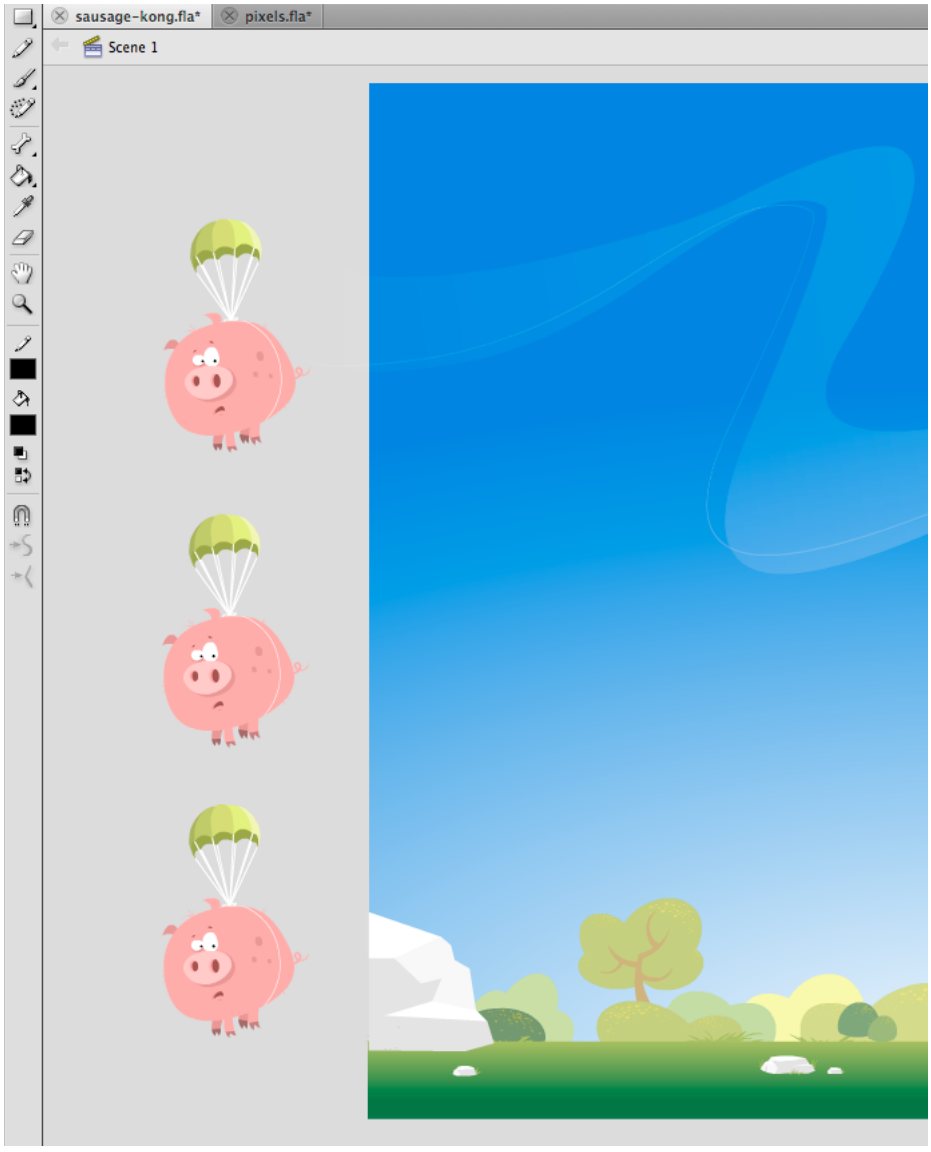
Altri argomenti presenti nell'Aiuto

"Blocco e sblocco degli oggetti" a pagina 28

Contenuto fuori dello stage

Evitate di inserire contenuto fuori dello stage. Al contrario, inserite semplicemente gli oggetti nell'elenco di visualizzazione quando necessario.

Se possibile, cercate di non inserire contenuto grafico fuori dallo stage. I designer e gli sviluppatori inseriscono solitamente degli elementi fuori dallo stage per riutilizzare le risorse per tutta la durata di esecuzione dell'applicazione. La figura seguente illustra questa tecnica comune:



Contenuto fuori dello stage

Anche se gli elementi fuori dallo stage non vengono visualizzati sullo schermo e non ne viene eseguito il rendering, sono ancora presenti nell'elenco di visualizzazione. Il runtime continua a eseguire test interni su tali elementi per verificare che siano ancora fuori dello stage e che non vi sia alcuna interazione da parte dell'utente. Evitate quindi, per quanto possibile, di inserire oggetti fuori dello stage e rimuoveteli invece dall'elenco di visualizzazione.

Qualità dei filmati

Utilizzate l'impostazione di qualità dello stage appropriata per migliorare il rendering.

Quando si sviluppa contenuto per i dispositivi mobili con schermi di piccole dimensioni, ad esempio i telefoni cellulari, la qualità delle immagini è meno importante rispetto a quando si sviluppa per le applicazioni desktop. L'impostazione del valore più appropriato per la qualità dello stage può migliorare le prestazioni di rendering.

Le impostazioni seguenti sono disponibili per la qualità dello stage:

- `StageQuality.LOW`: privilegia la velocità di riproduzione rispetto alla qualità e non utilizza l'antialiasing. Questa impostazione non è supportata in Adobe AIR per desktop o TV.
- `StageQuality.MEDIUM`: applica l'antialiasing, ma non smussa le bitmap modificate in scala. Questa impostazione è il valore predefinito per AIR sui dispositivi mobili, ma non è supportata in AIR per desktop o TV.
- `StageQuality.HIGH`: (impostazione predefinita sul desktop) privilegia la qualità rispetto alla velocità di riproduzione e applica sempre l'antialiasing. Se il file SWF contiene animazioni, le bitmap modificate in scala non vengono smussate; in caso contrario, le bitmap vengono smussate.
- `StageQuality.BEST`: offre la migliore qualità di visualizzazione e non considera la velocità di riproduzione. Tutto l'output è sottoposto ad antialiasing e le bitmap modificate in scala vengono sempre smussate.

L'uso di `StageQuality.MEDIUM` spesso assicura una qualità soddisfacente per le applicazioni su dispositivi mobili, e in alcuni casi anche `StageQuality.LOW` può essere sufficiente. A partire da Flash Player 8, il rendering del testo con antialiasing può essere eseguito con precisione anche con la qualità dello stage impostata su `LOW`.

Nota: *in alcuni dispositivi mobili, anche se la qualità è impostata su `HIGH`, viene utilizzato `MEDIUM` per ottenere prestazioni migliori nelle applicazioni Flash Player. Tuttavia, l'impostazione di qualità HIGH non produce una differenza degna di nota, poiché i display dei dispositivi mobili solitamente hanno un valore dpi più elevato. (Il valore dpi può variare da un dispositivo all'altro.)*

Nella figura seguente, la qualità del filmato è impostata su `MEDIUM` e il rendering del testo è impostato su Antialiasing per animazione:

Qualità dello stage MEDIUM e rendering del testo impostato su Antialiasing per animazione

L'impostazione della qualità dello stage influisce sulla qualità del testo perché l'impostazione di rendering del testo utilizzata non è quella più appropriata.

Il runtime consente di impostare il rendering del testo su Antialiasing per leggibilità, mantenendo una qualità perfetta del testo (con antialiasing) indipendentemente dall'impostazione di qualità utilizzata:

Qualità dello stage LOW e rendering del testo impostato su Antialiasing per leggibilità

La stessa qualità di rendering può essere ottenuta impostando il rendering del testo su Testo bitmap (senza antialiasing):

Qualità dello stage LOW e rendering del testo impostato su Testo bitmap (senza antialiasing)

Gli ultimi due esempi mostrano che si può produrre testo di alta qualità indipendentemente dall'impostazione di qualità utilizzata. Questa funzione è disponibile da Flash Player 8 e può essere utilizzata con i dispositivi mobili. Tenete presente che Flash Player 10.1 commuta automaticamente su `StageQuality.MEDIUM` su alcuni dispositivi per migliorare le prestazioni.

Fusione alfa

Evitate di usare la proprietà alpha, quando è possibile.

Evitare di impiegare gli effetti che richiedono la fusione alfa quando utilizzate la proprietà `alpha`, ad esempio gli effetti di dissolvenza. Quando un oggetto di visualizzazione usa la fusione alfa, per determinare il colore finale il runtime deve combinare i valori di colore di ogni oggetto di visualizzazione sovrapposto e il colore di sfondo. La fusione alfa può quindi richiedere una maggiore quantità di risorse del processore rispetto al disegno di un colore opaco. Questo calcolo supplementare può incidere negativamente sulle prestazioni nel caso di dispositivi lenti. Evitate di usare la proprietà `alpha`, quando è possibile.

Altri argomenti presenti nell'Aiuto

"Caching bitmap" a pagina 56

"Rendering di oggetti di testo" a pagina 69

Frequenza fotogrammi dell'applicazione

In generale, utilizzate la più bassa frequenza fotogrammi possibile per migliorare le prestazioni.

La frequenza fotogrammi di un'applicazione determina la quantità di tempo disponibile per ogni ciclo di "esecuzione del codice dell'applicazione e rendering", come descritto in "Elementi fondamentali dell'esecuzione del codice runtime" a pagina 1. Una frequenza fotogrammi più elevata crea un'animazione più fluida. Quando tuttavia non avvengono modifiche dell'animazione o di altri aspetti visivi, spesso non vi è motivo di utilizzare una frequenza fotogrammi più elevata. Una frequenza fotogrammi maggiore utilizza più cicli della CPU e consuma più energia rispetto a una frequenza inferiore.

Riportiamo di seguito alcune linee guida generali che potete seguire per impostare correttamente la frequenza fotogrammi predefinita della vostra applicazione:

- Se utilizzate il framework Flex, non modificate il valore predefinito della frequenza fotogrammi iniziale.
- Se l'applicazione comprende animazioni, una frequenza fotogrammi adeguata dovrebbe essere di almeno 20 fotogrammi al secondo. Spesso non è necessario impostare un valore superiore a 30 fotogrammi al secondo.
- Se l'applicazione non comprende animazioni, una frequenza di 12 fotogrammi al secondo sarà probabilmente sufficiente.

Il criterio della "più bassa frequenza fotogrammi possibile" può variare a seconda dell'attività corrente dell'applicazione. Per ulteriori informazioni vedete il suggerimento successivo, "Modificate dinamicamente la frequenza fotogrammi dell'applicazione".

Utilizzate una frequenza fotogrammi ridotta quando i contenuti video sono gli unici contenuti dinamici dell'applicazione.

Il runtime riproduce i contenuti video caricati con la frequenza fotogrammi nativa, indipendentemente dalla frequenza fotogrammi dell'applicazione. Se l'applicazione non contiene animazioni o altri contenuti visivi altamente dinamici, l'uso di una frequenza fotogrammi ridotta non incide negativamente sull'esperienza dell'utente.

Modificate dinamicamente la frequenza fotogrammi dell'applicazione.

La frequenza fotogrammi iniziale dell'applicazione viene definita nelle impostazioni del progetto o del compilatore, ma non si tratta di un valore fisso. Potete modificare la frequenza fotogrammi impostando la proprietà `Stage.frameRate` (o la proprietà `WindowedApplication.frameRate` in Flex).

Modificate la frequenza fotogrammi in base alle esigenze correnti dell'applicazione. Ad esempio, potete ridurre la frequenza fotogrammi quando l'applicazione non esegue alcuna animazione. Quando sta per iniziare una transizione animata, aumentate la frequenza fotogrammi. In modo analogo, se l'applicazione viene eseguita in background (dopo aver perso lo stato attivo), potete in genere ridurre ulteriormente la frequenza fotogrammi. È probabile che l'utente sia concentrato su un'altra applicazione o un'altra operazione.

Di seguito sono riportate alcune linee guida generali che potete utilizzare come punto di partenza per determinare la frequenza fotogrammi appropriata per vari tipi di attività:

- Se utilizzate il framework Flex, non modificate il valore predefinito della frequenza fotogrammi iniziale.

- Durante la riproduzione delle animazioni, impostate la frequenza fotogrammi su almeno 20 fotogrammi al secondo. Spesso non è necessario impostare un valore superiore a 30 fotogrammi al secondo.
- Quando l'applicazione non riproduce alcuna animazione, una frequenza di 12 fotogrammi al secondo sarà probabilmente sufficiente.
- I video caricati vengono riprodotti con la frequenza fotogrammi nativa, indipendentemente dalla frequenza fotogrammi dell'applicazione. Se i video sono gli unici contenuti dinamici dell'applicazione, una frequenza di 12 fotogrammi al secondo sarà probabilmente sufficiente.
- Quando l'applicazione non è attiva per l'input, una frequenza di 5 fotogrammi al secondo sarà probabilmente sufficiente.
- Quando un'applicazione AIR non è visibile, una frequenza massima di 2 fotogrammi al secondo sarà probabilmente sufficiente. Queste linee guida sono applicabili, ad esempio, se un'applicazione è ridotta a icona. È applicabile anche ai dispositivi desktop se la proprietà `visible` della finestra nativa è impostata su `false`.

Per le applicazioni create in Flex, la classe spark.components. WindowedApplication è dotata di supporto incorporato per la modifica dinamica della frequenza fotogrammi dell'applicazione. La proprietà `backgroundFrameRate` definisce la frequenza fotogrammi dell'applicazione quando l'applicazione non è attiva. Il valore predefinito è 1 e modifica la frequenza fotogrammi di un'applicazione creata con il framework Spak, impostandola su 1 fotogramma al secondo. Potete modificare la frequenza fotogrammi in background impostando la proprietà `backgroundFrameRate`. Potete impostare la proprietà su un altro valore oppure su -1 per disattivare la limitazione automatica della frequenza fotogrammi.

Per ulteriori informazioni sulla modifica dinamica della frequenza fotogrammi di un'applicazione, vedete i seguenti articoli:

- Articolo e codice di esempio pubblicati da Jonnie Hallman nel Centro per sviluppatori Adobe: Reducing CPU usage in Adobe AIR (Riduzione dell'uso della CPU in Adobe AIR)
- Articolo e applicazione di esempio pubblicati da Arno Gourdol: Writing well-behaved, efficient, AIR applications (Come scrivere applicazioni AIR efficienti e ad alte prestazioni)

Grant Skinner ha creato una classe throttler per la frequenza fotogrammi. Potete utilizzare questa classe nelle vostre applicazioni per ridurre automaticamente la frequenza fotogrammi quando l'applicazione è in esecuzione in background. Per ulteriori informazioni e per scaricare il codice sorgente per la classe FramerateThrottler, vedete l'articolo di Skinner relativo all'utilizzo della CPU inattiva in Adobe AIR e Flash Player all'indirizzo http://gskinner.com/blog/archives/2009/05/idle_cpu_usage.html.

Frequenza fotogrammi adattata

Quando compilate un file SWF, dovete impostare una frequenza di fotogrammi specifica per il filmato. In un ambiente soggetto a limitazioni e con una bassa frequenza della CPU, la frequenza dei fotogrammi a volte si abbassa durante la riproduzione. Per assicurare all'utente una frequenza di fotogrammi accettabile, il runtime salta il rendering di alcuni fotogrammi. In questo modo si evita di abbassare la frequenza sotto un valore minimo accettabile.

Nota: *in questo caso, il runtime non salta dei fotogrammi, ma solo il rendering del contenuto dei fotogrammi. Il codice viene comunque eseguito e l'elenco di visualizzazione viene aggiornato, solo che gli aggiornamenti non appaiono sullo schermo. Non esiste un modo per specificare un valore soglia in f/s (fotogrammi al secondo) al fine di indicare quanti fotogrammi saltare se il runtime non è in grado di mantenere stabile la frequenza dei fotogrammi.*

Caching bitmap

Utilizzate la funzione di caching bitmap per il contenuto vettoriale complesso, nei casi appropriati.

La funzione di caching bitmap consente di ottenere una buona ottimizzazione. Questa funzione provvede a memorizzare un oggetto vettore nella cache, ne esegue il rendering come bitmap internamente e utilizza la bitmap così ottenuta per il rendering. Il risultato può essere un miglioramento significativo delle prestazioni di rendering, che tuttavia può avvenire a spese della quantità di memoria utilizzata. Utilizzate la funzione di caching bitmap per il contenuto vettoriale complesso, ad esempio gradienti o testi sofisticati.

L'attivazione del caching bitmap per un oggetto animato che contiene grafica vettoriale complessa (ad esempio testo o gradienti) consente di migliorare le prestazioni. Se tuttavia è attivata la memorizzazione delle bitmap nella cache per un oggetto di visualizzazione, quale un clip filmato con la linea temporale in esecuzione, otterrete il risultato opposto. Per ogni fotogramma il runtime deve aggiornare la bitmap nella cache e ridisegnarla sullo schermo, richiedendo molti cicli della CPU. La funzione di caching bitmap è vantaggiosa solo quando la bitmap nella cache può essere generata una sola volta e quindi utilizzata senza che sia necessario aggiornarla.

Se attivate il caching bitmap per un oggetto Sprite, quest'ultimo può essere spostato senza che il runtime debba rigenerare la bitmap nella cache. La modifica delle proprietà `x` e `y` dell'oggetto non ne causa la rigenerazione. Tuttavia, qualsiasi tentativo di ruotare o ridimensionare l'oggetto, oppure di modificarne il valore alfa, fa sì che il runtime rigeneri la bitmap nella cache peggiorando, di conseguenza, le prestazioni.

***Nota:** la proprietà `DisplayObject.cacheAsBitmapMatrix` disponibile in AIR e in Packager per iPhone non è soggetta a questa limitazione. Utilizzando la proprietà `cacheAsBitmapMatrix` potete ruotare, modificare in scala, inclinare e cambiare il valore alfa di un oggetto di visualizzazione senza attivare la rigenerazione della bitmap.*

Una bitmap memorizzata nella cache può utilizzare più memoria di una normale istanza di clip filmato. Per esempio, se il clip filmato sullo stage misura 250 x 250 pixel, esso utilizza circa 250 KB se è memorizzato nella cache oppure solo 1 KB senza il caching.

L'esempio seguente è relativo a un oggetto Sprite che contiene l'immagine di una mela. La classe seguente è associata al simbolo della mela:

```
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);
        }
```

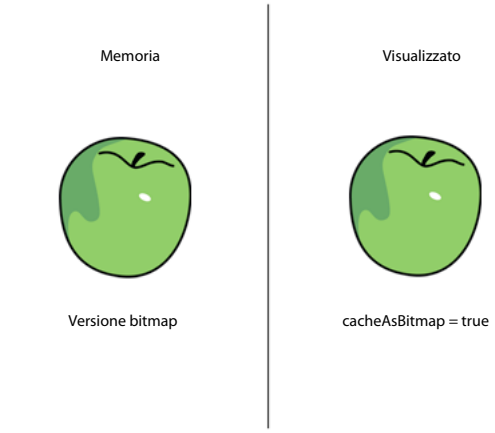
```
        private function deactivation(e:Event):void
        {
            removeEventListener(Event.ENTER_FRAME,handleMovement);
        }

        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();
        }
    }
}
```

Il codice utilizza la classe Sprite anziché la classe MovieClip perché non è necessaria una linea temporale per ciascuna mela. Per ottenere prestazioni ottimali, usate oggetti il più possibile "leggeri". Quindi, viene creata un'istanza della classe con il codice seguente:

```
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);
        }
    }
}
```

Quando l'utente fa clic sul mouse, le mele vengono create senza memorizzazione nella cache. Quando l'utente preme il tasto C (codice tasto 67), i vettori delle mele vengono memorizzati nella cache come bitmap e visualizzati sullo schermo. Questa tecnica migliora sensibilmente le prestazioni di rendering, sia sul desktop che sui dispositivi mobili, quando la CPU è lenta.

D'altra parte, benché la funzione di caching bitmap garantisca prestazioni di rendering migliori, essa può anche consumare rapidamente grandi quantità di memoria. Non appena un oggetto viene memorizzato nella cache, la sua superficie viene catturata come bitmap trasparente e memorizzata in memoria, come mostra il diagramma seguente:

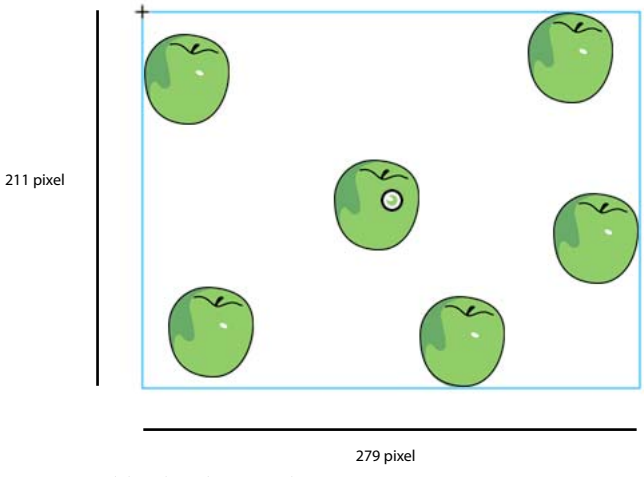


L'oggetto e la sua bitmap di superficie memorizzati in memoria

Flash Player 10.1 e AIR 2.5 ottimizzano l'uso della memoria adottando lo stesso approccio descritto in "Filtri e scaricamento dinamico delle bitmap" a pagina 20. Se un oggetto di visualizzazione memorizzato nella cache è nascosto oppure fuori schermo, la relativa bitmap in memoria viene liberata se non viene utilizzata per un certo intervallo di tempo.

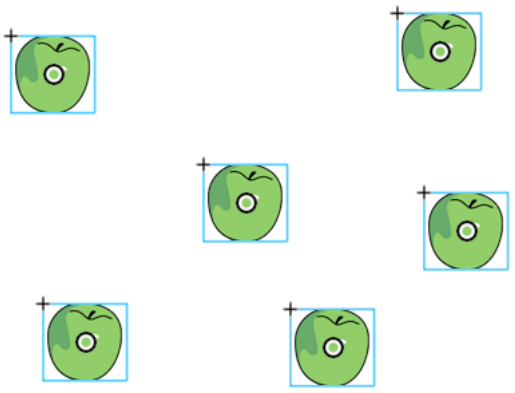
Nota: *se la proprietà* `opaqueBackground` *dell'oggetto di visualizzazione è impostata su un colore, specifico, il runtime considera l'oggetto di visualizzazione come opaco. Quando è utilizzata con la proprietà* `cacheAsBitmap`*, il runtime crea in memoria una bitmap non trasparente a 32 bit. Il canale alfa viene impostato su 0xFF, per migliorare le prestazioni, dal momento che non è richiesta alcuna trasparenza per disegnare la bitmap sullo schermo. Evitando di utilizzare la fusione alfa, si ottiene un rendering ancora più veloce. Se la profondità corrente dello schermo è limitata a 16 bit, la bitmap in memoria viene memorizzata come immagine a 16 bit. L'uso della proprietà* `opaqueBackground` *non attiva implicitamente il caching bitmap.*

Per risparmiare memoria, utilizzate la proprietà `cacheAsBitmap` e attivatela per ogni oggetto di visualizzazione diverso dal contenitore. L'attivazione del caching bitmap sul contenitore produrrebbe una bitmap finale molto più grande in memoria, creando una bitmap trasparente con dimensioni di 211 x 279 pixel. L'immagine utilizza circa 229 KB di memoria:



Attivazione del caching bitmap sul contenitore

Inoltre, memorizzando il contenitore nella cache rischiate di aggiornare l'intera bitmap in memoria, se una mela qualsiasi inizia a muoversi su un fotogramma. L'attivazione del caching bitmap sulle singole istanze determina la memorizzazione nella cache di sei superfici di 7 KB, con un consumo totale di soli 42 KB di memoria:



Attivazione del caching bitmap sulle istanze

Con l'accesso a ogni istanza della mela tramite l'elenco di visualizzazione e la chiamata del metodo `getChildAt()`, i riferimenti vengono memorizzati in un oggetto Vector per consentire un accesso più facile:

```
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);
        }
    }
}
```

Tenete presente che la funzione di caching bitmap migliora il rendering se il contenuto memorizzato nella cache non è ruotato o ridimensionato, né modificato su ciascun fotogramma. Al contrario, in caso di trasformazioni diverse dalla traslazione sugli assi x e y, il rendering non viene migliorato. In questi casi, Flash Player aggiorna la copia della bitmap nella cache per ogni trasformazione applicata all'oggetto di visualizzazione. L'aggiornamento della copia nella cache può determinare un uso intensivo della CPU, prestazioni rallentate e un consumo elevato della batteria. Di nuovo, la proprietà `cacheAsBitmapMatrix` in AIR o in Packager per iPhone non è soggetta a questa limitazione.

Il codice seguente modifica il valore alfa del metodo di movimento, cambiando l'opacità della mela su ciascun fotogramma:

```
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();
}
```

L'uso del caching bitmap causa un rallentamento delle prestazioni. La modifica del valore alfa obbliga il runtime ad aggiornare la bitmap nella cache a ogni modifica del valore.

I filtri fanno riferimento alle bitmap che vengono aggiornate ogni volta che l'indicatore di riproduzione di un clip filmato nella cache si sposta. Di conseguenza, l'uso di un filtro imposta automaticamente la proprietà `cacheAsBitmap` su `true`. La figura seguente illustra un clip filmato animato:

Clip filmato animato

Evitate di utilizzare i filtri sui contenuti animati, perché possono causare problemi di prestazioni. Nella figura seguente, il designer ha aggiunto un filtro ombra esterna:

Clip filmato animato con filtro Ombra esterna

Di conseguenza, se la linea temporale di un clip filmato è in corso di riproduzione, la bitmap deve essere rigenerata. Anche se il contenuto viene modificato in qualsiasi modo che non sia una semplice trasformazione x o y, la bitmap deve essere rigenerata. Per ogni fotogramma il runtime deve ridisegnare la bitmap, con il risultato che la CPU è sottoposta a un carico di lavoro maggiore, le prestazioni peggiorano e l'autonomia della batteria diminuisce.

Nelle seguenti esercitazioni video, PaulTrani fornisce degli esempi che mostrano come usare Flash Professional e ActionScript per ottimizzare la grafica utilizzando le bitmap:

- Optimizing Graphics (Ottimizzazione della grafica)
- Optimizing Graphics with ActionScript (Ottimizzazione della grafica con ActionScript)

Matrici di trasformazione delle bitmap memorizzate nella cache in AIR

Quando utilizzate bitmap memorizzate nella cache nelle applicazioni mobili AIR, impostate la proprietà `cacheAsBitmapMatrix`.

Nel profilo mobile AIR, potete assegnare un oggetto Matrix alla proprietà `cacheAsBitmapMatrix` di un oggetto di visualizzazione. Impostando questa proprietà, potete applicare qualsiasi trasformazione bidimensionale all'oggetto, senza rigenerare la bitmap memorizzata nella cache. Potete anche modificare la proprietà alpha senza rigenerare la bitmap nella cache. È inoltre necessario che la proprietà `cacheAsBitmap` sia impostata su `true` e che per l'oggetto non sia impostata alcuna proprietà 3D.

Impostando la proprietà `cacheAsBitmapMatrix`, la bitmap memorizzata nella cache viene generata anche se l'oggetto di visualizzazione è fuori dallo schermo, nascosto o ha la proprietà `visible` impostata su `false`. Anche la reimpostazione della proprietà `cacheAsBitmapMatrix` tramite un oggetto matrix contenente una trasformazione diversa rigenera la bitmap nella cache.

La trasformazione di matrice applicata alla proprietà `cacheAsBitmapMatrix` viene estesa all'oggetto di visualizzazione durante il rendering nella bitmap memorizzata nella cache. Di conseguenza, se la trasformazione contiene una scala 2x, il rendering della bitmap sarà il doppio delle dimensioni del rendering vettoriale. Il renderer applica la trasformazione inversa alla bitmap nella cache, in modo che la visualizzazione finale abbia lo stesso aspetto. Potete modificare in scala la bitmap memorizzata nella cache in modo da ridurre le dimensioni e l'uso della memoria, a scapito della fedeltà di rendering. Potete anche modificare in scala la bitmap memorizzata nella cache in modo da aumentare le dimensioni e, in alcuni casi, la qualità di rendering, a scapito dell'uso della memoria. In generale, utilizzate comunque una matrice di identità, ovvero una matrice che non applica alcuna trasformazione, per evitare alterazioni dell'aspetto, come illustrato nell'esempio seguente:

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

Dopo avere impostato la proprietà `cacheAsBitmapMatrix`, potete modificare in scala, inclinare, ruotare e convertire l'oggetto senza attivare la rigenerazione della bitmap.

Potete inoltre modificare il valore alfa nell'intervallo compreso tra 0 e 1. Se modificate il valore alfa tramite la proprietà `transform.colorTransform` con una trasformazione di colore, il valore alfa utilizzato nell'oggetto di trasformazione deve essere compreso tra 0 e 255. La modifica della trasformazione di colore in qualsiasi altro modo comporta la rigenerazione della bitmap memorizzata nella cache.

Impostate sempre la proprietà `cacheAsBitmapMatrix` quando impostate `cacheAsBitmap` su `true` nei contenuti creati per i dispositivi mobili. Consideratene tuttavia i potenziali svantaggi descritti di seguito. Dopo che un oggetto è stato ruotato, modificato in scala o inclinato, è possibile che il rendering finale presenti artefatti a livello di ridimensionamento o aliasing della bitmap rispetto a un normale rendering vettoriale.

Caching manuale delle bitmap

Utilizzate la classe BitmapData per creare comportamenti di caching bitmap personalizzati.

L'esempio seguente riutilizza una singola versione bitmap rasterizzata di un oggetto di visualizzazione e fa riferimento allo stesso oggetto BitmapData. Quando viene ridimensionato ciascun oggetto di visualizzazione, l'oggetto BitmapData originale presente in memoria non viene aggiornato né ridisegnato. Questo approccio consente di risparmiare risorse della CPU e di eseguire più rapidamente le applicazioni. Quando si ridimensiona l'oggetto di visualizzazione, la bitmap contenuta viene allungata.

Questa è la classe `BitmapApple` aggiornata:

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

Il valore alfa viene comunque modificato su ciascun fotogramma. Il codice seguente passa il buffer sorgente originale a ogni istanza BitmapApple:

```
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);
}
```

Questa tecnica consuma solo una piccola quantità di memoria, poiché solo una singola bitmap memorizzata nella cache viene utilizzata nella memoria e condivisa da tutte le istanze `BitmapApple`. Inoltre, nonostante tutte le modifiche apportate alle istanze `BitmapApple` (ad esempio la modifica alfa, la rotazione o il ridimensionamento), la bitmap sorgente originale non viene mai aggiornata. L'uso di questa tecnica evita un possibile rallentamento delle prestazioni.

Per ottenere una bitmap finale fluida, impostate la proprietà `smoothing` su `true`:

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

    smoothing = true;

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

Anche la regolazione della qualità dello stage può migliorare le prestazioni. Impostate la qualità dello stage su `HIGH` prima della rasterizzazione, quindi passate a `LOW` in seguito:

```
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 );
}
```

La regolazione della qualità dello stage prima e dopo della rasterizzazione di un vettore in bitmap può rappresentare una tecnica sofisticata per la visualizzazione del contenuto con antialiasing sullo schermo. Questa tecnica può rivelarsi efficace indipendentemente dalla qualità finale dello stage. Ad esempio, potete ottenere una bitmap con antialiasing con testo con antialiasing anche con la qualità dello stage impostata su `LOW`. Questa tecnica non è utilizzabile con la proprietà `cacheAsBitmap`. Con tale proprietà, l'impostazione della qualità dello stage su `LOW` determina un aggiornamento della qualità del vettore, che a sua volta produce un aggiornamento delle superfici bitmap in memoria e infine un aggiornamento della qualità finale.

Isolamento dei comportamenti

Isolate gli eventi come `Event.ENTER_FRAME` in un singolo gestore, quando è possibile.

Il codice può essere ulteriormente ottimizzato isolando l'evento `Event.ENTER_FRAME` della classe `Apple` in un singolo gestore. Questa tecnica consente di risparmiare risorse della CPU. L'esempio seguente illustra questo diverso approccio, in cui la classe BitmapApple non gestisce più il comportamento di movimento:

```
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;
        }
}
```

Il codice seguente crea le istanze delle mele e ne gestisce il movimento in un unico gestore:

```
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;
            }
        }
}
```

Il risultato è un unico evento `Event.ENTER_FRAME` che gestisce il movimento, anziché 200 gestori che spostano ciascuno una mela. L'animazione può essere messa in pausa facilmente, il che può essere utile in un videogioco.

Ad esempio, un gioco semplice può utilizzare il seguente gestore:

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

Il passaggio successivo consiste nell'abilitare l'interazione delle mele con il mouse o la tastiera e richiede delle modifiche della classe `BitmapApple`.

```
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);
        }
}
```

Il risultato è una serie di istanze `BitmapApple` interattive, come oggetti Sprite tradizionali. Tuttavia, le istanze sono collegate a una singola bitmap, che non viene ricampionata a seguito della trasformazione degli oggetti di visualizzazione.

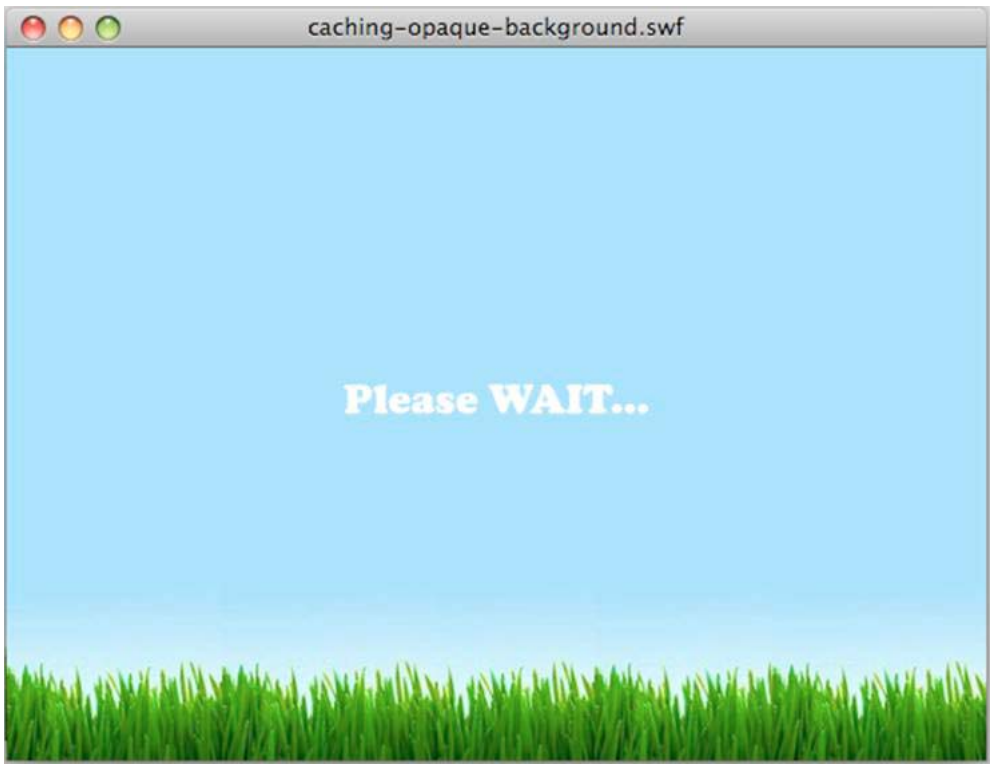
Rendering di oggetti di testo

Utilizzate la funzione di caching bitmap e la proprietà `opaqueBackground` per migliorare le prestazioni di rendering del testo.

Flash Text Engine offre alcune interessanti ottimizzazioni. Tuttavia, per visualizzare una singola riga di testo sono necessarie diverse classi. Per questo motivo, la creazione di un campo di testo modificabile con la classe TextLine richiede una notevole quantità di memoria e molte righe di codice ActionScript. La classe TextLine è indicata soprattutto per il testo statico e non modificabile, poiché per tale tipo di testo garantisce un rendering veloce e consuma meno memoria.

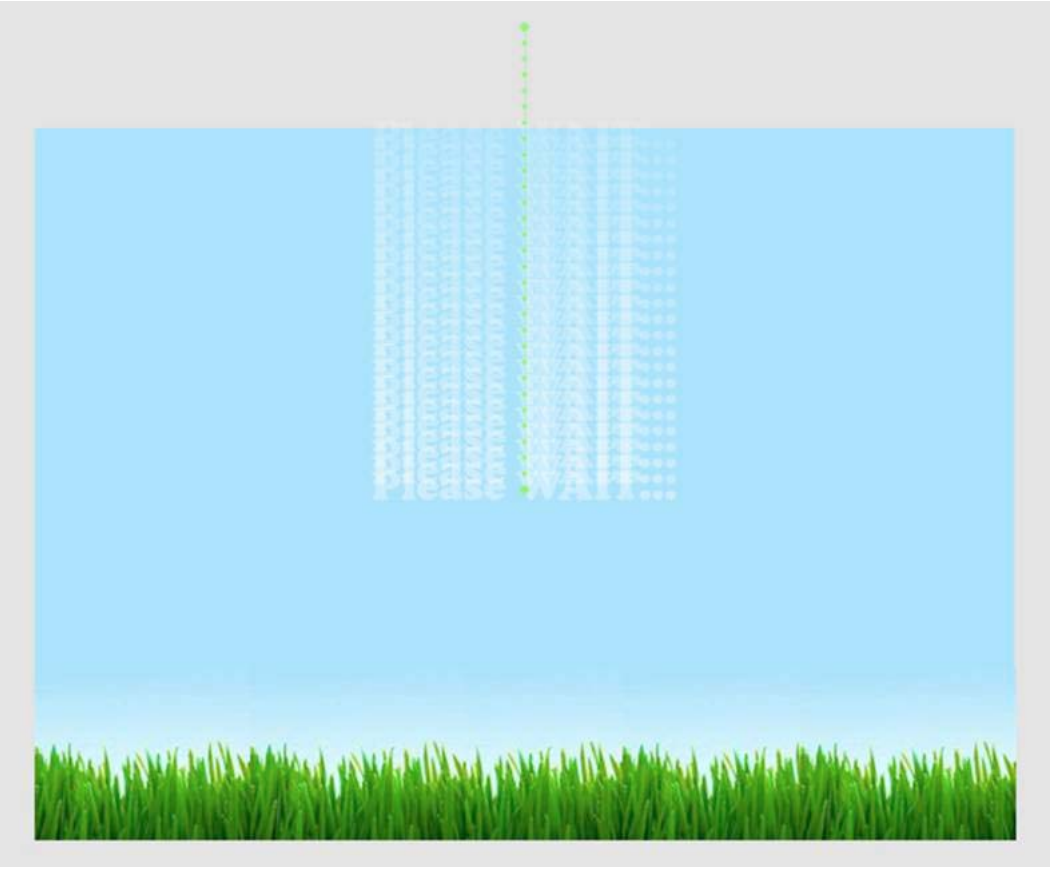
La funzione di caching bitmap consente di memorizzare nella cache il contenuto vettoriale sotto forma di bitmap per migliorare le prestazioni di rendering. Questa funzione risulta utile per il contenuto vettoriale complesso così come per il contenuto testuale il cui rendering richiede una certa quantità di elaborazione.

L'esempio seguente illustra come utilizzare la funzione di caching bitmap e la proprietà `opaqueBackground` per migliorare le prestazioni di rendering. La figura seguente mostra una tipica schermata di attesa che può essere visualizzata quando un utente sta aspettando che qualcosa venga caricato:



Schermata di attesa

La figura seguente illustra l'andamento che viene applicato programmaticamente all'oggetto TextField. L'andamento del testo procede lentamente dall'alto della scena fino al centro:



Andamento del testo

Il codice seguente crea l'andamento. La variabile `preloader` memorizza l'oggetto target corrente per ridurre al minimo le verifiche delle proprietà, che possono pregiudicare le prestazioni:

```
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 );
}
```

La funzione `Math.abs()` in questo caso potrebbe essere spostata inline per ridurre il numero di chiamate di funzione e ottenere un miglioramento delle proprietà. È buona norma utilizzare il tipo int per le proprietà `destX` e `destY`, in modo da ottenere valori a virgola fissa. Utilizzando il tipo int si ottiene un aggancio ai pixel perfetto e non occorre arrotondare manualmente i valori mediante metodi lenti come `Math.ceil()` o `Math.round()`. Questo codice non arrotonda le coordinate a un numero intero, poiché un arrotondamento costante dei valori non permetterebbe all'oggetto di muoversi in maniera fluida. L'oggetto potrebbe muoversi in modo irregolare perché le coordinate verrebbero adattate ai numeri interi più vicini per ciascun fotogramma. Tuttavia, questa tecnica può essere utile quando si imposta una posizione finale per un oggetto di visualizzazione. Non utilizzate il codice seguente:

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

Il codice seguente è molto più veloce:

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

Il codice precedente può essere ottimizzato ulteriormente utilizzando operatori di spostamento bit a bit per dividere i valori:

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

La funzione di caching bitmap facilita il rendering degli oggetti tramite il runtime utilizzando le bitmap dinamiche. Nell'esempio corrente, il clip filmato che contiene l'oggetto TextField viene memorizzato nella cache:

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

Un ulteriore modo per migliorare le prestazioni è quello di rimuovere la trasparenza alfa. La trasparenza alfa comporta un carico di lavoro aggiuntivo per il runtime quando devono essere disegnate immagini bitmap trasparenti, come nel codice precedente. Potete utilizzare la proprietà `opaqueBackground` per aggirare il problema, specificando un colore per lo sfondo.

Quando si usa la proprietà `opaqueBackground`, la superficie bitmap creata in memoria utilizza comunque 32 bit, ma l'offset alfa è impostato su 255 e la trasparenza non viene utilizzata. Di conseguenza, la proprietà `opaqueBackground` non riduce l'utilizzo della memoria ma migliora le prestazioni di rendering quando si usa la funzione di caching bitmap. Il codice seguente contiene tutte le ottimizzazioni:

```
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 );
}
```

L'animazione è ora ottimizzata e il caching bitmap è stato ottimizzato rimuovendo la trasparenza. Sui dispositivi mobili, valutate anche la possibilità di commutare la qualità dello stage tra `LOW` e `HIGH` durante i diversi stati dell'animazione utilizzando la funzione di caching bitmap:

```
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 );
    }
}
```

Tuttavia, in questo caso, la modifica della qualità dello stage obbliga il runtime a rigenerare la superficie bitmap dell'oggetto TextField per farla corrispondere alla qualità corrente dello stage. Per questo motivo, è meglio non cambiare la qualità dello stage quando si usa la funzione di caching bitmap.

In questo caso si potrebbe adottare una soluzione di caching bitmap manuale. Per simulare la proprietà `opaqueBackground`, il clip filmato può essere disegnato su un oggetto BitmapData non trasparente, così da non obbligare il runtime a rigenerare la superficie bitmap.

Questa tecnica funziona bene per i contenuti non soggetti a variazioni nel corso del tempo. Se invece è possibile che il contenuto del campo di testo subisca dei cambiamenti, valutate una strategia differente. Ad esempio, considerate un campo di testo che viene continuamente aggiornato con una percentuale che indica il grado di caricamento dell'applicazione. Se il campo di testo, o l'oggetto di visualizzazione che lo contiene, è stato memorizzato nella cache come bitmap, la sua superficie deve essere generata dopo ogni cambiamento del contenuto. Non potete usare il caching bitmap manuale in questo caso, perché l'oggetto di visualizzazione utilizzato come contenitore è soggetto a continui cambiamenti. Questo cambiamento costante vi obbligherebbe a chiamare manualmente il metodo `BitmapData.draw()` per aggiornare la bitmap memorizzata nella cache.

Tenete presente che a partire da Flash Player 8 (e AIR 1.0), indipendentemente dal valore di qualità dello stage, un campo di testo con il rendering impostato su Antialiasing per leggibilità mantiene un perfetto antialiasing. Questa soluzione consuma meno memoria, ma richiede più risorse della CPU e determina un rendering più lento rispetto alla funzione di caching bitmap.

Il codice seguente usa questo tipo di soluzione:

```
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 );
    }
}
```

L'uso di questa opzione (Antialiasing per leggibilità) non è consigliato per il testo in movimento. Quando si ridimensiona il testo, l'opzione tenta di mantenere allineato il testo, causando un effetto di spostamento. Se tuttavia il contenuto dell'oggetto di visualizzazione cambia in continuazione ed è necessario ridimensionare il testo, potete migliorare le prestazioni nelle applicazioni mobili impostando la qualità su `LOW`. Quando il movimento finisce, riportate la qualità a `HIGH`.

GPU

Rendering tramite GPU nelle applicazioni Flash Player

Un'importante nuova funzione di Flash Player 10.1 è la possibilità di utilizzare la GPU per eseguire il rendering del contenuto grafico sui dispositivi mobili. Nel passato, il rendering della grafica veniva eseguito solo mediante la CPU. L'uso della GPU permette di ottimizzare il rendering di filtri, bitmap, video e testo. Tenete presente che il rendering tramite GPU non è sempre preciso come quello eseguito dal software. Il contenuto può apparire leggermente frammentato quando si usa il renderer hardware. Inoltre, Flash Player 10.1 presenta una limitazione che può impedire il rendering degli effetti Pixel Bender sullo schermo. Questi effetti potrebbero essere riprodotti come quadrati neri quando si usa l'accelerazione hardware.

Anche se Flash Player 10 presentava già una funzione di accelerazione tramite GPU, questa non veniva utilizzata per calcolare la grafica, bensì solo per inviare tutta la grafica allo schermo. In Flash Player 10.1, la GPU viene utilizzata anche per calcolare la grafica, con un possibile aumento significativo delle prestazioni. Viene inoltre ridotto il carico di lavoro della CPU, un aspetto molto utile per i dispositivi con risorse limitate come quelli mobili.

La modalità GPU viene impostata automaticamente quando si esegue il contenuto su dispositivi mobili, per ottenere le migliori prestazioni possibili. Anche se `wmode` non deve più essere impostato su `gpu` per abilitare il rendering tramite GPU, l'impostazione di `wmode` su `opaque` o `transparent` disattiva l'accelerazione tramite GPU.

Nota: *Flash Player sul desktop utilizza tuttora la CPU per il rendering software. Il rendering software viene utilizzato perché i driver, che possono accentuare le differenze di rendering, variano moltissimo in ambiente desktop. Possono verificarsi differenze di rendering anche tra i dispositivi desktop e alcuni dispositivi mobili.*

Rendering tramite GPU nelle applicazioni AIR mobili

Potete abilitare l'accelerazione grafica hardware in un'applicazione AIR includendo `<renderMode>gpu</renderMode>` nel descrittore dell'applicazione. Le modalità di rendering non possono essere modificate in fase di runtime. Sui computer desktop, l'impostazione `renderMode` viene ignorata; l'accelerazione grafica tramite GPU non è attualmente supportata.

Limitazioni della modalità di rendering tramite GPU

L'uso della modalità di rendering tramite GPU in AIR 2.5 presenta le seguenti limitazioni:

- Un oggetto che non può essere renderizzato tramite GPU non viene visualizzato del tutto. Non esiste alcun metodo di riserva per il rendering tramite CPU.
- I seguenti metodi di fusione non sono supportati: Livello, Alfa, Cancella, Sovrapponi, Luce intensa, Schiarisci e Scurisci.
- I filtri non sono supportati.
- PixelBender non è supportato.
- Molte GPU hanno dimensioni massime delle texture di 1024x1024. In ActionScript, si traduce nelle dimensioni massime finali di un oggetto di visualizzazione sottoposto a rendering, dopo tutte le trasformazioni.
- Adobe non consiglia di utilizzare la modalità di rendering tramite GPU nelle applicazioni AIR che riproducono video.
- Nella modalità di rendering tramite GPU, i campi di testo non vengono sempre spostati in un'area visibile quando si apre la tastiera virtuale. Per avere la certezza che i campi di testo siano visibili mentre l'utente immette il testo, effettuate una delle seguenti operazioni. Posizionate il capo di testo nella metà superiore dello schermo o spostatelo nella metà superiore dello schermo quando viene attivato.
- La modalità di rendering tramite GPU è disabilitata per alcuni dispositivi nei quali non funziona correttamente. Per informazioni aggiornate, vedete le note sulla versione per gli sviluppatori AIR.

Procedure ottimali relative alla modalità di rendering tramite GPU

Le seguenti linee guida possono velocizzare il rendering tramite GPU:

- Limitate il numero di elementi visibili sullo stage. Il rendering e la composizione di ciascun elemento richiedono tempo quando intorno sono presenti altri elementi. Quando non desiderate più vedere un oggetto di visualizzazione, impostate la relativa proprietà `visible` su `false`. Evitate di spostarlo semplicemente fuori dallo stage, nasconderlo dietro un altro oggetto o impostare la relativa proprietà `alpha` su 0. Se un oggetto di visualizzazione non è più necessario, potete rimuoverlo definitivamente dallo stage tramite `removeChild()`.
- Riutilizzate gli oggetti, anziché crearli ed eliminarli.
- Create bitmap le cui dimensioni siano prossime, ma non inferiori, a 2^n x 2^m bit. Le dimensioni devono essere prossime alla potenza di 2, senza che siano esattamente uguali o superiori. Ad esempio, il rendering di un'immagine 31 x 15 pixel viene eseguito più rapidamente rispetto a un'immagine 33 x 17 pixel. (31 e 15 sono appena inferiori alle potenze di 2: 32 e 16.)
- Se possibile, impostate il parametro repeat su false durante la chiamata del metodo `Graphic.beginBitmapFill()`.
- Non eccedete nel disegno. Utilizzate il colore di sfondo come sfondo. Non sovrapponete forme di grandi dimensioni una sopra l'altra. Ad ogni pixel da disegnare è associato un costo,
- Evitate forme con lunghe punte sottili, bordi che si auto intersecano o molti dettagli fini lungo i bordi. Il rendering di queste forme richiede più tempo rispetto a quello degli oggetti di visualizzazione con bordi arrotondati.
- Limitate le dimensioni degli oggetti di visualizzazione.

- Abilitate cacheAsBitMap e cacheAsBitmapMatrix per gli oggetti di visualizzazione contenenti grafica che non viene aggiornata di frequente.
- Evitate di utilizzare le API di disegno ActionScript (la classe Graphics) per creare grafici. Se possibile, create invece tali oggetti in modo statico in fase di authoring.
- Modificate in scala le risorse bitmap alle dimensioni finali prima di importarle.

Rendering tramite GPU nelle applicazioni AIR 2.0.3 mobili

Il rendering tramite GPU è più restrittivo nelle applicazioni AIR mobili create con Packager per iPhone. La GPU è efficace solo per le bitmap, le forme solide e gli oggetti di visualizzazione per i quali è impostata la proprietà `cacheAsBitmap`. Inoltre, per gli oggetti per i quali sono impostate le proprietà `cacheAsBitmap` e `cacheAsBitmapMatrix`, la GPU può eseguire in modo efficace il rendering degli oggetti che vengono ruotati o ridimensionati. La GPU viene utilizzata insieme ad altri oggetti di visualizzazione, fornendo in genere prestazioni di rendering scadenti.

Suggerimenti per ottimizzare le prestazioni del rendering tramite GPU

Anche se il rendering GPU può migliorare sensibilmente le prestazioni del contenuto SWF, la progettazione del contenuto gioca un ruolo fondamentale. Tenete presente che le impostazioni che hanno dato buoni risultati nel rendering software talvolta non funzionano bene con il rendering GPU. I suggerimenti che seguono possono aiutare a ottenere buone prestazioni nel rendering GPU senza pregiudicare quelle del rendering software.

***Nota:** i dispositivi mobili che supportano il rendering hardware accedono spesso al contenuto SWF dal web. Pertanto è importante tenere presenti questi suggerimenti quando si crea contenuto SWF, per garantire un'esperienza ottimale su tutti i tipi di schermi.*

- Evitate di usare `wmode=transparent` o `wmode=opaque` nei parametri HTML embed. Queste modalità possono determinare un calo delle prestazioni, nonché una piccola perdita nella sincronizzazione audio-video nel rendering sia software che hardware. Inoltre, molte piattaforme non supportano il rendering GPU quando queste modalità sono attive, con un conseguente calo significativo delle prestazioni.
- Utilizzate solo i metodi di fusione Normale e Alfa. Evitate di usare altri metodi di fusione, in particolare il metodo Livello. Non tutti i metodi di fusione possono essere riprodotti fedelmente con il rendering GPU.
- Quando una GPU esegue il rendering di un'immagine vettoriale, prima di disegnarla la suddivide in strutture composte da piccoli triangoli. Questo processo viene definito "tassellatura". La tassellatura determina un lieve calo delle prestazioni, che cresce man mano che la forma diventa più complessa. Per ridurre al minimo l'impatto delle prestazioni, evitate le forme di morphing, per le quali il rendering GPU esegue la tassellatura su ogni fotogramma.
- Evitate di usare curve che si auto-intersecano, aree curve molto sottili (ad esempio uno spicchio di luna crescente) e dettagli complessi lungo i bordi di una forma. Per la GPU è difficile eseguire la tassellatura di queste forme per produrre le strutture triangolari. Per capirne il motivo, considerate due vettori: un quadrato 500 x 500 e una mezzaluna 100 x 10. Una GPU può eseguire con facilità il rendering del quadrato perché questo può essere suddiviso in due triangoli. Al contrario, ci vogliono molti triangoli per descrivere la curva della mezzaluna. Di conseguenza, il rendering della forma è più complicato anche se coinvolge un numero minore di pixel.
- Evitate di modificare pesantemente le dimensioni, poiché tali modifiche possono anche far sì che la GPU debba ripetere la tassellatura dell'immagine.

- Evitate l'overdrawing se possibile. L'overdrawing è la stratificazione di più elementi grafici in modo che si oscurino tra loro. Quando si utilizza il rendering software, ogni pixel viene disegnato una sola volta. Pertanto, nel caso del rendering software, l'applicazione non subisce alcun calo delle prestazioni a prescindere dal numero di elementi grafici che si coprono l'un l'altro nella posizione di quel pixel. Al contrario, con il rendering hardware ciascun pixel viene disegnato per ogni elemento, indipendentemente dal fatto che altri elementi oscurino o meno tale posizione. Se due rettangoli sono parzialmente sovrapposti, il renderer hardware disegna la parte sovrapposta due volte, mentre il renderer software la disegna una sola volta.

  Quindi, sul desktop (che usa il renderer software), in genere non noterete un calo delle prestazioni dovuto all'overdrawing. Tuttavia, la presenza di molte forme sovrapposte può incidere sulle prestazioni dei dispositivi che utilizzano il rendering GPU. È buona norma rimuovere gli oggetti dall'elenco di visualizzazione anziché nasconderli.
- Evitate di usare un rettangolo pieno di grandi dimensioni come sfondo. Impostate invece il colore di sfondo dello stage.
- Evitate di usare il metodo di riempimento bitmap Ripeti quando è possibile. Usate invece la modalità bitmap "clamp" per ottenere prestazioni migliori.

Operazioni asincrone

Privilegiate le versioni asincrone delle operazioni, se disponibili, rispetto a quelle sincrone.

Le operazioni sincrone vengono eseguite non appena ricevono l'istruzione di esecuzione dal codice, che attende il loro completamento prima di proseguire. All'interno del ciclo di fotogramma, pertanto, queste operazioni vengono eseguite nella fase di esecuzione del codice dell'applicazione. Se un'operazione sincrona richiede troppo tempo, le dimensioni del ciclo di fotogramma aumentano e possono causare un blocco apparente dello schermo o una visualizzazione a scatti.

Quando il codice esegue un'operazione asincrona, l'esecuzione non è necessariamente immediata. L'esecuzione del codice della vostra applicazione e delle altre applicazioni incluse nel thread di esecuzione prosegue normalmente. Il runtime esegue quindi l'operazione appena possibile cercando allo stesso tempo di evitare che si verifichino problemi di rendering. In alcuni casi l'esecuzione avviene in background e non nell'ambito del ciclo di fotogramma runtime. Quando l'operazione viene completata, infine, il runtime invia un evento e il codice può intercettare tale evento per eseguire altre attività.

Le operazioni asincrone vengono pianificate e suddivise in modo da evitare problemi di rendering. Di conseguenza, è molto più facile ottenere un'applicazione con tempi di risposta rapidi se si utilizzano le versioni asincrone delle operazioni. Per ulteriori informazioni, vedete “Differenza tra prestazioni percepite e prestazioni effettive” a pagina 3.

L'esecuzione di operazioni in modalità asincrona comporta tuttavia un certo sovraccarico. Il tempo di esecuzione effettivo può essere maggiore per le operazioni asincrone, specialmente per quelle che vengono completate in poco tempo.

Nel runtime, molte operazioni sono intrinsecamente sincrone o asincrone e non consentono di scegliere la modalità di esecuzione. In Adobe AIR, tuttavia, esistono tre tipi di operazioni che potete scegliere di eseguire in modo sincrono o asincrono:

- Operazioni delle classi File e FileStream

  Molte operazioni della classe File possono essere eseguite sia in modo sincrono che asincrono. I metodi utilizzati per copiare o eliminare un file o una directory ed elencare i contenuti di una directory, ad esempio, dispongono di una versione asincrona. Il nome della versione asincrona di questi metodi contiene il suffisso "Async". Per eliminare un file in modo asincrono, ad esempio, chiamate il metodo `File.deleteFileAsync()` anziché il metodo `File.deleteFile()`.

  Quando utilizzate un oggetto FileStream per eseguire operazioni di lettura o scrittura su un file, la modalità di apertura dell'oggetto FileStream determina se le operazioni vengono eseguite in modo asincrono. Utilizzate il metodo `FileStream.openAsync()` per le operazioni asincrone. I dati verranno scritti in modo asincrono. I dati verranno letti a blocchi e saranno disponibili una porzione per volta. In modalità sincrona, al contrario, l'oggetto FileStream legge l'intero file prima di continuare l'esecuzione del codice.

- Operazioni del database SQL locale

  Quando si utilizza un database SQL locale, è possibile utilizzare sia la modalità sincrona che quella asincrona per tutte le operazioni eseguite mediante un oggetto SQLConnection. Per specificare l'esecuzione asincrona delle operazioni, aprite la connessione al database utilizzando il metodo `SQLConnection.openAsync()` anziché il metodo `SQLConnection.open()`. Le operazioni di database asincrone vengono eseguite in background. Poiché il motore del database non viene eseguito nel ciclo di fotogramma runtime, è molto improbabile che le operazioni di database causino problemi di rendering.

  Per ulteriori informazioni su come migliorare le prestazioni del database SQL locale, vedete "Prestazioni del database SQL" a pagina 91.

- Shader Pixel Bender in modalità autonoma

  La classe ShaderJob vi permette di utilizzare un'immagine o un set di dati come input per uno shader Pixel Bender e di accedere ai dati non elaborati dei risultati. Quando chiamate il metodo `ShaderJob.start()`, lo shader viene eseguito in modalità asincrona per impostazione predefinita. L'esecuzione avviene in background e non nel ciclo di fotogramma runtime. Per forzare l'esecuzione sincrona dell'oggetto ShaderJob (scelta non consigliata), passate il valore `true` al primo parametro del metodo `start()`.

Oltre a utilizzare questi meccanismi integrati per l'esecuzione asincrona del codice, potete strutturare il codice in modo che venga eseguito in modo asincrono anziché sincrono. Quando scrivete il codice di un'attività con tempi di esecuzione potenzialmente lunghi, potete strutturare il codice in modo che l'esecuzione avvenga in più parti. Suddividendo il codice in più parti consentirete al runtime di eseguire le operazioni di rendering tra i vari blocchi di esecuzione del codice, riducendo la probabilità che si verifichino problemi di rendering.

Di seguito vengono elencate varie tecniche che potete utilizzare per suddividere il codice. Tutte queste tecniche si basano sul principio che il codice viene scritto in modo da eseguire solo una parte delle attività previste in ogni momento specifico. Tracciate le attività eseguite dal codice e verificate in quali momenti si interrompe l'esecuzione. Utilizzate meccanismi come un oggetto Timer per verificare a più riprese se sono rimaste attività in sospeso ed eseguirle in blocchi finché non sono state completate.

Esistono alcuni modelli consolidati per strutturare il codice in modo che le attività vengano suddivise nella maniera descritta. Gli articoli e le librerie di codice seguenti descrivono questi modelli e forniscono il codice necessario per implementarli nelle vostre applicazioni:

- Articolo di Trevor McCauley contenente informazioni generali e numerosi esempi di implementazione: Asynchronous ActionScript Execution (Esecuzione asincrona del codice ActionScript)

- Articolo di Jesse Warden contenente informazioni generali ed esempi relativi agli approcci “modello Builder” e “green thread”: Parsing & Rendering Lots of Data in Flash Player (Analisi e rendering di quantità elevate di dati in Flash Player)
- Articolo di Drew Cummins che descrive la tecnica dei “green thread” e fornisce il codice sorgente di esempio: Green Threads
- Libreria di codice open source di Charlie Hubbard per l'implementazione dei “green thread” in ActionScript: greenthreads. Per ulteriori informazioni, vedete greenthreads Quick Start (Guida rapida ai green thread).
- Per ulteriori informazioni, vedete l'articolo relativo ai thread in ActionScript 3 all'indirizzo http://www.adobe.com/go/learn_fp_as3_threads_it (articolo a cura di Alex Harui contenente un esempio di implementazione della tecnica “pseudo threading”)

Finestre trasparenti

Nelle applicazioni desktop AIR, utilizzate una finestra di applicazione rettangolare opaca anziché una finestra trasparente.

Per utilizzare una finestra opaca, impostate il seguente valore nel file XML descrittore dell'applicazione per la finestra iniziale di un'applicazione desktop AIR:

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

Per le finestre create dal codice dell'applicazione, create un oggetto NativeWindowInitOptions con la proprietà `transparent` impostata su `false` (impostazione predefinita) e passatelo alla funzione di costruzione NativeWindow mentre create l'oggetto NativeWindow:

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

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

Per un componente Window di Flex, accertatevi che la proprietà transparent del componente sia impostata su false (impostazione predefinita) prima di chiamare il metodo `open()` dell'oggetto Window.

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

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

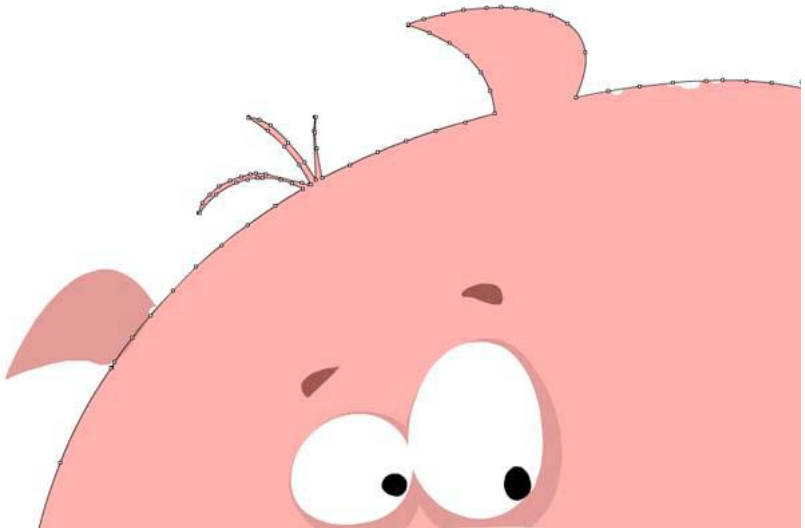
Una finestra trasparente può mostrare una parte del desktop dell'utente o le finestre di altre applicazioni dietro la finestra dell'applicazione. Di conseguenza, il runtime richiede una quantità di risorse superiore per eseguire il rendering di una finestra trasparente. Una finestra rettangolare non trasparente, indipendentemente dal fatto che utilizzi il chrome del sistema operativo o un chrome personalizzato, non comporta lo stesso carico di lavoro per il rendering.

Utilizzate una finestra trasparente solo se vi serve una visualizzazione diversa da quella rettangolare o volete fare in modo che il contenuto dello sfondo sia visibile attraverso la finestra dell'applicazione.

Arrotondamento di forme vettoriali

Arrotondare le forme per migliorare le prestazioni di rendering.

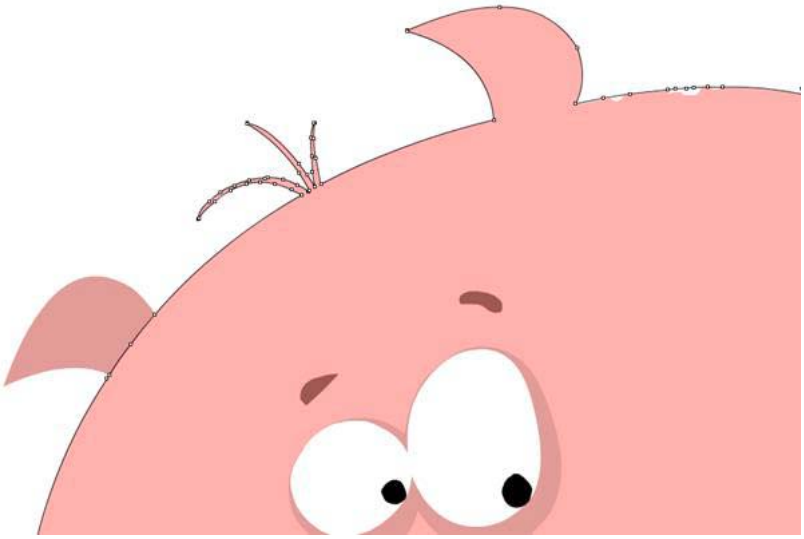
A differenze delle bitmap, il rendering del contenuto vettoriale richiede molti calcoli, in particolare per gradienti e tracciati complessi contenenti numerosi punti di controllo. In qualità di progettista o sviluppatore, accertatevi che le forme sia sufficientemente ottimizzate. La figura seguente illustra tracciati non semplificati con numerosi punti di controllo:



Tracciati non ottimizzati

Utilizzando lo strumento Arrotonda in Flash Professional, potete rimuovere punti di controllo aggiuntivi. Uno strumento equivalente è disponibile in Adobe® Illustrator® e il numero totale di punti e tracciati è visibile nel pannello Informazioni documento.

L'arrotondamento rimuove punti di controllo extra, riducendo la dimensione finale del file SWF e migliorando le prestazioni di rendering. Nella figura seguente vengono illustrati gli stessi tracciati dopo l'arrotondamento:



Tracciati ottimizzati

A condizione di non semplificare troppo i tracciati, questa ottimizzazione non cambia niente dal punto di vista visivo. Tuttavia, la frequenza fotogrammi media dell'applicazione finale può essere notevolmente migliorata semplificando tracciati complessi.

Capitolo 6: Ottimizzazione dell'interazione di rete

Miglioramenti per l'interazione di rete

Flash Player 10.1 e AIR 2.5 introducono una serie di nuove funzioni, quali il buffering circolare e la ricerca intelligente, finalizzate a ottimizzare le prestazioni di rete su tutte le piattaforme.

Buffering circolare

Quando caricate contenuti multimediali sui dispositivi mobili, si possono verificare problemi che non vi aspettereste mai di incontrare su un computer desktop. Ad esempio, è più facile trovarsi in situazioni di esaurimento dello spazio su disco o della memoria. Quando viene caricato un video, le versioni desktop di Flash Player 10.1 e AIR 2.5 scaricano e memorizzano l'intero file FLV (o MP 4) nella cache sul disco rigido, quindi il runtime riproduce il video dal file della cache. È raro che lo spazio su disco si esaurisca. Se tale situazione si verifica, il runtime desktop interrompe la riproduzione del video.

L'esaurimento dello spazio su disco è un problema più frequente su un dispositivo mobile. Se il dispositivo esaurisce lo spazio su disco, il runtime non interrompe la riproduzione come nella versione desktop, bensì inizia a riutilizzare il file della cache riscrivendolo dall'inizio, consentendo così all'utente di continuare la visione del video. L'utente non può spostarsi su un punto del video appartenente alla parte che è stata riscritta, può solo tornare all'inizio del file. Il buffering circolare non viene avviato per impostazione predefinita. Può essere attivato durante la riproduzione, oppure anche all'inizio della riproduzione se il filmato occupa più spazio di quello disponibile su disco o nella RAM. Il runtime richiede almeno 4 MB di RAM o 20 MB di spazio su disco per utilizzare la funzione di buffering circolare.

Nota: *se il dispositivo ha spazio sufficiente su disco, la versione del runtime per dispositivi mobili si comporta esattamente come la versione desktop. Tenete presente che un buffer nella RAM viene utilizzato come alternativa se il dispositivo non dispone di un disco oppure il disco è pieno. È possibile impostare un limite per le dimensioni del file di cache e il buffer della RAM in fase di compilazione. Alcuni file MP4 hanno una struttura che richiede il download dell'intero file prima dell'inizio della riproduzione. Il runtime rileva i file di questo tipo e ne impedisce il download, se lo spazio su disco è insufficiente, e il file MP4 non può essere riprodotto. Può essere opportuno evitare di richiedere il download di tali file.*

Come sviluppatori, ricordate che la ricerca funziona solo entro i limiti del flusso memorizzato nella cache. `NetStream.seek()` a volte fallisce se l'offset è fuori intervallo e, in questo caso, viene inviato un evento `NetStream.Seek.InvalidTime`.

Ricerca intelligente

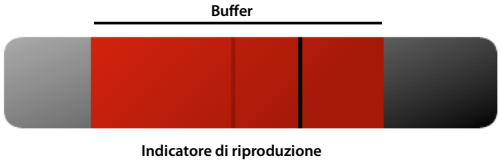
Nota: *la funzione di ricerca intelligente richiede Adobe® Flash® Media Server 3.5.3.*

Flash Player 10.1 e AIR 2.5 introducono un nuovo comportamento, chiamato "ricerca intelligente", che migliora l'esperienza dell'utente durante lo streaming video. Se l'utente cerca una destinazione all'interno dei limiti del buffer, il runtime riutilizza il buffer per offrire una ricerca istantanea. Nelle versioni precedenti del runtime, il buffer non veniva riutilizzato. Se un utente riproduceva, ad esempio, un video da un server di streaming e il tempo del buffer era impostato su 20 secondi (`NetStream.bufferTime`), e l'utente tentava di spostarsi di 10 secondi in avanti, il runtime eliminava tutti i dati di buffer invece di riutilizzare i 10 secondi già caricati. Questo comportamento obbligava il runtime a richiedere nuovi dati dal server molto più frequentemente e causava prestazioni di riproduzione scadenti in caso di connessione lenta.

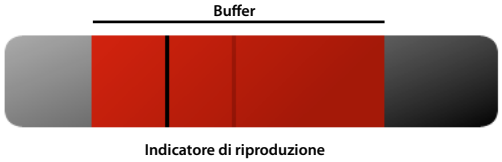
La figura seguente illustra il comportamento del buffer nelle versioni precedenti del runtime. La proprietà `bufferTime` specifica il numero di secondi da precaricare in anticipo in modo che, se la connessione viene interrotta, il buffer possa essere utilizzato senza interrompere il video:

Comportamento del buffer prima della funzione di ricerca intelligente

Grazie alla funzione di ricerca intelligente, il runtime ora utilizza il buffer per fornire la funzionalità di ricerca in avanti e indietro quando l'utente aziona la barra di avanzamento del video. La figura seguente illustra questo nuovo comportamento:



Ricerca in avanti con la funzionalità di ricerca intelligente



Ricerca indietro con la funzionalità di ricerca intelligente

La ricerca intelligente riutilizza il buffer quando l'utente effettua ricerche in avanti o all'indietro, garantendo un'esperienza di riproduzione più veloce e fluida. Uno dei vantaggi di questo nuovo comportamento è il risparmio di banda per chi pubblica contenuti video. Se però una ricerca viene eseguita al di fuori dei limiti del buffer, viene utilizzato il comportamento standard e il runtime richiede i nuovi dati al server.

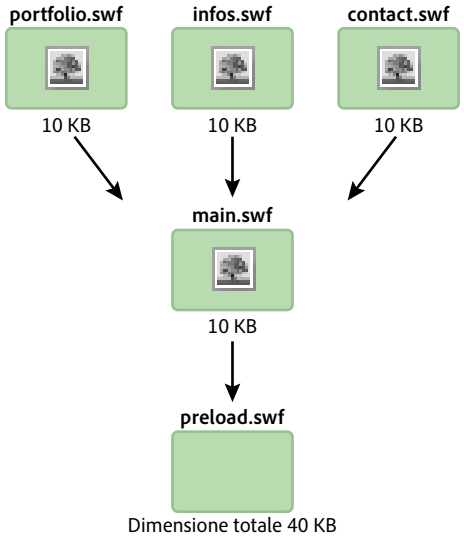
Nota: *questo comportamento non è disponibile per il download video progressivo.*

Per usare la ricerca intelligente, impostate `NetStream.inBufferSeek` su `true`.

Contenuto esterno

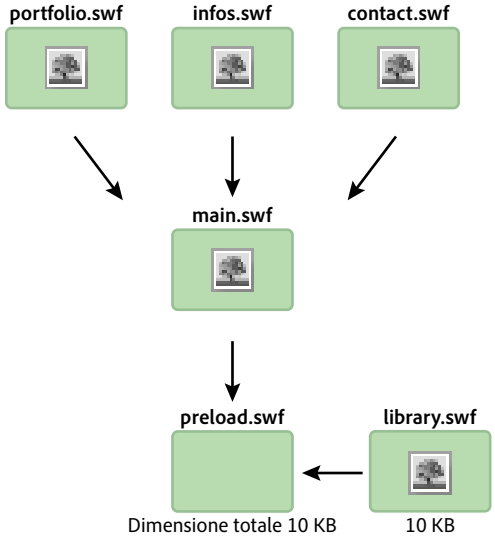
Dividete l'applicazione in più file SWF.

I dispositivi mobili possono avere un accesso limitato alla rete. Per caricare rapidamente il vostro contenuto, dividete l'applicazione in più file SWF. Cercate di riutilizzare la logica del codice e le risorse in tutta l'applicazione. Ad esempio, considerate un'applicazione che è stata divisa in più file SWF, come quella raffigurata nel diagramma seguente:



Applicazione divisa in più file SWF

In questo esempio, ogni file SWF contiene una copia distinta della stessa bitmap. Questa duplicazione può essere evitata utilizzando una libreria condivisa in runtime, come illustrato dal diagramma seguente:



Utilizzando una libreria condivisa in runtime

In base a questa tecnica, una libreria condivisa in runtime viene caricata per rendere disponibile la bitmap agli altri file SWF. La classe ApplicationDomain include tutte le definizioni di classe che sono state caricate e le rende disponibili in runtime tramite il metodo `getDefinition()`.

Una libreria condivisa in runtime può anche contenere tutta la logica del codice. L'intera applicazione può così essere aggiornata in runtime senza che sia necessario ricompilarla. Il codice seguente carica una libreria condivisa in runtime ed estrae la definizione contenuta nel file SWF in fase di runtime. Questa tecnica può essere utilizzata con i font, le bitmap, i suoni o qualsiasi classe ActionScript:

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

Per facilitare l'estrazione della definizione, è possibile caricare le definizioni di classe nel dominio applicazione del file SWF che esegue il caricamento:

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

Ora le classi disponibili nel file SWF caricato possono essere utilizzate chiamando il metodo `getDefinition()` sul dominio applicazione corrente. Potete accedere alle classi anche chiamando il metodo `getDefinitionByName()`. Questa tecnica permette di risparmiare banda caricando i font e le risorse più grandi una sola volta. Le risorse non vengono mai esportate in altri file SWF. L'unica limitazione consiste nel fatto che l'applicazione deve essere testata ed eseguita mediante il file loader.swf. Questo file carica prima le risorse, quindi carica i vari file SWF che compongono l'applicazione.

Errori di input/output

Fornite gestori di eventi e messaggi di errore per gli errori IO.

Su un dispositivo mobile, una connessione di rete può essere meno affidabile rispetto a un computer desktop connesso a Internet ad alta velocità. L'accesso al contenuto esterno sui dispositivi mobili è soggetto a due vincoli: disponibilità e velocità. Pertanto, dovete assicurarvi che le risorse siano "leggere" e aggiungere gestori per ogni evento IO_ERROR in modo da fornire il feedback appropriato all'utente.

Immaginate, ad esempio, che un utente stia navigando nel vostro sito Web tramite un dispositivo mobile e improvvisamente la connessione di rete venga interrotta mentre è in corso di caricamento una risorsa dinamica. Su un computer desktop, potreste usare un listener di eventi vuoto per evitare che venga visualizzato un errore runtime, essendo uno scenario di questo tipo altamente improbabile. Su un dispositivo mobile, invece, dovete gestire la situazione con un accorgimento diverso da un semplice listener vuoto.

Il codice seguente non risponde a un errore I/O. Non utilizzatelo così come è riportato qui:

```
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;
}
```

Una soluzione migliore deve prevedere la gestione di un errore di questo tipo e fornire un messaggio di errore per l'utente. Il codice seguente gestisce correttamente l'errore:

```
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
}
```

Inoltre, è bene offrire all'utente la possibilità di caricare di nuovo il contenuto. Potete implementare questo comportamento mediante il gestore `onIOError()`.

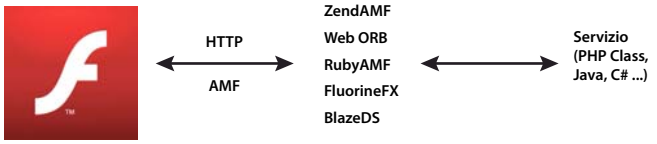
Flash Remoting

Utilizzate Flash Remoting e AMF per ottimizzare le comunicazioni dati client-server.

Potete utilizzare il codice XML per caricare contenuto remoto nei file SWF. XML è tuttavia testo semplice che viene caricato e analizzato dal runtime. È indicato soprattutto per le applicazioni che caricano una quantità limitata di contenuto. Se sviluppate un'applicazione che carica una grande quantità di contenuto, valutate la possibilità di utilizzare la tecnologia Flash Remoting e il formato AMF (Action Message Format).

AMF è un formato binario che consente di condividere dati tra un server e il runtime. L'uso di AMF riduce le dimensioni dei dati e migliora la velocità di trasmissione. Poiché AMF è un formato nativo del runtime, inviando dati AMF al runtime si evita il consumo di memoria dovuto alla serializzazione e deserializzazione sul lato client. Il gateway di remoting si occupa di queste operazioni. Quando un tipo di dati ActionScript viene inviato a un server, il gateway di remoting gestisce autonomamente la serializzazione sul lato server e provvede a inviare il tipo di dati corrispondente. Questo tipo di dati è una classe creata sul server, che espone una serie di metodi che possono essere chiamati dal runtime. I gateway di Flash Remoting disponibili sono ZendAMF, FluorineFX, WebORB e BlazeDS, un gateway Flash Remoting Java ufficiale e open-source creato da Adobe.

La figura seguente illustra il concetto di Flash Remoting:



Flash Remoting

Nell'esempio seguente viene usata la classe NetConnection per connettersi a un gateway Flash Remoting:

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

La connessione a un gateway di Flash Remoting è estremamente lineare. Tuttavia, l'uso di Flash Remoting può essere ulteriormente semplificato utilizzando la classe RemoteObject inclusa in Adobe® Flex® SDK.

Nota: *i file SWC esterni, come quelli del framework Flex, possono essere utilizzati in un progetto di Adobe® Flash® Professional. L'uso dei file SWC vi permette di utilizzare la classe RemoteObject e le relative dipendenze senza utilizzare il resto dell'SDK Flex. Gli sviluppatori più esperti possono anche comunicare direttamente con un gateway di remoting tramite la classe Socket, se necessario.*

Operazioni di rete non necessarie

Memorizzate le risorse nella cache locale dopo averle caricate, anziché caricarle dalla rete ogni volta che servono.

Se l'applicazione carica risorse come file multimediali o dati, memorizzate queste risorse nella cache salvandole sul dispositivo locale. Per le risorse che vengono modificate raramente, prendete in considerazione la possibilità di aggiornare la cache a intervalli regolari. Ad esempio, l'applicazione può controllare se è disponibile una nuova versione di un file di immagine una volta al giorno, oppure verificare la disponibilità di dati aggiornati ogni due ore.

Per memorizzare le risorse nella cache potete procedere in vari modi, a seconda del tipo e della natura della risorsa:

- Risorse multimediali come immagini e video: salvate i file nel file system utilizzando le classi File e FileStream
- Valori di dati singoli o set di dati di piccole dimensioni: salvate i valori come oggetti locali condivisi utilizzando la classe SharedObject
- Set di dati di grandi dimensioni: salvate i dati in un database locale oppure serializzateli e salvateli in un file

Il progetto open-source AS3CoreLib comprende una classe ResourceCache che esegue automaticamente le operazioni di caricamento e memorizzazione nella cache per i valori dei dati.

Capitolo 7: Operazioni con i contenuti multimediali

Video

Per informazioni sull'ottimizzazione delle prestazioni video sui dispositivi mobili, vedete l'articolo Optimize web content for mobile delivery (Ottimizzare contenuti Web da distribuire su dispositivi mobili) sul sito Web Adobe Developer Connection.

In particolare, leggete le seguenti sezioni:

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

Queste sezioni contengono informazioni sullo sviluppo di lettori video per dispositivi mobili, quali:

- Linee guida per la codifica video
- Procedure ottimali
- Come tracciare il profilo delle prestazioni del lettore video
- Un'implementazione di riferimento di un lettore video

StageVideo

Utilizzate la classe StageVideo per sfruttare l'accelerazione hardware per presentare il video.

Per informazioni sull'utilizzo dell'oggetto StageVideo, vedete Uso della classe StageVideo per la presentazione con accelerazione hardware nella Guida per gli sviluppatori di ActionScript 3.0.

Audio

A partire da Flash Player 9.0.115.0 e AIR 1.0, il runtime consente l'esecuzione di file AAC (AAC Main, AAC LC e SBR). Una semplice ottimizzazione può essere ottenuta utilizzando i file AAC anziché il formato MP3. Il formato AAC offre una qualità migliore e dimensioni più contenute rispetto al formato MP3 alla stessa frequenza di bit. La riduzione delle dimensioni dei file permette di risparmiare ampiezza di banda, fattore importante per i dispositivi mobili che non dispongono di connessioni Internet ad alta velocità.

Decodifica audio hardware

Analogamente alla decodifica video, la decodifica audio richiede elevati cicli della CPU e può essere ottimizzata sfruttando le capacità dell'hardware disponibile sul dispositivo. Flash Player 10.1 e AIR 2.5 sono in grado di rilevare e utilizzare i driver audio dell'hardware per migliorare le prestazioni di codifica dei file AAC (LC, profili HE/SBR) o MP 3 (PCM non è supportato). L'uso della CPU si riduce in modo significativo, con un conseguente risparmio della carica della batteria e una maggiore disponibilità della CPU per altre operazioni.

Nota: *quando si usa il formato AAC, il profilo AAC Main non è supportato sui dispositivi a causa dell'assenza del supporto hardware in quasi tutti i dispositivi.*

La decodifica audio hardware è trasparente all'utente e allo sviluppatore. Quando il runtime inizia a riprodurre i flussi audio, verifica innanzitutto l'hardware, come nel caso del video. Se è disponibile un driver hardware e il formato audio è supportato, viene eseguita la decodifica audio hardware. Tuttavia, anche se la decodifica dei flussi audio AAC o MP3 in entrata può essere gestita mediante l'hardware, a volte l'hardware non è in grado di elaborare tutti gli effetti. Ad esempio, in alcuni casi l'hardware non elabora il mixaggio audio e il ricampionamento, a seconda delle caratteristiche e dei limiti dell'hardware stesso.

Capitolo 8: Prestazioni del database SQL

Strategie di progettazione dell'applicazione per il miglioramento delle prestazioni del database

Non modificate la proprietà `text` di un oggetto SQLStatement dopo averlo eseguito. Utilizzate invece un'istanza SQLStatement per ogni istruzione SQL e utilizzate i parametri delle istruzioni per fornire valori diversi.

Prima dell'esecuzione di qualsiasi istruzione SQL, il runtime la prepara (compila) al fine di determinare i passi da effettuare internamente per eseguirla. Quando si chiama `SQLStatement.execute()` su un'istanza SQLStatement che non è mai stata eseguita in precedenza, l'istruzione viene preparata automaticamente prima dell'esecuzione. In occasione delle successive chiamate del metodo `execute()`, se la proprietà `SQLStatement.text` non ha subito modifiche, l'istruzione risulta ancora preparata e, di conseguenza, viene eseguita più rapidamente.

Per ottenere il massimo vantaggio dal riutilizzo delle istruzioni, se i valori cambiano tra un'esecuzione e l'altra, utilizzate i parametri per personalizzare le istruzioni. I parametri vengono specificati mediante la proprietà di array associativo `SQLStatement.parameters`. A differenza di quanto avviene quando si modifica la proprietà `text` dell'istanza SQLStatement, se si modificano i valori dei parametri delle istruzioni non è necessario che il runtime prepari di nuovo le istruzioni.

Quando si riutilizza un'istanza SQLStatement, l'applicazione deve memorizzare un riferimento a tale istanza dopo che è stata preparata. Per tenere un riferimento all'istanza, dichiarate la variabile come variabile di classe-ambito invece che di funzione-ambito. A questo scopo è consigliabile strutturare l'applicazione in modo tale che l'istruzione SQL sia contenuta in una sola classe. Anche i gruppi di istruzioni che vengono eseguite congiuntamente possono essere contenuti in una sola classe. Questa tecnica utilizza il modello di progettazione Command. Se le istanze vengono definite come variabili membro della classe, permangono per tutto il tempo in cui l'istanza della classe che le contiene continua a esistere nell'applicazione. Come minimo, è possibile semplicemente definire una variabile che contiene l'istanza SQLStatement al fuori di una funzione, in modo che tale istanza permanga in memoria. Ad esempio, dichiarate l'istanza SQLStatement come variabile membro in una classe ActionScript o come variabile senza funzione in un file JavaScript. È poi possibile impostare i valori dei parametri dell'istruzione e chiamare il relativo metodo `execute()` quando si desidera eseguire effettivamente la query.

Utilizzate gli indici dei database per aumentare la velocità di esecuzione per il confronto e l'ordinamento dei dati.

Quando create un indice per una colonna, il database memorizza una copia dei dati della colonna. I dati della copia vengono memorizzati in ordine numerico o alfabetico. L'ordinamento consente al database di confrontare rapidamente i valori (ad esempio quando si utilizza l'operatore di uguaglianza) e ordinare i dati dei risultati utilizzando la clausola `ORDER BY`.

Gli indici del database vengono mantenuti costantemente aggiornati, causando un leggero rallentamento delle operazioni di modifica dei dati (INSERT o UPDATE) nella tabella. In genere, tuttavia, l'aumento della velocità di recupero dei dati può essere significativo. Per via di questo compromesso a livello di prestazioni, è consigliabile evitare di indicizzare ogni colonna di ogni tabella e utilizzare invece una strategia mirata per definire gli indici. Per pianificare la vostra strategia di indicizzazione, attenetevi alle seguenti linee guida:

- Indicizzate le colonne che vengono utilizzate nelle operazioni di join delle tabelle, nelle clausole WHERE o nelle clausole ORDER BY
- Create un unico indice per le colonne che vengono utilizzate spesso insieme

- Per le colonne che contengono dati di testo da recuperare in ordine alfabetico, specificate la sequenza di confronto COLLATE NOCASE per l'indice

Precompilate le istruzioni SQL durante i periodi di inattività dell'applicazione.

La prima esecuzione di un'istruzione SQL è più lenta perché il testo SQL viene preparato (compilato) dal motore del database. Poiché la preparazione e l'esecuzione di un'istruzione possono richiedere molte risorse, è consigliabile precaricare i dati iniziali e quindi eseguire le altre istruzioni in background:

1. Caricate per primi i dati che servono all'applicazione.
2. Quando le operazioni di avvio iniziali dell'applicazione sono state completate, oppure durante un altro periodo di "inattività" dell'applicazione, eseguite le altre istruzioni.

Supponiamo, ad esempio, che l'applicazione non abbia la necessità di accedere al database per visualizzare la schermata iniziale. In questo caso, aspettate che venga visualizzata la schermata iniziale prima di aprire la connessione al database. Infine, create le istanze SQLStatement ed eseguitene il maggior numero possibile.

Supponiamo in alternativa che l'applicazione visualizzi immediatamente alcuni dati, ad esempio i risultati di una query, al momento dell'avvio. In tal caso, procedete all'esecuzione dell'istanza SQLStatement relativa a quella query. Dopo il caricamento e la visualizzazione dei dati iniziali, create le istanze SQLStatement per le altre operazioni di database e, se possibile, eseguite altre istruzioni necessarie in un secondo momento.

Se riutilizzate le istanze SQLStatement, in pratica, il tempo aggiuntivo che serve per preparare l'istruzione è un "inconveniente" che si verifica una tantum e probabilmente non ha un impatto rilevante sulle prestazioni generali.

Raggruppate più operazioni di modifica dei dati SQL in una transazione.

Supponiamo che eseguiate numerose istruzioni SQL che comportano l'aggiunta o la modifica di dati (istruzioni `INSERT` o `UPDATE`). È possibile ottenere un notevole aumento delle prestazioni eseguendo tutte le istruzioni nell'ambito di una transazione esplicita. Se non si dà inizio esplicitamente a una transazione, ciascuna istruzione viene eseguita in una propria transazione individuale creata automaticamente. Al termine dell'esecuzione di ogni transazione (istruzione), il runtime scrive i dati risultanti nel file del database sul disco.

Consideriamo invece ora gli sviluppi a seguito della creazione esplicita di una transazione e dell'esecuzione delle istruzioni nell'ambito di tale transazione. Il runtime apporta tutte le modifiche in memoria, poi le scrive tutte insieme, contemporaneamente, nel file del database una volta che è stato eseguito il commit della transazione. La scrittura dei dati sul disco rappresenta di solito la fase dell'operazione che richiede più tempo. Di conseguenza, una sola operazione di scrittura su disco, invece che un'operazione di scrittura per ciascuna istruzione SQL, consente un notevole miglioramento delle prestazioni.

Elaborate in più parti i set di risultati di grandi dimensioni delle query `SELECT` utilizzando i metodi `execute()` (con il parametro `prefetch`) e `next()` della classe SQLStatement.

Supponiamo che eseguiate un'istruzione SQL che restituisce un set di risultati di grandi dimensioni. L'applicazione elabora ciascuna riga di dati in modo ciclico, ad esempio per formattare i dati o creare oggetti a partire dai dati stessi. L'elaborazione di questi dati può richiedere molto tempo e causare problemi di rendering, ad esempio una mancanza di risposta o il blocco dello schermo. Come illustrato in "Operazioni asincrone" a pagina 76, una possibile soluzione è rappresentata dalla suddivisione delle attività di elaborazione in più blocchi. L'API del database SQL semplifica la suddivisione dell'elaborazione dei dati.

Il metodo `execute()` della classe SQLStatement è dotato di un parametro `prefetch` opzionale (il primo parametro). L'eventuale valore fornito specifica il numero massimo di righe di risultati che vengono restituite dal database al termine dell'esecuzione:

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

Dopo che è stato restituito il primo set di dati dei risultati, potete chiamare il metodo `next()` per continuare l'esecuzione dell'istruzione e recuperare un altro set di righe di risultati. Analogamente al metodo `execute()`, il metodo `next()` accetta un parametro `prefetch` che specifica il numero massimo di righe da restituire:

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

Potete continuare a chiamare il metodo `next()` finché non sono stati caricati tutti i dati. Come illustrato nell'esempio precedente, è possibile verificare se il caricamento dei dati è stato completato o meno. Verificate la proprietà `complete` dell'oggetto SQLResult che viene creato al termine di ogni esecuzione del metodo `execute()` o `next()`.

Nota: *per dividere l'elaborazione dei dati dei risultati, potete utilizzare il parametro `prefetch` e il metodo `next()`. Non utilizzate questo parametro e questo metodo per limitare i risultati di una query a una parte del relativo set di risultati. Se desiderate recuperare solo un sottoinsieme di righe nel set di risultati di un'istruzione, utilizzate la clausola `LIMIT` dell'istruzione `SELECT`. Per i set di risultati di grandi dimensioni potete comunque utilizzare il parametro `prefetch` e il metodo `next()` per suddividere l'elaborazione dei risultati.*

Utilizzate più oggetti SQLConnection asincroni con un unico database per eseguire più istruzioni contemporaneamente.

Se un oggetto SQLConnection si connette a un database utilizzando il metodo `openAsync()`, l'oggetto viene eseguito in background anziché nel thread di esecuzione runtime principale. Inoltre, ogni oggetto SQLConnection viene eseguito in un thread di background distinto. Utilizzando più oggetti SQLConnection, potete quindi eseguire più istruzioni SQL contemporaneamente.

L'uso di questo approccio può comportare alcuni svantaggi. Il principale è che ogni oggetto SQLStatement aggiuntivo richiede una quantità supplementare di memoria. Le esecuzioni contemporanee determinano inoltre un carico di lavoro aggiuntivo per il processore, soprattutto sui computer che hanno una sola CPU o un solo core di CPU. Questo approccio è quindi sconsigliato per i dispositivi mobili.

Un ulteriore problema è rappresentato dal fatto che non sempre si possono sfruttare i vantaggi offerti dal riutilizzo degli oggetti SQLStatement perché ogni oggetto SQLStatement è collegato a un unico oggetto SQLConnection. Di conseguenza, un oggetto SQLStatement non può essere riutilizzato se l'oggetto SQLConnection associato è già in uso.

Se scegliete di utilizzare più oggetti SQLConnection connessi a un unico database, tenete presente che ognuno di questi oggetti esegue le proprie istruzioni in una transazione a se stante. Tenete conto di queste transazioni separate, ad esempio l'aggiunta, la modifica o l'eliminazione di dati, contenute in qualsiasi blocco di codice che modifica i dati.

Paul Robertson ha creato una libreria di codice open source che vi consente di sfruttare i vantaggi offerti dall'uso di più oggetti SQLConnection riducendo al minimo i possibili inconvenienti. Questa libreria utilizza un pool di oggetti SQLConnection e gestisce gli oggetti SQLStatement associati. In questo modo garantisce la possibilità di riutilizzare gli oggetti SQLStatement e la disponibilità di più oggetti SQLConnection per l'esecuzione contemporanea di più istruzioni. Per ulteriori informazioni e per scaricare la libreria, visitate http://probertson.com/projects/air-sqlite/.

Ottimizzazione dei file di database

Evitate di apportare modifiche allo schema del database.

Se possibile, evitate di modificare lo schema (cioè la struttura delle tabelle) dopo che sono stati aggiunti dati nelle tabelle del database. Di norma, i file di database sono strutturati in modo da elencare le definizioni delle tabelle all'inizio. Quando si apre la connessione al database, il runtime carica tali definizioni. Quando si aggiungono dati alle tabelle del database, essi vengono aggiunti al file di seguito ai dati di definizione delle tabelle. Se, tuttavia, apportate modifiche allo schema, i nuovi dati di definizione della tabella vengono mischiati nel file del database con quelli della tabella. Aggiungendo, ad esempio, una colonna a una tabella o aggiungendo una nuova tabella, è possibile che i tipi di dati vengano mischiati. Se i dati di definizione della tabella non si trovano tutti all'inizio del file di database, l'apertura della connessione al database richiederà più tempo. La lentezza di apertura della connessione è causata dal runtime, che impiega più tempo per leggere i dati di definizione della tabella da più parti del file.

Utilizzate il metodo `SQLConnection.compact()` per ottimizzare un database dopo averne modificato lo schema.

Se risulta necessario apportare modifiche allo schema, potete chiamare il metodo `SQLConnection.compact()` dopo aver portato a termine le modifiche. Questa operazione ristruttura il file del database in modo da riunire in un solo blocco contiguo all'inizio del file i dati di definizione delle tabelle. Tuttavia, l'operazione `compact()` può richiedere molto tempo, specialmente man mano che il file del database aumenta di dimensioni.

Elaborazione runtime non necessaria del database

Utilizzate nomi di tabella completi (comprensivi del nome del database) nelle istruzioni SQL.

All'interno delle istruzioni, specificate il nome del database in modo esplicito insieme al nome di ogni tabella. Utilizzate "main" se si tratta del database principale. Il codice seguente, ad esempio, include esplicitamente il nome di database `main`:

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

Specificando esplicitamente il nome del database si evita al runtime di controllare tutti i database connessi per individuare la tabella corrispondente. Si evita inoltre la possibilità che il runtime scelga il database errato. Seguite questa regola anche se un oggetto SQLConnection è connesso a un solo database. Dietro le quinte, infatti, l'oggetto SQLConnection è connesso anche a un database temporaneo accessibile tramite istruzioni SQL.

Utilizzate nomi di colonna espliciti nelle istruzioni SQL `INSERT` e `SELECT`.

Gli esempi seguenti illustrano l'uso dei nomi di colonna espliciti:

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

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

Confrontate gli esempi precedenti con quelli che seguono. Evitate questo stile di codice:

```
-- 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
```

Se non utilizzate nomi di colonna espliciti, il runtime deve svolgere un lavoro aggiuntivo per individuare i nomi delle colonne. Se un'istruzione `SELECT` utilizza un carattere jolly anziché colonne esplicite, il runtime recupererà dati aggiuntivi. che richiedono un'elaborazione supplementare e creano istanze di oggetti superflue.

Evitate di eseguire più volte il join della stessa tabella in un'istruzione, a meno che non desideriate confrontare la tabella con se stessa.

Con l'aumento delle istruzioni SQL, potreste unire involontariamente più volte una tabella di database alla query. Spesso lo stesso risultato può essere ottenuto utilizzando la tabella una sola volta. È possibile che una tabella venga unita più volte se utilizzate una o più viste in una query. Potreste, ad esempio, essere in procinto di unire una tabella alla query e avere aperta contemporaneamente una vista contenente i dati della tabella. Le due operazioni potrebbero causare più di un'unione (join).

Sintassi SQL efficiente

Utilizzate `JOIN` (nella clausola `FROM`) per includere una tabella in una query anziché usare una sottoquery nella clausola `WHERE`. Questo suggerimento vale anche se i dati della tabella vi servono solo per le operazioni di filtraggio e non per il set di risultati.

L'esecuzione del join di più tabelle nella clausola `FROM` offre risultati migliori rispetto all'uso di una sottoquery nella clausola `WHERE`.

Evitate le istruzioni SQL che non supportano l'uso di indici. Queste istruzioni includono l'uso di funzioni aggregate in una sottoquery, un'istruzione `UNION` in una sottoquery o una clausola `ORDER BY` con un'istruzione `UNION`.

Gli indici possono aumentare significativamente la velocità di elaborazione di una query `SELECT`. In alcuni casi, tuttavia, la sintassi SQL impedisce al database di utilizzare gli indici e lo obbliga a usare i dati effettivi per le operazioni di ricerca e ordinamento.

Evitate di utilizzare l'operatore `LIKE`, soprattutto con un carattere jolly iniziale come in `LIKE('%XXXX%')`.

Poiché l'operazione `LIKE` supporta le ricerche con caratteri jolly, la sua esecuzione è più lenta rispetto ai confronti con corrispondenza esatta. In particolare, se iniziate la stringa di ricerca con un carattere jolly, il database non potrà utilizzare gli indici nella ricerca. È invece necessario che il database possa cercare tutto il testo di ogni riga della tabella.

Evitate di utilizzare l'operatore `IN`. Se conoscete in anticipo i valori possibili, potete scrivere l'operazione `IN` utilizzando `AND` o `OR` per accelerare l'esecuzione.

La seconda delle due istruzioni seguenti viene eseguita più velocemente. perché utilizza semplici espressioni di uguaglianza insieme a `OR` invece di utilizzare le istruzioni `IN()` o `NOT IN()`:

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

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

Provate a utilizzare versioni alternative di un'istruzione SQL per migliorare le prestazioni.

Come dimostrano gli esempi precedenti, il modo in cui scrivete un'istruzione SQL può influire anche sulle prestazioni del database. In genere è possibile scrivere un'istruzione SQL `SELECT` in vari modi per recuperare un determinato set di risultati. In alcuni casi un approccio è significativamente più veloce di un altro. Oltre ai suggerimenti indicati in precedenza, potete consultare risorse specifiche del linguaggio SQL per ottenere maggiori informazioni sulle varie istruzioni SQL e sulle relative prestazioni.

Prestazioni delle istruzioni SQL

Eseguite un confronto diretto tra istruzioni SQL alternative per individuare quella più veloce.

Il modo migliore per confrontare le prestazioni di più versioni di un'istruzione SQL è testarle direttamente con il database e i dati reali.

Gli strumenti di sviluppo indicati di seguito forniscono i tempi di esecuzione delle istruzioni SQL e possono essere utilizzati per confrontare la velocità delle versioni alternative delle istruzioni:

- Run! (strumento di Paul Robertson che consente di creare e testare le query SQL di AIR)
- Lita (strumento di amministrazione SQLite di David Deraedt)

Capitolo 9: Benchmarking e distribuzione

Benchmarking

Sono disponibili alcuni tool per il benchmarking delle applicazioni. Potete utilizzare la classe Stats e la classe PerformanceTest, sviluppate dai membri della comunità Flash. Oppure, potete usare il profiler in Adobe® Flash® Builder™, e il tool FlexPMD.

La classe Stats

Per eseguire il profiling del codice in fase di runtime utilizzando la versione release del runtime, senza tool esterni, potete ricorrere alla classe Stats sviluppata da mr. doob della comunità Flash. Potete scaricare la classe Stats all'indirizzo seguente: https://github.com/mrdoob/Hi-ReS-Stats.

La classe Stats permette di tenere traccia dei seguenti aspetti:

- I fotogrammi al secondo di cui viene eseguito il rendering (più elevato è il numero, migliore è il risultato)
- I millisecondi utilizzati per il rendering di un fotogramma (più basso è il numero, migliore è il risultato)
- La quantità di memoria utilizzata dal codice. Se aumenta per ciascun fotogramma, è possibile che l'applicazione presenti una perdita di memoria. È importare verificare questo potenziale problema.
- La quantità massima di memoria utilizzata dall'applicazione.

Una volta scaricata, la classe Stats può essere utilizzata con il seguente codice compatto:

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

Utilizzando la compilazione condizionale in Adobe® Flash® Professional o Flash Builder, potete abilitare l'oggetto Stats:

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

Commutando il valore della costante `DEBUG`, potete abilitare o disabilitare la compilazione dell'oggetto Stats. Lo stesso approccio può essere adottato per sostituire la logica del codice che non volete sia compilata nell'applicazione.

La classe PerformanceTest

Per il profiling dell'esecuzione del codice ActionScript, Grant Skinner ha sviluppato un tool che può essere integrato nel flusso di lavoro di testing di un'unità. Viene passata una classe personalizzata alla classe PerformanceTest, che esegue una serie di test sul codice. La classe PerformanceTest permette di effettuare con facilità il benchmarking di più approcci diversi. La classe PerformanceTest può essere scaricata dall'indirizzo seguente: http://www.gskinner.com/blog/archives/2009/04/as3_performance.html.

Il profiler di Flash Builder

Flash Builder viene fornito con un profiler che consente di eseguire il benchmarking del codice con un maggior livello di dettaglio.

Nota: *utilizzate la versione debugger di Flash Player per accedere al profiler, altrimenti verrà visualizzato un messaggio di errore.*

Il profiler può essere utilizzato anche con i contenuti creati in Adobe Flash Professional. A questo scopo, caricate il file SWF compilato da un progetto ActionScript o Flex in Flash Builder, quindi potrete eseguire il profiler sul progetto caricato. Per ulteriori informazioni sul profiler, vedete "Profiling Flex applications" in Using Flash Builder 4.

FlexPMD

Adobe Technical Services ha rilasciato un tool chiamato FlexPMD che consente di verificare la qualità del codice ActionScript 3.0. FlexPMD è un tool ActionScript simile a JavaPMD. Permette di migliorare la qualità del codice verificando una directory di origine ActionScript 3.0 o Flex. È in grado di rilevare le imperfezioni del codice, ad esempio parti di codice inutilizzate, codice troppo complesso o troppo lungo e uso non corretto del ciclo di vita dei componenti Flex.

FlexPMD è un progetto open-source Adobe disponibile al seguente indirizzo: http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD. È anche disponibile un plugin Eclipse al seguente indirizzo: http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD+Eclipse+plugin.

FlexPMD facilita la verifica del codice e vi permette di ottenere un codice sempre "pulito" e ottimizzato. La vera forza di FlexPMD è nella sua estensibilità. Ogni sviluppatore può infatti creare dei propri set di regole da seguire per la verifica di qualunque codice. Ad esempio, potete creare un set di regole per rilevare un uso eccessivo di filtri, o qualunque altro esempio di uso non corretto del codice abbiate bisogno di individuare.

Distribuzione

Quando esportate la versione finale di un'applicazione in Flash Builder, è importante esportare la versione release. L'esportazione di una versione release rimuove tutte le informazioni di debug contenute nel file SWF, producendo così un file SWF di dimensioni più piccole che consentirà un'esecuzione più veloce dell'applicazione.

Per esportare la versione release di un progetto, utilizzate il pannello Project in Flash Builder e l'opzione Export Release Build.

Nota: *se compilate un progetto in Flash Professional, non avete la possibilità di scegliere tra versione release e versione debug. Il file SWF compilato è sempre in versione release per impostazione predefinita.*