Optimisation des performances pour la plate-forme ADOBE® FLASH®

Informations juridiques

Dernière mise à jour le 9/5/2012

Sommaire

Chapitre 1 : Présentation

Il est possible d'exécuter les applications Adobe® AIR® et Adobe® Flash® sur de nombreuses plates-formes, notamment sur des ordinateurs de bureau, des périphériques mobiles, des tablettes et des téléviseurs. Le présent document s'appuie sur des exemples et des études de cas pour présenter les normes de bonne pratique destinées aux développeurs qui déploient ces applications. Il traite des sujets suivants :

- Conservation de mémoire
- Réduction de la charge de l'unité centrale
- Amélioration des performances d'ActionScript 3.0
- Augmentation de la vitesse de rendu
- Optimisation de l'interaction avec le réseau
- Utilisation de l'audio et de la vidéo
- Optimisation des performances de la base de données SQL
- Test de performances et déploiement d'applications

La plupart des ces optimisations s'appliquent aux applications sur tous les périphériques, qu'il s'agisse du moteur d'exécution d'AIR ou du moteur d'exécution de Flash Player. Ce document décrit également les ajouts et exceptions correspondant à certains périphériques.

Certaines de ces optimisations sont centrées sur les fonctionnalités introduites dans Flash Player 10.1 et AIR 2.5. Néanmoins, la plupart d'entre elles s'appliquent également aux versions précédentes d'AIR et de Flash Player.

Principes fondamentaux relatifs à l'exécution du code par le moteur d'exécution

Pour comprendre comment améliorer les performances d'une application, il est essentiel de comprendre comment le moteur d'exécution de la plate-forme Flash exécute le code. Le moteur d'exécution fonctionne en boucle, certaines actions se produisant sur chaque « image ». On entend ici par image un simple bloc de temps déterminé par la cadence définie pour l'application. Le temps alloué à chaque image correspond directement à la cadence. Si vous spécifiez une cadence de 30 images par seconde, par exemple, le moteur d'exécution s'efforce de faire durer chaque image un trentième de seconde.

Vous définissez la cadence initiale de l'application au moment où vous créez celle-ci. Pour ce faire, vous pouvez utiliser les paramètres correspondants d'Adobe® Flash® Builder™ ou Flash Professional. Libre à vous également de définir la cadence initiale dans le code. Pour définir la cadence d'une application basée uniquement sur ActionScript, appliquez la balise de métadonnées `[SWF(frameRate="24")]` à la classe du document racine. En MXML, définissez l'attribut `frameRate` dans la balise Application ou WindowedApplication.

Chaque boucle d'image comprend deux phases, divisées en trois parties : les événements, l'événement `enterFrame` et le rendu.

La première phase comporte deux parties (les événements et l'événement `enterFrame`), qui entraînent potentiellement toutes deux l'appel de votre code. Dans la première partie de la première phase, des événements du moteur d'exécution arrivent et sont distribués. Ces événements peuvent représenter la fin ou la progression d'opérations asynchrones telles qu'une réponse à un chargement de données sur un réseau. Ils peuvent également être la conséquence d'une entrée de l'utilisateur. Au fur et à mesure de la distribution des événements, le moteur d'exécution exécute le code dans des écouteurs que vous avez enregistrés. En l'absence d'événements, le moteur d'exécution attend, sans agir, la fin de cette phase d'exécution. Il n'accélère jamais la cadence par manque d'activité. Si des événements se produisent dans d'autres parties du cycle d'exécution, le moteur d'exécution les place en file d'attente et les distribue sur l'image suivante.

La deuxième partie de la première phase correspond à l'événement `enterFrame`. Cet événement se distingue des autres en ce qu'il est toujours distribué une fois par image.

Une fois tous les événements distribués, la phase de rendu de la boucle d'image commence. A ce stade, le moteur d'exécution calcule l'état de tous les éléments visibles à l'écran et les dessine. Le processus peut alors recommencer, à l'instar d'un coureur qui fait des circuits dans un stade.

Remarque : *dans le cas des événements comprenant la propriété* `updateAfterEvent`*, il est possible d'imposer un rendu immédiat plutôt que d'attendre la phase de rendu. Evitez toutefois d'utiliser* `updateAfterEvent` *si elle entraîne fréquemment des problèmes de performances.*

Il est plus facile d'imaginer que la durée des deux phases de la boucle d'image est identique. Dans ce cas, une moitié de chaque boucle est dévolue à l'exécution des gestionnaires d'événements et du code d'application, tandis que la seconde moitié est consacrée au rendu. La réalité est néanmoins souvent toute autre. Il arrive que le code d'application utilise plus de la moitié du temps disponible dans l'image, étirant ainsi le créneau qui lui est alloué et réduisant celui du rendu. Dans d'autres cas, notamment lorsque le contenu visuel est complexe (filtres et modes de fondu, par exemple), c'est le rendu qui exige plus de la moitié du temps. La durée des phases étant variable, on dit souvent de la boucle d'image qu'elle est « élastique ».

Si les opérations combinées de la boucle d'image (exécution du code et rendu) durent trop longtemps, le moteur d'exécution ne peut pas assurer une cadence uniforme. L'image s'étend au-delà du temps qui lui est alloué, retardant ainsi le déclenchement de l'image suivante. Si, par exemple, une boucle d'image dépasse un trentième de seconde, le moteur d'exécution ne peut pas mettre l'écran à jour à 30 images par seconde. Le ralentissement de la cadence se traduit par une détérioration de l'expérience de l'utilisateur. Au mieux, l'animation est saccadée ; au pire, l'application se bloque et la fenêtre est vide.

Pour plus de détails sur l'exécution du code par le moteur d'exécution de la plate-forme Flash et le modèle de rendu, voir les ressources suivantes :

- Flash Player Mental Model - The Elastic Racetrack (article de Ted Patrick, disponible en anglais uniquement)
- Asynchronous ActionScript Execution (article de Trevor McCauley, disponible en anglais uniquement)
- Optimizing Adobe AIR for code execution, memory & rendering à l'adresse http://www.adobe.com/go/learn_fp_air_perf_tv_fr (Enregistrement vidéo de la présentation de Sean Christmann lors de la conférence MAX, disponible en anglais uniquement)

Perception et réalité en matière de performances

Les utilisateurs de votre application sont les ultimes juges de ses performances. Les développeurs peuvent mesurer les performances d'une application en terme de la durée d'exécution de certaines opérations ou du nombre d'occurrences d'objet créées. Ces mesures ne présentent cependant aucun intérêt pour l'utilisateur. Celui-ci mesure parfois les performances selon d'autres critères. Par exemple, l'application s'exécute-t-elle rapidement et sans saccades ? Réagit-elle rapidement aux entrées ? A-t-elle un impact négatif sur les performances du système ? Pour tester les performances perçues, posez-vous les questions suivantes :

- Les animations sont-elles fluides ou saccadées ?
- Le contenu vidéo est-il fluide ou saccadé ?
- Les clips audio s'exécutent-ils en continu ou contiennent-ils des interruptions ?
- La fenêtre scintille-t-elle ou se vide-t-elle pendant les opérations de longue durée ?
- Y a-t-il un décalage entre le moment où vous effectuez une saisie et l'affichage du texte ?
- Si vous cliquez sur un élément, la réponse est-elle instantanée ou un délai se produit-il ?
- Le ventilateur de l'unité centrale fait-il plus de bruit lorsque l'application s'exécute ?
- Sur un ordinateur portable ou un périphérique mobile, la batterie se décharge-t-elle rapidement lors de l'exécution de l'application ?
- Les autres applications réagissent-elles plus lentement lorsque l'application s'exécute ?

Il est important de faire la distinction entre perception et réalité, car vous ne procéderez pas nécessairement de même pour optimiser les performances perçues ou pour accélérer au maximum les performances. Veillez à ce que l'application n'exécute jamais de tels volumes de code que le moteur d'exécution se trouve dans l'impossibilité de mettre à jour l'écran et de recueillir les entrées de l'utilisateur à fréquence régulière. Pour parvenir à cet équilibre, il est parfois nécessaire de diviser une tâche de programme en plusieurs parties, entre lesquelles le moteur d'exécution pourra mettre l'écran à jour (voir « Performances de rendu » à la page 49 pour plus de détails).

Les astuces et techniques décrites ci-après visent à vous permettre d'optimiser l'exécution du code lui-même et les performances perçues par l'utilisateur.

Ciblage des optimisations

Certaines améliorations des performances ne font pas de différence notoire pour l'utilisateur. Il est important de bien cibler les optimisations sur les zones problématiques de l'application concernée. Certaines techniques d'optimisation sont bonnes à mettre en pratique dans tous les cas. Pour d'autres, ce sont les exigences de l'application et la base d'utilisateurs visée qui en déterminent l'utilité. Il est vrai, par exemple, que les applications sont plus performantes si vous éliminez toute animation ou vidéo, de même que les filtres graphiques et les effets. Ce sont cependant ses fonctionnalités multimédias et graphiques qui sont l'une des raisons d'utiliser la plate-forme Flash pour créer des applications riches et expressives. Déterminez si le niveau de complexité souhaité est adapté aux performances caractéristiques des machines et périphériques sur lesquelles l'application s'exécutera.

Suivez ce conseil courant : « Ne cherchez pas à optimiser les performances trop tôt. » Certaines optimisations nécessitent de programmer du code peu lisible ou plus rigide. Il est alors plus difficile d'assurer la maintenance de ce code une fois qu'il est optimisé. Dans ce cas, il est souvent préférable d'attendre pour déterminer si les performances d'une portion spécifique du code sont médiocres avant de décider de son optimisation.

L'amélioration des performances exige souvent des compromis. En théorie, la réduction de la quantité de mémoire consommée par une application se traduit par une accélération de la vitesse d'exécution d'une tâche par l'application. En pratique, ce type d'amélioration n'est pas toujours possible. Imaginons que l'application se bloque pendant une opération. Pour résoudre ce problème, il est souvent nécessaire de répartir des tâches sur plusieurs images. Cette division se soldera, selon toute probabilité, par un ralentissement global du processus. Il se peut toutefois que l'utilisateur ne remarque pas le temps supplémentaire, car l'application continue de répondre à ses entrées et ne se bloque pas.

Pour identifier les éléments à optimiser et déterminer l'utilité des optimisations, il est essentiel d'effectuer des tests de performances. Vous trouverez des techniques et des conseils à ce sujet à la section « Test de performances et déploiement » à la page 97.

Pour plus d'informations sur la façon de déterminer les parties d'une application qu'il serait judicieux d'optimiser, voir les ressources suivantes :

- Performance-tuning apps for AIR à l'adresse http://www.adobe.com/go/learn_fp_goldman_tv_fr (Enregistrement vidéo de la présentation d'Oliver Goldman lors de la conférence MAX. Disponible en anglais uniquement)
- Performance-tuning Adobe AIR applications à l'adresse http://www.adobe.com/go/learn_fp_air_perf_devnet_fr (Article d'Oliver Goldman, fondé sur la présentation, sur Adobe Developer Connection. Disponible en anglais uniquement.)

Chapitre 2 : Conservation de mémoire

Lors du développement d'applications, y compris les applications de bureau, il est toujours important de conserver la mémoire. Les périphériques mobiles sont toutefois particulièrement gourmands en mémoire et il est souhaitable de limiter la quantité de mémoire que consomme une application.

Objets d'affichage

Sélectionnez un objet d'affichage approprié.

ActionScript 3.0 propose un grand nombre d'objets d'affichage. Une des plus simples techniques d'optimisation visant à limiter la consommation de mémoire consiste à choisir le type approprié d'objet d'affichage. Pour créer des formes simples qui ne sont pas interactives, utilisez les objets Shape. Pour créer des objets interactifs ne nécessitant pas de scénario, faites appel aux objets Spirite. Pour une animation s'appuyant sur un scénario, recourez aux objets MovieClip. Choisissez toujours le type d'objet le plus performant pour l'application.

Le code suivant indique la quantité de mémoire utilisée par différents objets d'affichage :

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

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

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

La méthode `getSize()` indique la quantité de mémoire, exprimée en nombre d'octets, que consomme un objet. Vous pouvez constater que l'utilisation de plusieurs objets MovieClip, plutôt que des objets Shape simples, peut gaspiller de la mémoire si les fonctionnalités d'un objet MovieClip ne sont pas nécessaires.

Types de primitives

La méthode `getSize()` permet de tester les performances du code et de déterminer l'objet le plus adapté à la tâche concernée.

Tous les types de primitives, à l'exception de String, requièrent 4 à 8 octets de mémoire. Aucun type spécifique de primitive ne permet d'optimiser l'utilisation de la mémoire :

```
// 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
```

Si vous n'attribuez pas de valeur à une primitive Number, qui représente une valeur 64 bits, la machine virtuelle ActionScript (AVM) lui alloue 8 octets. Tous les autres types de primitives sont stockés dans 4 octets.

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

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

Le comportement du type String est différent. La quantité d'espace de stockage alloué est fonction de la longueur de la chaîne :

```
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
```

La méthode `getSize()` permet de tester les performances du code et de déterminer l'objet le plus adapté à la tâche concernée.

Réutilisation d'objets


Réutilisez des objets, si possible, au lieu de les recréer.

Une autre technique simple d'optimisation de la mémoire consiste à réutiliser les objets pour éviter, dans la mesure possible, de les recréer. N'utilisez pas le code suivant dans une boucle, par exemple :

```
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);
}
```

Le fait de recréer l'objet Rectangle dans chaque itération de la boucle utilise plus de mémoire et est une procédure plus lente, car un objet est créé à chaque itération. Procédez plutôt comme suit :

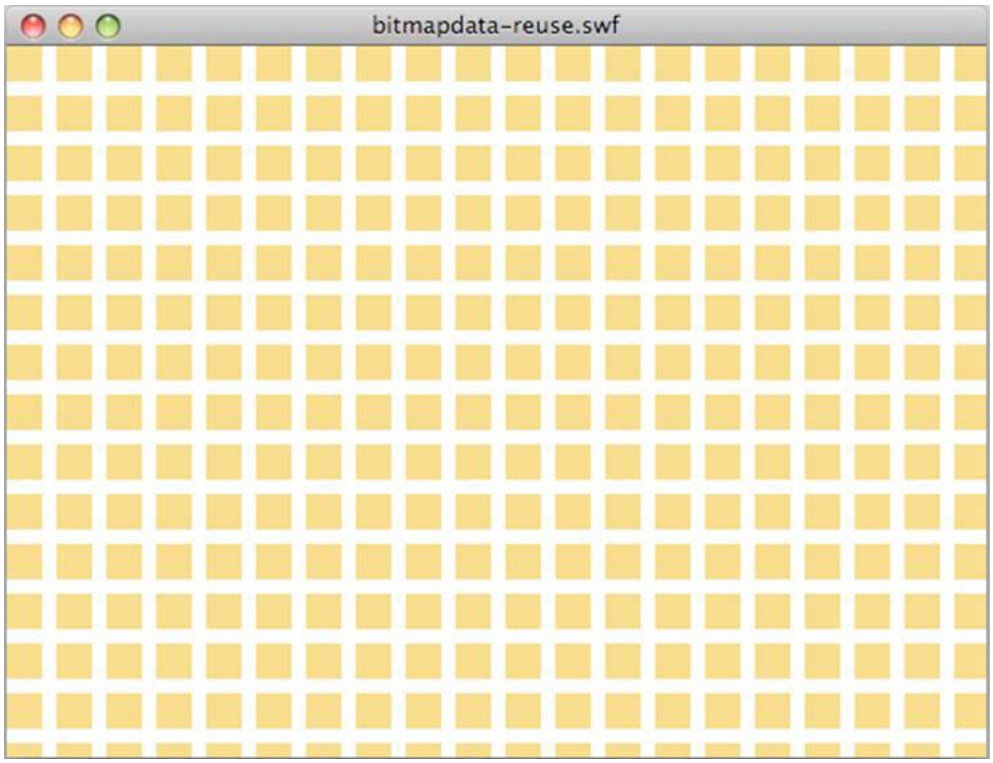
```
const MAX_NUM:int = 18;
const COLOR:uint = 0xCCCCCC;

// 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);
}
```

L'exemple précédent reposait sur un objet dont l'impact sur la mémoire était relativement faible. L'exemple suivant montre comment conserver encore plus de mémoire en réutilisant un objet BitmapData. Le code ci-dessous, qui crée un effet de mosaïque, gaspille de la mémoire :

```
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);
}
```

Remarque : *lorsque vous utilisez des valeurs positives, il est beaucoup plus rapide d'associer la valeur arrondie à int que d'utiliser la méthode* `Math.floor()`.

L'image suivante illustre le résultat de l'effet de mosaïque sur le bitmap :



Effet de mosaïque résultant

Une version optimisée crée une occurrence unique de BitmapData référencée par plusieurs occurrences de Bitmap et donne le même résultat :

```
// 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);
}
```

Cette technique conserve environ 700 Ko de mémoire, ce qui est considérable sur un périphérique mobile standard. Les propriétés de Bitmap permettent de manipuler tout conteneur de bitmap sans incidence sur l'occurrence de BitmapData originale :

```
// 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'image suivante illustre le résultat de l'effet des transformations du bitmap :

Résultat des transformations du bitmap

Voir aussi

« Mise en cache sous forme de bitmap » à la page 55

Pool d'objets

Faites appel à la technique de pool d'objets, dans la mesure du possible.

Le pool d'objets, qui consiste à réutiliser des objets à terme, est une autre technique d'optimisation importante. Vous créez un nombre défini d'objets lors de l'initialisation de l'application et les enregistrez dans un pool, tel qu'un objet Array ou Vector. Lorsque vous en avez terminé avec un objet, vous le désactivez pour éviter qu'il ne consomme des ressources de l'unité centrale et vous supprimez toutes les références mutuelles. Vous ne définissez toutefois pas les références sur `null`, ce qui rendrait l'objet éligible pour le processus de nettoyage de la mémoire. Vous vous contentez de lui faire réintégrer le pool et de l'en extraire lorsque vous avez besoin d'un nouvel objet.

Lorsque vous réutilisez des objets, il n'est pas autant nécessaire de les instancier, processus gourmand en ressources. Elle limite également le déclenchement du nettoyeur de mémoire, qui est susceptible de ralentir l'application. Le code suivant illustre la technique de pool d'objets :

```
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 crée un pool de nouveaux objets lors de l'initialisation de l'application. La méthode `getSprite()` renvoie des occurrences de ces objets tandis que la méthode `disposeSprite()` les libère. Le code autorise l'expansion du pool une fois celui-ci entièrement consommé. Il est également possible de créer un pool de taille fixe, qui une fois épuisé, interdit l'allocation de nouveaux objets. Evitez si possible de créer des objets dans des boucles. Pour plus d'informations, voir « Libération de mémoire » à la page 12. Dans le code suivant, la classe SpritePool extrait de nouvelles occurrences :

```
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 );
    }
}
```

Le code suivant supprime tous les objets de la liste d'affichage lorsque l'utilisateur clique sur la souris et les réutilise ultérieurement à d'autres fins :

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

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

Remarque : *le vecteur de pool fait toujours référence aux objets Sprite. Si vous souhaitez définitivement supprimer l'objet de la mémoire, appliquez la méthode* `dispose()` *à la classe SpritePool : elle efface toutes les références restantes.*

Libération de mémoire

Supprimez toutes les références aux objets pour activer le déclenchement du nettoyage de la mémoire.

Il est impossible de démarrer directement le nettoyeur de mémoire dans la version commerciale de Flash Player. Pour être certain qu'un objet est collecté par le nettoyeur, supprimez toutes ses références. Rappelez-vous que l'opérateur `delete` d'ActionScript 1.0 et 2.0 se comporte différemment dans ActionScript 3.0. Il permet uniquement de supprimer des propriétés dynamiques sur un objet dynamique.

Remarque : *vous pouvez appeler directement le nettoyeur de mémoire dans Adobe® AIR® et dans la version de débogage de Flash Player.*

Le code suivant, par exemple, définit une référence Sprite sur `null` :

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

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

Un objet défini sur `null` n'est pas nécessairement supprimé de la mémoire. Il arrive que le nettoyeur de mémoire ne s'exécute pas si la quantité de mémoire disponible est considérée comme suffisante. Le nettoyage de la mémoire est un processus imprévisible. L'affectation de mémoire, et non la suppression d'objets, déclenche le nettoyage de la mémoire. Lorsqu'il s'exécute, il détecte des graphes d'objets qui n'ont pas encore été nettoyés. Il détecte les objets inactifs dans ces graphes en identifiant les objets qui se font référence mais que l'application n'utilise plus, et les supprime.

Dans une application de grande taille, ce processus, qui est susceptible de solliciter fortement l'unité centrale, peut affecter les performances et entraîner un ralentissement notable de l'application. Réutilisez autant que possible les objets pour essayer de réduire le nombre d'exécutions du nettoyeur de mémoire. Définissez également les références sur null, le cas échéant, afin que le nettoyeur consacre moins de temps de traitement à rechercher les objets. Vous pourriez envisager le nettoyage de la mémoire comme une assurance : gérez donc la durée de vie des objets de manière explicite et systématique, dans la mesure du possible.

Remarque : *définir une référence à un objet d'affichage sur null ne garantit pas le figement de l'objet. L'objet continue de consommer les ressources de l'unité centrale jusqu'à ce qu'il soit nettoyé. Veillez à désactiver votre objet avant de définir sa référence sur* `null`.

Vous pouvez lancer le nettoyeur de mémoire à l'aide de la méthode `System.gc()`, que proposent Adobe AIR et la version de débogage de Flash Player. Le profileur livré avec Adobe® Flash® Builder™ permet de lancer manuellement le nettoyeur de mémoire. L'exécution de ce dernier permet de vérifier le comportement de l'application et de déterminer si les objets sont correctement supprimés de la mémoire.

Remarque : *tout objet servant d'écouteur d'événements peut être référencé par un autre objet. Dans ce cas, supprimez les écouteurs d'événements à l'aide de la méthode* `removeEventListener()` *avant de définir les références sur* `null`.

Il est heureusement possible de réduire instantanément la quantité de mémoire utilisée par les bitmaps. La classe BitmapData, par exemple, possède une méthode `dispose()`. L'exemple qui suit crée une occurrence de BitmapData de 1,8 Mo. La mémoire utilisée actuellement atteint 1,8 Mo et la propriété `System.totalMemory` renvoie une valeur inférieure :

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

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

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

L'occurrence de BitmapData est ensuite manuellement supprimée de la mémoire, qui est à nouveau vérifiée :

```
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
```

Bien que la méthode `dispose()` supprime les pixels de la mémoire, il est néanmoins nécessaire de définir la référence sur `null` pour la libérer totalement. Appelez systématiquement la méthode `dispose()` et définissez la référence sur `null` lorsque vous n'avez plus besoin d'un objet BitmapData afin de libérer immédiatement la mémoire.

Remarque : *la classe System de Flash Player 10.1 et d'AIR 1.5.2 comporte une nouvelle méthode, `disposeXML()`. Cette méthode vous permet de mettre immédiatement un objet XML à la disposition du nettoyeur de mémoire, en transmettant l'arborescence XML en tant que paramètre.*

Voir aussi

« Figement et libération d'objets » à la page 28

Utilisation des bitmaps

L'utilisation de vecteurs et non de bitmaps est un bon moyen d'économiser de la mémoire. Cependant, les vecteurs sollicitent beaucoup l'unité centrale et le processeur graphique, particulièrement s'ils sont nombreux. Les bitmaps, quant à eux, permettent d'optimiser le rendu, car le moteur d'exécution nécessite moins de ressources de traitement pour dessiner des pixels à l'écran que pour effectuer le rendu du contenu des vecteurs.

Voir aussi

« Mise en cache manuelle sous forme de bitmap » à la page 63

Sous-échantillonnage des bitmaps

Pour assurer une meilleure utilisation de la mémoire, les images opaques de 32 bits sont réduites en images de 16 bits lorsque Flash Player détecte un écran 16 bits. Ce sous-échantillonnage nécessite la moitié des ressources de mémoire et le rendu des images est plus rapide. Cette fonction est uniquement disponible dans Flash Player 10.1 pour Windows Mobile.

Remarque : *avant Flash Player 10.1, tous les pixels créés en mémoire étaient stockés dans 32 bits (4 octets). Un simple logo de 300 x 300 pixels exigeait 350 Ko de mémoire (300*300*4/1024). Grâce à ce nouveau comportement, le même logo opaque requiert uniquement 175 Ko. Si le logo est transparent, il n'est pas sous-échantillonné à 16 bits et sa taille en mémoire ne change pas. Cette fonction s'applique uniquement aux bitmaps intégrés ou aux images chargées à l'exécution (PNG, GIF, JPG).*

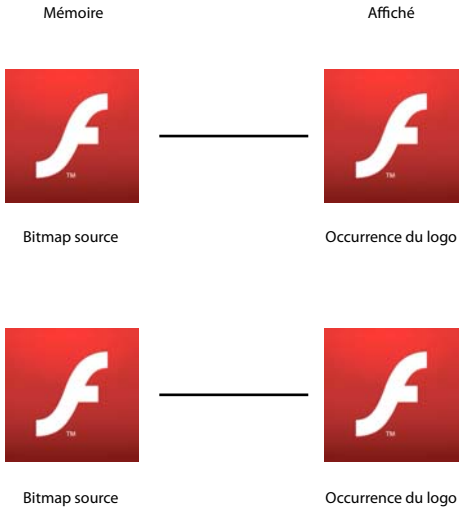
Sur les périphériques mobiles, il est parfois difficile de faire la différence entre le rendu d'une image en 16 bits ou en 32 bits. Sur une image simple ne comportant que quelques couleurs, aucune différence n'est détectable. Même sur une image plus complexe, les différences sont peu notables. Une certaine dégradation des couleurs peut cependant se produire lorsque l'utilisateur effectue un zoom avant sur l'image, et un dégradé de 16 bits est moins régulier que la version 32 bits.

Référence unique à BitmapData

Il est important d'optimiser l'utilisation de la classe BitmapData en réutilisant les occurrences autant que faire se peut. Flash Player 10.1 et AIR 2.5 proposent une nouvelle fonction de référence unique à BitmapData sur toutes les plates-formes. Lors de la création d'occurrences de BitmapData à partir d'une image intégrée, une version unique du bitmap est utilisée pour toutes ces occurrences. Si le bitmap est modifié par la suite, un bitmap qui lui est propre lui est attribué en mémoire. L'image intégrée peut provenir de la bibliothèque ou d'une balise [Embed].

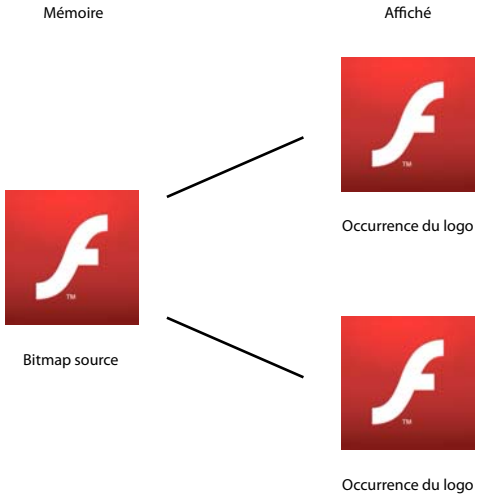
Remarque : *cette nouvelle fonction présente également des avantages pour le contenu existant, car Flash Player 10.1 et AIR 2.5 réutilisent automatiquement les bitmaps.*

Lors de l'instanciation d'une image intégrée, un bitmap associé est créé en mémoire. Préalablement à Flash Player 10.1 et AIR 2.5, à chaque occurrence était attribué son propre bitmap en mémoire, comme illustré ci-dessous :



Bitmaps en mémoire avant Flash Player 10.1 et AIR 2.5

Dans Flash Player 10.1 et AIR 2.5, lors de la création de plusieurs occurrences d'une même image, une version unique du bitmap est utilisée pour toutes les occurrences de BitmapData. Ce concept est illustré ci-dessous :

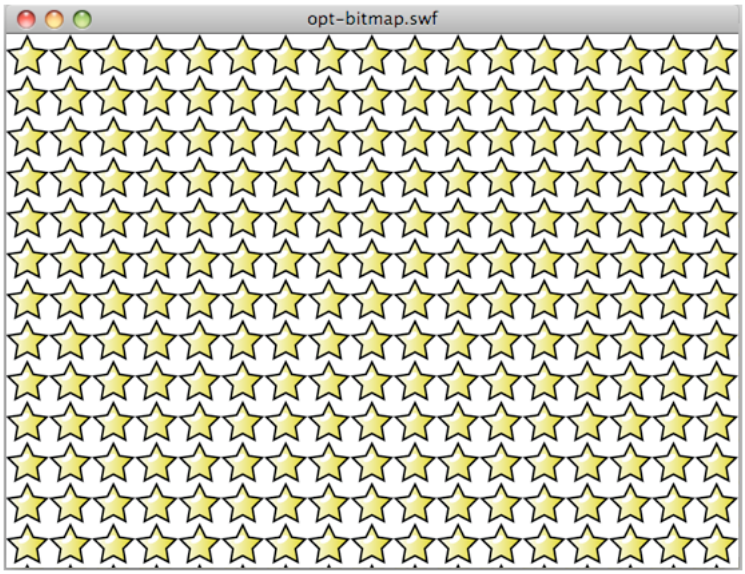


Bitmaps en mémoire dans Flash Player 10.1 et AIR 2.5

Cette technique réduit considérablement la quantité de mémoire que nécessite une application utilisant de nombreux bitmaps. Le code suivant crée plusieurs occurrences du symbole `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)
    }
}
```

La figure suivante illustre le résultat du code :



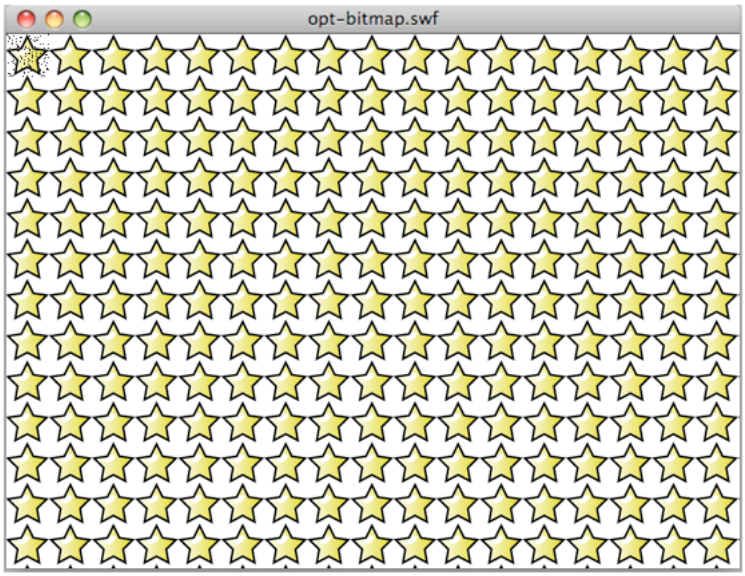
Résultat du code utilisé pour créer plusieurs occurrences du symbole

Avec Flash Player 10, par exemple, l'animation ci-dessus utilise environ 1 008 Ko de mémoire. Avec Flash Player 10.1, l'animation utilise seulement 4 Ko, que l'application soit installée sur un ordinateur de bureau ou sur un périphérique mobile.

Le code suivant modifie une seule occurrence de BitmapData :

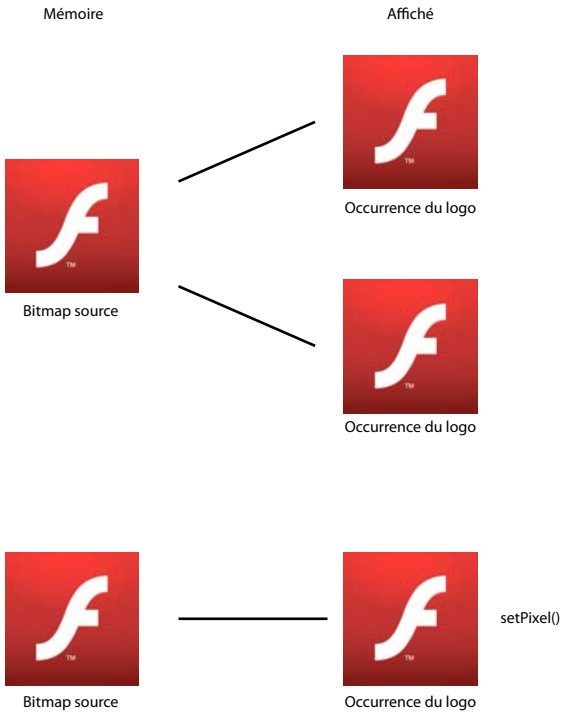
```
const MAX_NUM:int = 18;

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);
```

La figure suivante illustre le résultat de la modification d'une seule occurrence de `Star` :



Résultat de la modification d'une occurrence unique

En interne, le moteur d'exécution attribue et crée automatiquement un bitmap en mémoire pour gérer les modifications au niveau des pixels. L'appel d'une méthode de la classe BitmapData entraîne la modification des pixels et la création d'une occurrence en mémoire, et aucune autre occurrence n'est mise à jour. La figure suivante illustre ce concept :



Résultat en mémoire de la modification d'un bitmap unique

Si une étoile est modifiée, une nouvelle copie est créée dans la mémoire. L'animation résultante utilise environ 8 Ko de mémoire dans Flash Player 10.1 et AIR 2.5.

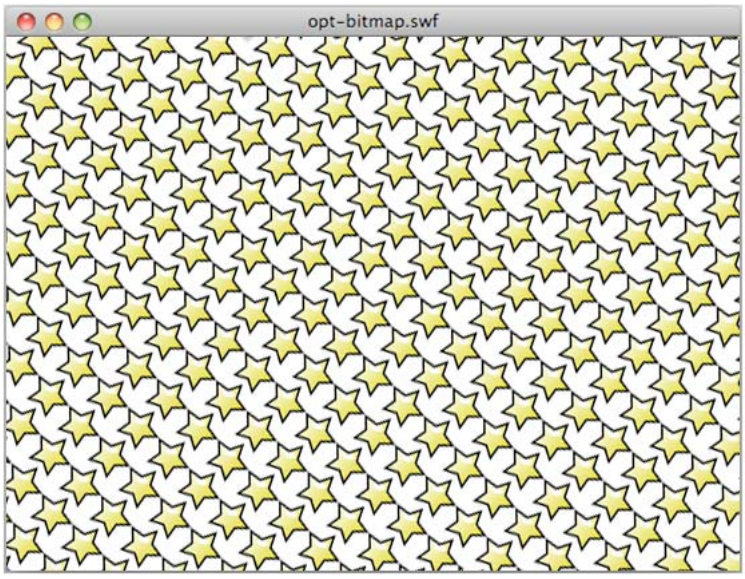
Dans l'exemple précédent, chaque bitmap peut être transformé individuellement. Pour créer uniquement l'effet de mosaïque, la méthode `beginBitmapFill()` est la plus adaptée :

```
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);
```

Cette technique donne le même résultat et une seule occurrence de BitmapData est créée. Pour imprimer une rotation permanente aux étoiles, plutôt que d'accéder à chaque occurrence de Star, utilisez un objet Matrix pivotant sur chaque image et transmettez-le à la méthode `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);
}
```

Il est ainsi inutile de recourir à une boucle ActionScript pour créer l'effet. Le moteur d'exécution effectue toutes les opérations en interne. La figure suivante illustre le résultat de la transformation des étoiles :



Résultat de la rotation des étoiles

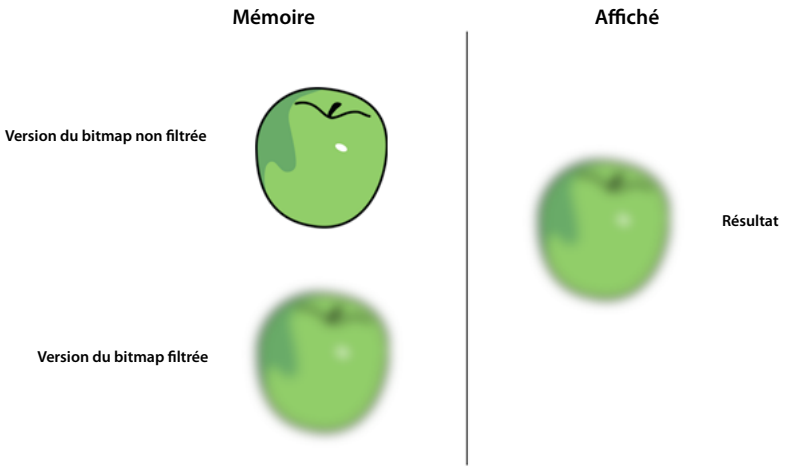
La mise à jour de l'objet BitmapData source original est automatiquement répercutée sur son utilisation à un autre emplacement sur la scène. Cette technique peut se révéler très performante. Elle ne permet pas cependant de mettre individuellement à l'échelle chaque étoile, comme dans l'exemple précédent.

Remarque : *lors de l'utilisation de plusieurs occurrences d'une même image, le dessin varie selon qu'une classe est associée au bitmap original en mémoire. Si aucune classe n'est associée au bitmap, les images sont dessinées en tant qu'objets Shape avec des remplissages de bitmap.*

Filtres et déchargement dynamique de bitmaps

Evitez les filtres, y compris ceux traités par le biais de Pixel Bender.

Utilisez un minimum d'effets tels que les filtres, y compris ceux traités sur les périphériques mobiles par le biais de Pixel Bender. Lors de l'application d'un filtre à un objet d'affichage, le moteur d'exécution crée deux bitmaps en mémoire, chacun de la taille de l'objet d'affichage. Le premier est une version pixellisée de l'objet d'affichage et sert à créer le second, auquel le filtre est appliqué :



Deux bitmaps en mémoire lors de l'application d'un filtre

Lors de la modification de l'une des propriétés d'un filtre, les deux bitmaps sont mis à jour en mémoire pour créer le bitmap résultant. Ce processus sollicite l'unité centrale et les deux bitmaps peuvent nécessiter beaucoup de mémoire.

Flash Player 10.1 et AIR 2.5 proposent un nouveau comportement de filtrage sur toutes les plates-formes. Si le filtre n'est pas modifié sous 30 secondes, ou s'il est masqué ou hors écran, la mémoire utilisée par le bitmap non filtré est libérée.

Cette fonction économise donc la moitié de la mémoire exigée par un filtre sur toutes les plates-formes. Considérez par exemple un objet de texte auquel est appliqué un filtre de flou. Le texte en question est purement décoratif et ne subit aucune modification. Après 30 secondes, le bitmap non filtré est libéré de la mémoire. Il se produit le même résultat si le texte est masqué pendant 30 secondes ou est hors écran. Lors de la modification de l'une des propriétés du filtre, le bitmap non filtré en mémoire est recréé. Cette fonction s'appelle déchargement dynamique de bitmap. Même avec ces optimisations, utilisez les filtres avec précaution. Leur modification sollicite énormément l'unité centrale ou le processeur graphique.

Il est recommandé de créer des bitmaps dans un outil de création, tel qu'Adobe® Photoshop®, pour émuler les filtres, si possible. Evitez l'utilisation de bitmaps dynamiques créés à l'exécution dans ActionScript. L'utilisation de bitmaps créés en externe permet au moteur d'exécution de réduire la charge de l'unité centrale ou du processeur graphique, surtout si les propriétés du filtre ne changent pas à terme. Si possible, créez les effets requis sur un bitmap dans un outil de création. Vous pouvez ensuite afficher ce bitmap dans le moteur d'exécution sans le traiter, ce qui est beaucoup plus rapide.

Mip-mapping direct

Utilisez le mip-mapping pour mettre à l'échelle les images volumineuses, si besoin est.

Flash Player 10.1 et AIR 2.5 proposent, sur toutes les plates-formes, une autre nouvelle fonction liée au mip-mapping. Flash Player 9 et AIR 1.0 offraient une fonction de mip-mapping qui permettait d'améliorer la qualité et les performances des bitmaps sous-échantillonnés.

Remarque : *le mip-mapping s'applique uniquement aux images chargées dynamiquement ou aux bitmaps intégrés ; il ne s'applique pas aux objets d'affichage filtrés ou mis en cache. Il est uniquement traité si la largeur et la hauteur du bitmap sont des nombres pairs. Si la hauteur ou la largeur est un nombre impair, le mip-mapping s'arrête. Ainsi, il est possible d'appliquer un mip-mapping à une image de 250 x 250 pour la réduire à 125 x 125 au maximum. Dans ce cas, en effet, l'une des dimensions au moins est un nombre impair. Les bitmaps dont les dimensions sont des puissances de deux permettent d'obtenir des résultats optimaux. Exemple : 256 x 256, 512 x 512, 1024 x 1024, etc.*

Imaginons qu'une image de 1024 x 1024 est chargée et qu'un développeur souhaite la mettre à l'échelle pour créer une vignette dans une galerie. La fonction de mip-mapping effectue correctement le rendu de l'image lors de la mise à l'échelle en utilisant les versions sous-échantillonnées intermédiaires du bitmap en tant que textures. Les versions antérieures du moteur d'exécution créaient des versions sous-échantillonnées intermédiaires du bitmap en mémoire. Si une image de 1024 x 1024 était chargée et affichée à 64 x 64, les anciennes versions du moteur d'exécution créaient un bitmap pour chaque demi-taille ; dans notre exemple, 512 x 512, 256 x 256, 128 x 128 et 64 x 64.

Flash Player 10.1 et AIR 2.5 permettent à présent d'effectuer un mip-mapping direct de la taille originale à la taille de destination. Dans l'exemple précédent, seuls seraient créés le bitmap original de 4 Mo (1024 x 1024) et le bitmap de 16 Ko (64 x 64) auquel un mip-mapping a été appliqué.

La logique de mip-mapping est également compatible avec le déchargement dynamique de bitmap. Si seul le bitmap de 64 x 64 est utilisé, l'original de 4 Mo est vidé de la mémoire. S'il est nécessaire de répéter le mip-mapping, cet original est rechargé. Par ailleurs, si d'autres bitmaps de plusieurs tailles ayant fait l'objet d'un mip-mapping sont nécessaires, la chaîne de mip-mapping intermédiaire est utilisée pour les créer. S'il est nécessaire de créer un bitmap au 1:8, par exemple, les bitmaps au 1:4, 1:2 et 1:1 sont examinés pour déterminer lequel d'entre eux est chargé en mémoire en premier. En l'absence des autres versions, le bitmap original, au 1:1, est chargé à partir de la ressource et utilisé.

Le décompresseur JPEG peut effectuer un mip-mapping dans son propre format. Ce mip-mapping direct permet de décompresser directement un bitmap de grande taille vers un format mip-map sans charger intégralement l'image non compressée. La génération du mip-map est considérablement plus rapide. La mémoire exigée par les bitmaps volumineux n'est pas allouée et, donc, libérée. La qualité d'image JPEG est comparable à la technique de mip-mapping générale.

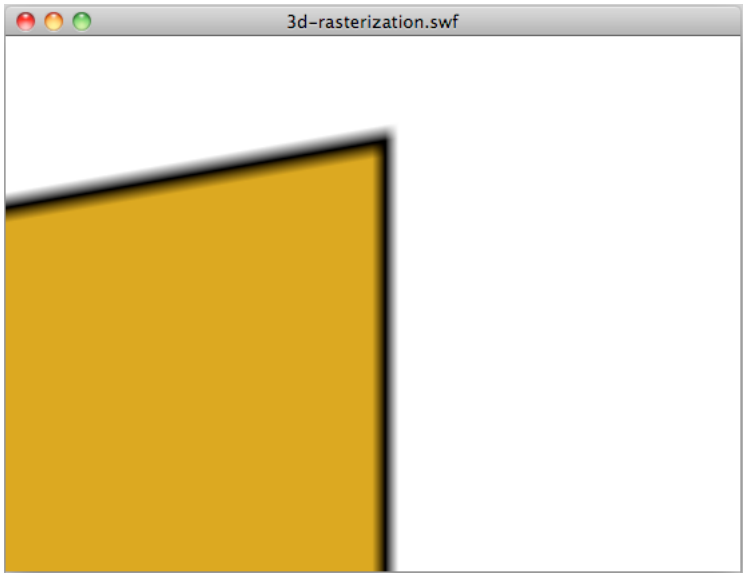
Remarque : *évitez une utilisation excessive du mip-mapping. Bien qu'il améliore la qualité des images téléchargées, il n'est pas sans incidence sur la bande passante, la mémoire et la vitesse. Dans certains cas, il est préférable de mettre à l'échelle une version du bitmap dans un outil externe et de l'importer dans votre application. Ne créez pas des bitmaps de grande taille si vous allez les réduire plus tard.*

Utilisation des effets 3D

Envisagez de créer manuellement les effets 3D.

Flash Player 10 et AIR 1.5 ont introduit un nouveau moteur 3D, qui permet d'appliquer des transformations de perspective aux objets d'affichage. Vous pouvez appliquer ces transformations à l'aide des propriétés `rotationX` et `rotationY` ou de la méthode `drawTriangles()` de la classe Graphics. Vous pouvez aussi définir un effet de profondeur à l'aide de la propriété `z`. Gardez à l'esprit que chaque objet d'affichage auquel une transformation de perspective est appliquée est pixellisé en tant que bitmap et exige donc plus de mémoire.

La figure suivante illustre l'anticrènelage résultant de la pixellisation lors de l'application d'une transformation de perspective :



Anticrènelage résultant de l'application d'une transformation de perspective

L'anticrènelage est produit par la pixellisation dynamique sous forme de bitmap du contenu du vecteur. L'anticrènelage est appliqué lors de l'utilisation d'effets 3D dans la version de bureau d'AIR et de Flash Player, et dans la version mobile d'AIR 2.0.1 et d'AIR 2.5. L'anticrènelage n'est toutefois pas appliqué dans la version mobile de Flash Player.

Vous pouvez économiser de la mémoire s'il vous est possible de créer l'effet 3D manuellement sans faire appel à l'API native. Toutefois, les nouvelles fonctions 3D introduites dans Flash Player 10 et AIR 1.5 facilitent le mappage des textures, car les méthodes comme `drawTriangles()` gèrent ce processus en natif.

Il vous appartient, en tant que développeur, de décider si les performances de l'effet 3D recherché sont optimisées par un traitement manuel ou via l'API native. Outre les considérations liées à la mémoire, tenez également compte des performances d'exécution et de rendu d'ActionScript.

Dans les applications mobiles d'AIR 2.0.1 et d'AIR 2.5 dans lesquelles vous définissez la propriété `renderMode` sur `GPU`, c'est le processeur graphique qui se charge des transformations 3D. En revanche, si la propriété `renderMode` est définie sur `CPU`, c'est l'unité centrale (et non le processeur graphique) qui effectue les transformations 3D. Dans les applications de Flash Player 10.1, c'est l'unité centrale qui effectue les transformations 3D.

Lorsque l'unité centrale effectue les transformations 3D, tenez compte du fait que l'application d'une transformation 3D à un objet d'affichage nécessite deux bitmaps en mémoire. Un bitmap est nécessaire pour le bitmap source et un autre pour la version à laquelle une transformation de perspective est appliquée. Le comportement des transformations 3D est similaire à celui des filtres. Par conséquent, utilisez les propriétés 3D avec modération lorsque c'est l'unité centrale qui effectue les transformations 3D.

Objets de texte et mémoire

Utilisez Adobe® Flash® Text Engine pour le texte en lecture seule et des objets TextFields pour le texte de saisie.

Flash Player 10 et AIR 1.5 ont introduit un nouveau moteur de texte puissant, Adobe Flash Text Engine (FTE), qui permet de conserver la mémoire système. Ce moteur est néanmoins une API de bas niveau qui nécessite une couche ActionScript 3.0 supplémentaire, fournie dans le package flash.text.engine.

Pour le texte en lecture seule, il est préférable d'utiliser Flash Text Engine, qui est peu gourmand en mémoire et offre un meilleur rendu. Les objets TextFields, quant à eux, sont plus adaptés au texte de saisie, car ils nécessitent moins de code ActionScript pour créer des comportements standard tels que la gestion de la saisie et le retour à la ligne.

Voir aussi

« Rendu des objets de texte » à la page 69

Modèle d'événement et rappels

Envisagez d'utiliser de simples rappels plutôt que le modèle d'événement.

Le modèle d'événement d'ActionScript 3.0 est fondé sur le concept de distribution d'objets. Il est orienté objet et optimisé pour la réutilisation du code. La méthode `dispatchEvent()` effectue une boucle dans la liste d'écouteurs et appelle la méthode du gestionnaire d'événement sur chaque objet enregistré. Ce modèle présente cependant un inconvénient : vous risquez en effet de créer un grand nombre d'objets tout au long de la vie de votre application.

Imaginons que vous devez distribuer un événement à partir du scénario, pour indiquer la fin d'une séquence d'animation. Pour accomplir la notification, vous pouvez distribuer un événement à partir d'une image spécifique du scénario, comme illustré dans le code suivant :

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

La ligne de code suivante permet à la classe Document d'écouter cet événement :

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

Bien que cette technique soit correcte, l'utilisation du modèle d'événement natif peut être plus lente et exiger plus de mémoire qu'une fonction de rappel classique. Il est nécessaire de créer des objets d'événement et de les affecter en mémoire, ce qui ralentit les performances. Lorsque vous écoutez l'événement Event.ENTER_FRAME, par exemple, un objet d'événement est créé sur chaque image pour le gestionnaire d'événement. Les performances des objets d'affichage peuvent être particulièrement lentes, en raison des phases de capture et de propagation, ce qui se traduit par une forte sollicitation des ressources si la liste d'affichage est complexe.

Chapitre 3 : Réduction de la charge de l'unité centrale

En matière d'optimisation, il est également impératif de tenir compte de l'utilisation de l'unité centrale. L'optimisation du traitement par l'unité centrale améliore les performances et, par conséquent, la durée de vie de la batterie des périphériques mobiles.

Améliorations de Flash Player 10.1 liées à l'utilisation de l'unité centrale

Flash Player 10.1 propose deux nouvelles fonctions qui permettent d'économiser la puissance de traitement de l'unité centrale, à savoir la mise en pause et la reprise du contenu SWF lorsque ce dernier sort de l'écran, et la limitation du nombre d'occurrences de Flash Player sur une page.

Pause, ralentissement et reprise

Remarque : *la fonction de pause, ralentissement et reprise n'est pas disponible sur les applications Adobe® AIR®.*

Pour optimiser l'utilisation de l'unité centrale et de la batterie, Flash Player 10.1 propose une nouvelle fonction relative aux occurrences inactives. Cette fonction permet de limiter l'utilisation de l'unité centrale en mettant en pause et en reprenant le fichier SWF lorsque le contenu sort de l'écran et y revient. Grâce à cette fonction, Flash Player libère autant de mémoire que possible en supprimant tout objet pouvant être recréé lorsque la lecture du contenu reprend. Est considéré comme hors écran, tout contenu qui est totalement sorti de l'écran.

Deux cas de figure entraînent la sortie de l'écran du contenu SWF :

- L'utilisateur fait défiler la page et le contenu SWF sort de l'écran.

  Dans ce cas, tout contenu audio ou vidéo en cours de lecture continue de s'exécuter mais le rendu est interrompu. Si aucun contenu audio ou vidéo n'est en cours de lecture, pour parer à toute mise en pause de la lecture ou de l'exécution d'ActionScript, définissez le paramètre HTML `hasPriority` sur true. Souvenez-vous cependant que, lorsque le contenu SWF est hors écran ou masqué, son rendu est mis en pause, quelle que soit la valeur du paramètre `hasPriority`.

- L'utilisateur ouvre un onglet dans le navigateur, faisant passer à l'arrière-plan le contenu SWF.

  Dans ce cas, quelle que soit la valeur de la balise HTML `hasPriority`, le contenu SWF est *ralenti* ou réduit de 2 à 8 ips. La lecture audio et vidéo est arrêtée et aucun rendu n'est traité à moins que le contenu SWF ne soit à nouveau visible.

Pour Flash Player 11.2 et les versions ultérieures exécutées sur des navigateurs Windows et Mac, vous pouvez utiliser l'événement ThrottleEvent dans votre application. Flash Player distribue un événement ThrottleEvent lorsqu'il interrompt, ralenti ou reprend la lecture.

L'événement ThrottleEvent est un événement de diffusion, c'est-à-dire qu'il est distribué par tous les objets EventDispatcher disposant d'un écouteur de cet événement. Pour plus d'informations sur les événements de diffusion, voir la classe DisplayObject.

Gestion des occurrences

Remarque : *la fonction de gestion des occurrences ne s'applique pas aux applications Adobe® AIR®.*

Utilisez le paramètre HTML `hasPriority` *pour retarder le chargement des fichiers SWF hors écran.*

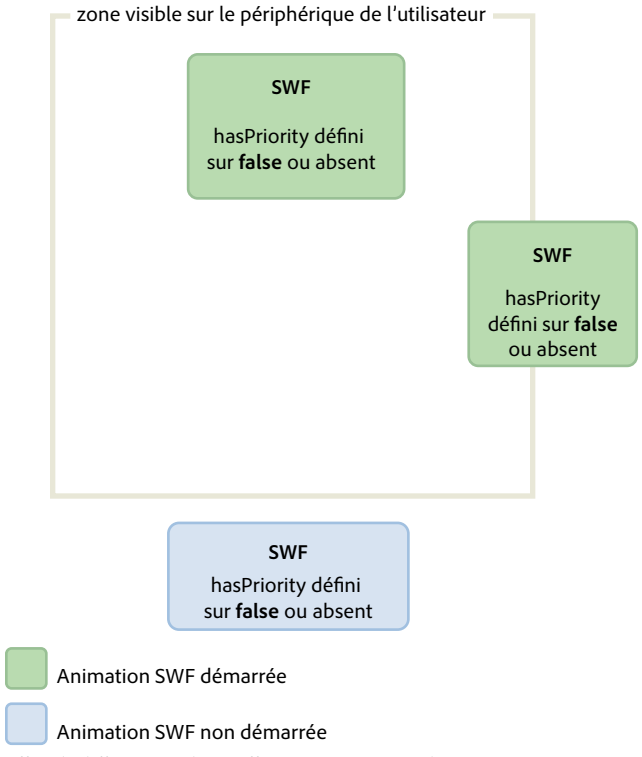
Flash Player 10.1 propose un nouveau paramètre HTML, `hasPriority` :

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

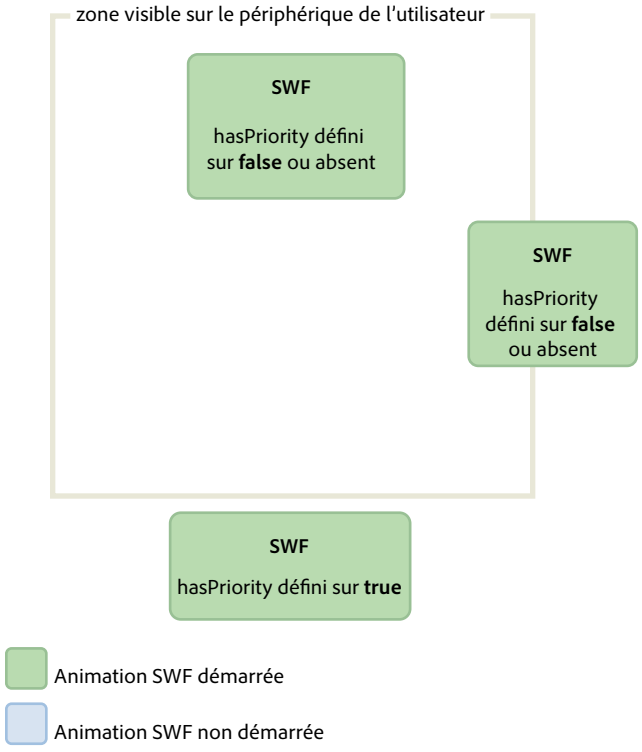
Cette fonction limite le nombre d'occurrences de Flash Player démarrées sur une page, ce qui permet de conserver les ressources de l'unité centrale et de la batterie. Il s'agit d'attribuer une priorité spécifique au contenu SWF, de sorte qu'un certain contenu soit prioritaire sur une page. Prenons un exemple simple : un utilisateur explore un site Web dont la page d'index héberge trois fichiers SWF différents. Le premier est entièrement visible, le deuxième l'est partiellement et le dernier est hors écran et ne peut être atteint que par défilement. Les deux premières animations démarrent normalement mais la troisième est différée jusqu'à ce qu'elle devienne visible. C'est le comportement par défaut lorsque le paramètre `hasPriority` est absent ou défini sur `false`. Pour s'assurer du démarrage d'un fichier SWF, même s'il est hors écran, définissez le paramètre `hasPriority` sur `true`. Toutefois, quelle que soit la valeur du paramètre `hasPriority`, le rendu d'un fichier SWF qui n'est pas visible pour l'utilisateur est toujours mis en pause.

Remarque : *si les ressources de l'unité centrale sont insuffisantes, les occurrences de Flash Player ne démarrent plus automatiquement, même si le paramètre* `hasPriority` *correspond à* `true`*. Les nouvelles occurrences éventuellement créées par le biais de JavaScript après le chargement de la page ne tiennent pas compte de l'indicateur* `hasPriority`*. Tout contenu de 1 x 1 ou 0 x 0 pixel démarre, empêchant le report du lancement des fichiers SWF d'aide si le webmestre omet l'indicateur* `hasPriority`*. Il est cependant toujours possible de démarrer les fichiers SWF en cliquant dessus. Ce comportement est appelé « cliquer pour lire ».*

Le schéma ci-dessous illustre l'effet de la définition du paramètre `hasPriority` sur différentes valeurs :



Effets de différentes valeurs affectées au paramètre hasPriority



Effets de différentes valeurs affectées au paramètre hasPriority

Mode veille

Flash Player 10.1 et AIR 2.5 offrent, sur les périphériques mobiles, une nouvelle fonction qui permet d'économiser la puissance de traitement et, par conséquent, de prolonger la durée de vie des batteries. Cette fonction est liée à la fonction de rétroéclairage des périphériques mobiles. Par exemple, si un utilisateur exécutant une application mobile est interrompu et cesse d'utiliser son périphérique, le moteur d'exécution détecte le passage en mode veille du rétroéclairage. Il réduit alors la cadence à 4 images par seconde et interrompt le rendu. Pour les applications AIR, le mode veille est également activé lorsque l'application passe en arrière-plan.

Le code ActionScript continue à s'exécuter en mode veille, ce qui revient à définir la propriété `Stage.frameRate` sur 4 images par seconde. Cependant comme l'étape de rendu est omise, l'utilisateur ne se rend pas compte de la cadence. Le choix s'est arrêté sur 4 images par seconde, et non sur zéro, pour que toutes les connexions (NetStream, Socket et NetConnection) restent ouvertes. Une cadence nulle entraînerait la fermeture de toutes les connexions ouvertes. De même, la fréquence d'actualisation est définie sur 250 ms (4 images par seconde), car c'est celle qu'utilisent de nombreux fabricants de périphériques. Cette valeur permet donc d'adapter la cadence du moteur d'exécution au réglage du périphérique.

Remarque : *lorsque le moteur d'exécution est en mode veille, la propriété* `Stage.frameRate` *renvoie la cadence du fichier SWF d'origine, et non 4 ips.*

Lors de la réactivation du rétroéclairage, le rendu reprend et la cadence originale est rétablie. Imaginons qu'utilisateur écoute de la musique sur un lecteur multimédia. Si l'écran passe en mode veille, le comportement du moteur d'exécution dépend du type de contenu en cours de lecture. Voici une liste des divers cas de figure possibles et des comportements correspondants du moteur d'exécution :

- Le rétroéclairage passe en mode veille et le contenu en cours de lecture est de type non-A/V : le rendu est mis en pause et la cadence est définie sur 4 images par seconde.
- Le rétroéclairage passe en mode veille et le contenu en cours de lecture est de type A/V : le moteur d'exécution empêche la mise en veille du rétroéclairage et rien ne change pour l'utilisateur.
- Le rétroéclairage quitte le mode veille : le moteur d'exécution définit la cadence sur le paramètre correspondant du fichier SWF original et reprend le rendu.
- Flash Player est mis en pause pendant la lecture de contenu A/V : Flash Player rétablit le comportement système par défaut du rétroéclairage, car la lecture du contenu A/V est interrompue.
- Le périphérique mobile reçoit un appel pendant la lecture de contenu A/V : le rendu est mis en pause et la cadence est définie sur 4 images par seconde.
- Le mode veille du rétroéclairage est désactivé sur un périphérique mobile : le moteur d'exécution se comporte normalement.

Lorsque le rétroéclairage est mis en veille, le rendu est mis en pause et la cadence ralentit. Si bénéfique soit-elle sur la charge de l'unité centrale, cette fonction ne permet cependant pas de créer une pause réelle, comme dans une application de jeu.

Remarque : *aucun événement ActionScript n'est distribué lorsque le moteur d'exécution passe en mode veille ou le quitte.*

Figement et libération d'objets

Pour figer et libérer des objets correctement, utilisez les événements `REMOVED_FROM_STAGE` *et* `ADDED_TO_STAGE`*.*

Pour optimiser le code, figez et libérez systématiquement les objets. Il est important d'effectuer ces opérations pour tous les objets, et plus particulièrement les objets d'affichage. Même s'ils ne figurent plus sur la liste d'affichage et sont en attente de nettoyage, les objets d'affichage sont toujours susceptibles de solliciter fortement l'unité centrale. Il se peut, par exemple, qu'ils continuent d'utiliser Event.ENTER_FRAME. Il est donc impératif de figer et de libérer correctement les objets à l'aide des événements `Event.REMOVED_FROM_STAGE` et `Event.ADDED_TO_STAGE`. L'exemple suivant illustre un clip qui s'exécute sur la scène et communique avec le clavier :

```
// 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 interagissant avec le clavier

Lorsque l'utilisateur clique sur le bouton Remove (Supprimer), le clip est effacé de la liste d'affichage :

```
// 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);
}
```

Bien que supprimé de la liste d'affichage, le clip distribue toujours l'événement `Event.ENTER_FRAME`. Son exécution se poursuit même si son rendu est interrompu. Pour gérer correctement cette situation, écoutez les événements appropriés et supprimez des écouteurs d'événements, afin d'éviter l'exécution de code sollicitant fortement l'unité centrale :

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

Lorsque l'utilisateur appuie sur le bouton Show (Afficher), le clip redémarre, écoute à nouveau les événements `Event.ENTER_FRAME` et le clavier le contrôle correctement.

Remarque : *si un objet d'affichage est supprimé de la liste d'affichage, il ne suffit pas, pour le figer, de définir sa référence sur* `null` *après sa suppression. Si le nettoyeur de mémoire ne s'exécute pas, l'objet continue de solliciter la mémoire et l'unité centrale même s'il n'est plus affiché. Pour vous assurer que l'objet sollicite le moins possible l'unité centrale, veillez à le figer complètement lors de sa suppression de la liste d'affichage.*

A partir de Flash Player 10 et AIR 1.5, le comportement suivant se produit également. Si la tête de lecture atteint une image vide, l'objet d'affichage est automatiquement figé même si vous n'avez pas implémenté un tel comportement.

Le concept de figement est également important lors du chargement de contenu à distance à l'aide de la classe Loader. Lorsque vous utilisiez cette classe dans Flash Player 9 et AIR 1.0, il était nécessaire de figer manuellement le contenu en écoutant l'événement `Event.UNLOAD` distribué par l'objet LoaderInfo. Il fallait donc figer manuellement tous les objets, une tâche laborieuse. La nouvelle méthode `unloadAndStop()` de la classe Loader constitue une nouveauté importante dans Flash Player 10 et AIR 1.5. Elle permet en effet de décharger un fichier SWF, de figer automatiquement chaque objet du fichier et d'imposer l'exécution du nettoyeur de mémoire.

Dans le code ci-après, le fichier SWF est chargé puis déchargé par le biais de la méthode `unload()`, qui est gourmande en ressources de traitement et exige un figement manuel :

```
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();
}
```

Il est préférable d'utiliser la méthode `unloadAndStop()`, qui gère le figement en natif et impose l'exécution du nettoyeur de mémoire :

```
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();
}
```

L'appel de la méthode `unloadAndStop()` entraîne les actions suivantes :

- Les sons sont arrêtés.
- Les écouteurs enregistrés sur le scénario principal du fichier SWF sont supprimés.
- Les objets Timer sont arrêtés.
- Les périphériques matériels (caméras, microphones, etc.) sont libérés.
- Tous les clips sont arrêtés.
- Les événements `Event.ENTER_FRAME`, `Event.FRAME_CONSTRUCTED`, `Event.EXIT_FRAME`, `Event.ACTIVATE` et `Event.DEACTIVATE` ne sont plus distribués.

Evénements activate et deactivate

Utilisez les événements `Event.ACTIVATE` et `Event.DEACTIVATE` pour détecter l'inactivité de l'arrière-plan et optimiser votre application.

Deux événements (`Event.ACTIVATE` et `Event.DEACTIVATE`) peuvent vous aider à optimiser votre application de façon à ce qu'elle consomme le moins de ressources de l'unité centrale possible. Ces événements permettent de savoir quand le moteur d'exécution possède ou perd le focus. Il est ainsi possible d'optimiser le code de telle sorte qu'il réagisse aux changements de contexte. Le code suivant écoute ces deux événements et définit dynamiquement la cadence sur zéro lorsque l'application perd le focus. Par exemple, l'animation peut perdre le focus lorsque l'utilisateur sélectionne un autre onglet ou fait passer l'application en arrière-plan :

```
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;
}
```

Lorsque l'application retrouve le focus, la cadence reprend sa valeur d'origine. Plutôt que de modifier dynamiquement la cadence, vous pouvez effectuer d'autres optimisations, par exemple le figement ou la libération d'objets.

Les événements activate et deactivate permettent de mettre en oeuvre un mécanisme similaire à la fonction « pause et reprise » parfois disponible sur les périphériques mobiles et les ordinateurs miniportables.

Voir aussi

« Cadence de l'application » à la page 54

« Figement et libération d'objets » à la page 28

Interactions de la souris

Dans la mesure du possible, désactivez les interactions de la souris.

Lors de l'utilisation d'un objet interactif, tel qu'un objet MovieClip ou Sprite, le moteur d'exécution exécute du code natif pour détecter et gérer les interactions de la souris. Lorsque de nombreux objets interactifs sont affichés, ce processus de détection peut solliciter fortement l'unité centrale, surtout si les objets se chevauchent. Pour parer à cet écueil, le plus simple consiste à désactiver les interactions de la souris sur les objets pour lesquels elles ne sont pas nécessaires. Le code suivant illustre l'utilisation des propriétés `mouseEnabled` et `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;
```

Lorsque cela est possible, désactivez les interactions de la souris : votre application fait alors une utilisation moins intensive de l'unité centrale, d'où une consommation réduite de la batterie sur les périphériques mobiles.

Minuteurs et événements ENTER_FRAME

Optez pour des minuteurs ou des événements `ENTER_FRAME`, selon que le contenu est animé ou non.

Lorsque le contenu n'est pas animé et s'exécute pendant une longue période, les minuteurs sont préférables aux événements `Event.ENTER_FRAME`.

Dans ActionScript 3.0, il est possible d'appeler une fonction à intervalles réguliers de deux manières. La première repose sur l'utilisation de l'événement `Event.ENTER_FRAME` distribué par les objets d'affichage (DisplayObject). La seconde est fondée sur l'utilisation d'un minuteur. Les développeurs ActionScript privilégient fréquemment l'utilisation de l'événement `ENTER_FRAME`. L'événement `ENTER_FRAME` est distribué sur chaque image. La fréquence d'appel de la fonction est donc relative à la cadence en cours. La cadence est accessible par le biais de la propriété `Stage.frameRate`. Il est cependant préférable, dans certains cas, de faire appel à un minuteur plutôt qu'à l'événement `ENTER_FRAME`. C'est le cas, par exemple, si vous ne définissez pas d'animation mais souhaitez appeler le code à fréquence régulière.

Un minuteur peut se comporter de manière similaire à un événement `ENTER_FRAME`, mais il permet de distribuer un événement sans être tributaire de la cadence. Ce comportement peut permettre une optimisation sensible. Prenons l'exemple d'un lecteur vidéo. Dans ce cas, une cadence élevée n'est pas indispensable, car seuls les contrôles de l'application se déplacent.

***Remarque :** la cadence n'affecte pas la vidéo parce que celle-ci n'est pas intégrée au scénario. La vidéo est chargée dynamiquement par téléchargement progressif ou diffusion en continu.*

Dans cet exemple, la cadence est réglée sur une valeur faible, 10 ips. Le minuteur met à jour les contrôles, à raison de une mise à jour par seconde. Le taux de mise à jour plus élevé est possible grâce à la méthode `updateAfterEvent()`, qui est disponible sur l'objet TimerEvent. Cette méthode impose la mise à jour de l'écran chaque fois que le minuteur distribue un événement, si besoin est. Le code suivant illustre ce concept :

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

L'appel de la méthode `updateAfterEvent()` ne modifie pas la cadence. Cette méthode force simplement le moteur d'exécution à mettre à jour le contenu de l'écran qui a changé. Le scénario s'exécute toujours à 10 ips. Souvenez-vous cependant que les minuteurs et les événements `ENTER_FRAME` ne sont pas d'une précision à toute épreuve sur les périphériques dont les performances sont médiocres ou si les fonctions de gestionnaire d'événement contiennent du code exigeant un traitement intensif. A l'instar de la cadence des fichiers SWF, le taux de mise à jour du minuteur peut varier en fonction des circonstances.

Définissez un minimum d'objets Timer et de gestionnaires `enterFrame` enregistrés dans l'application.

Sur chaque image, le moteur d'exécution distribue un événement `enterFrame` à chaque objet figurant sur sa liste d'affichage. Vous pouvez enregistrer des écouteurs pour l'événement `enterFrame` sur plusieurs objets d'affichage mais cela signifie que le volume de code exécuté sur chaque image est plus élevé. Envisagez plutôt de définir un gestionnaire `enterFrame` centralisé unique qui exécute tout le code destiné à chaque image. En centralisant ce code, il est plus facile de gérer tout le code qui s'exécute fréquemment.

De même, si vous utilisez plusieurs objets Timer, la création et la distribution des événements correspondants se traduisent par une augmentation du temps système nécessaire. Si vous devez déclencher différentes opérations à différents intervalles, nous vous suggérons de procéder comme suit, au choix :

- Utilisez un minimum d'objets Timer et regroupez les opérations en fonction de leur fréquence d'exécution..

  Définissez, par exemple, un objet Timer se déclenchant toutes les 100 millisecondes pour les opérations fréquentes et un autre, réglé sur 2000 millisecondes, pour les opérations moins fréquentes ou exécutées en arrière-plan.

- Utilisez un seul objet Timer et déclenchez les opérations à des fréquences consistant en des multiples de la propriété `delay` de l'objet.

  Supposons, par exemple, que vous souhaitiez exécuter certaines opérations toutes les 100 millisecondes et d'autres toutes les 200 millisecondes. A cet effet, créez un seul objet Timer et définissez sa propriété `delay` sur 100 millisecondes. Dans le gestionnaire d'événement `timer`, ajoutez une instruction conditionnelle qui exécute une fois sur deux seulement les opérations à répéter toutes les 200 millisecondes. L'exemple suivant illustre cette technique :

```
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;
}
```

Arrêtez les objets Timer lorsqu'ils ne sont pas utilisés.

Si le gestionnaire d'événement `timer` d'un objet Timer n'exécute des opérations que dans certaines conditions, appelez la méthode `stop()` de l'objet lorsqu'aucune des conditions n'est vérifiée.

Dans un événement `enterFrame` ou des gestionnaires Timer, évitez autant que possible d'apporter à l'apparence de l'objet d'affichage des modifications entraînant l'actualisation de l'écran.

Sur chaque image, la phase de rendu retrace la zone de la scène qui a changé pendant l'image. Si cette zone est de grande taille ou si elle est petite mais contient des objets d'affichage nombreux ou complexes, le temps nécessaire pour le rendu augmente. La fonction « Afficher les zones de retraçage » de la version de débogage de Flash Player ou AIR permet de déterminer la quantité de retraçage nécessaire.

Pour plus d'informations sur l'amélioration des performances des actions répétées, voir l'article suivant :

- Writing well-behaved, efficient, AIR applications (Article et exemple d'application d'Arno Gourdol ; disponibles en anglais uniquement.)

Voir aussi

« Isolation de comportements » à la page 66

Interpolations

Evitez d'utiliser des interpolations pour économiser la puissance de traitement de l'unité centrale, la mémoire et la vie de la batterie.

Les concepteurs et développeurs qui produisent du contenu pour Flash sur le bureau ont tendance à être grands consommateurs d'interpolations de mouvement dans leurs applications. Lorsque vous créez du contenu destiné à des périphériques mobiles, limitez autant que possible l'utilisation des interpolations de mouvement afin que le contenu s'exécute plus rapidement sur les périphériques bas de gamme.

Chapitre 4 : Performances d'ActionScript 3.0

Comparaison des classes Vector et Array

Dans la mesure du possible, faites appel à la classe Vector plutôt qu'à la classe Array.

La classe Vector autorise des accès en lecture et écriture plus rapides que la classe Array.

Un simple test de performances démontre les avantages que présente la classe Vector par rapport à la classe Array. Le code suivant illustre un test de performances concernant 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
```

Le code suivant illustre un test de performances concernant 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
```

Il est possible d'optimiser encore plus l'exemple en attribuant une longueur précise fixe au vecteur :

```
// 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
```

Si la taille du vecteur n'est pas définie à l'avance, elle augmente lorsque le vecteur n'a plus de place. A chaque augmentation de la taille du vecteur, un nouveau bloc de mémoire est affecté. Le contenu actif du vecteur est copié dans le nouveau bloc de mémoire. Cette affectation de mémoire supplémentaire et la copie des données ont une incidence sur les performances. Dans le code ci-dessus, la taille initiale du vecteur est définie en vue d'optimiser les performances. Ce code n'est cependant pas optimisé à des fins de maintenabilité. Pour améliorer sa maintenabilité, placez la valeur réutilisée dans une constante :

```
// 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
```

Dans la mesure du possible, utilisez les API de l'objet Vector, car leur exécution sera probablement plus rapide.

API de dessin

L'API de dessin accélère l'exécution du code.

Flash Player 10 et AIR 1.5 intègrent une nouvelle API de dessin visant à optimiser les performances d'exécution du code. Si elle n'assure pas de meilleures performances de rendu, cette API permet cependant de réduire le nombre de lignes de code nécessaires, et cette réduction se traduit par une optimisation des performances d'exécution d'ActionScript.

La nouvelle API de dessin comporte les méthodes suivantes :

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

Remarque : *nous ne nous attarderons pas ici sur la méthode `drawTriangles()`, qui concerne le 3D. Cette méthode permet toutefois d'optimiser les performances d'ActionScript, car elle gère le mappage des textures en natif.*

Le code ci-dessous appelle explicitement la méthode appropriée pour chaque ligne à tracer :

```
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 );
```

Le code suivant s'exécute plus rapidement que celui de l'exemple précédent, car il contient un moins grand nombre de lignes. Plus le tracé est complexe et meilleures sont les performances résultant de l'utilisation de la méthode `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 );
```

La méthode `drawGraphicsData()` offre des possibilités d'optimisation similaires.

Capture et propagation d'événement

La capture et la propagation d'événement permettent de réduire le nombre de gestionnaires d'événement.

Le modèle d'événement d'ActionScript 3.0 a introduit les concepts de capture et de propagation d'événement. En propageant un événement, vous pouvez optimiser le temps d'exécution du code ActionScript. Vous pouvez enregistrer un gestionnaire d'événement sur un objet, plutôt que sur plusieurs objets, pour améliorer les performances.

Imaginons, par exemple, un jeu dans lequel l'utilisateur doit cliquer sur des pommes aussi vite que possible pour les détruire. Chaque pomme sur laquelle clique l'utilisateur est effacée et des points sont ajoutés au score de celui-ci. Pour écouter l'événement `MouseEvent.CLICK` distribué par chaque pomme, vous pourriez être tenté de rédiger le code suivant :

```
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 );
}
```

Le code appelle la méthode `addEventListener()` sur chaque occurrence d'Apple. Il supprime également chaque écouteur lorsque l'utilisateur clique sur une pomme, à l'aide de la méthode `removeEventListener()`. Cependant, le modèle d'événement d'ActionScript 3.0 propose une phase de capture et de propagation de certains événements, vous permettant d'écouter ceux-ci à partir d'un objet interactif (InteractiveObject) parent. Il est donc possible d'optimiser le code ci-dessus et de réduire le nombre d'appels des méthodes `addEventListener()` et `removeEventListener()`. Le code suivant utilise la phase de capture pour écouter les événements en provenance de l'objet parent :

```
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 );
}
```

Le code est simplifié et grandement optimisé ; il appelle la méthode `addEventListener()` une seule fois sur le conteneur parent. Les écouteurs n'étant plus enregistrés sur les occurrences d'Apple, il est inutile de les supprimer lorsque l'utilisateur clique sur une pomme. Il est possible d'optimiser encore plus le gestionnaire `onAppleClick()`, en arrêtant la propagation de l'événement, ce qui l'empêche d'aller plus loin :

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

Vous pouvez aussi utiliser la phase de propagation pour capturer l'événement en transmettant la valeur `false` en tant que troisième paramètre à la méthode `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 );
```

La valeur par défaut du paramètre de la phase de capture est `false` ; vous pouvez donc l'omettre :

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

Utilisation des pixels

Pour peindre des pixels, utilisez la méthode `setVector()`.

Lors de la peinture de pixels, il est possible de réaliser des optimisations simples à l'aide des méthodes appropriées de la classe BitmapData. Pour peindre rapidement des pixels, vous pouvez utiliser la méthode `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 );
```

Lorsque vous utilisez des méthodes lentes, telles que `setPixel()` ou `setPixel32()`, faites appel aux méthodes `lock()` et `unlock()` pour accélérer le processus. Dans le code suivant, les méthodes `lock()` et `unlock()` ont pour objet d'accélérer les performances :

```
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
```

La méthode `lock()` de la classe BitmapData verrouille une image et empêche la mise à jour des objets qui y font référence lors de la modification de l'objet BitmapData. Si un objet Bitmap fait référence à un objet BitmapData, par exemple, vous pouvez verrouiller ce dernier, le modifier et le déverrouiller. L'objet Bitmap n'est pas modifié tant que l'objet BitmapData n'est pas déverrouillé. Pour améliorer les performances, utilisez cette méthode en conjonction avec la méthode `unlock()` avant et après des appels répétés de la méthode `setPixel()` ou `setPixel32()`. L'appel de `lock()` et `unlock()` empêche les mises à jour superflues de l'écran.

Remarque : *lors du traitement des pixels d'un bitmap ne figurant pas sur la liste d'affichage (double mise en mémoire tampon), cette technique n'améliore pas toujours les performances. Si un objet bitmap ne fait pas référence à la mémoire tampon de bitmap, l'utilisation de `lock()` et `unlock()` n'apporte aucune amélioration des performances. Flash Player détecte que la mémoire tampon n'est pas référencée et le bitmap n'est pas rendu à l'écran.*

Les méthodes faisant l'objet d'une itération sur les pixels, telles que `getPixel()`, `getPixel32()`, `setPixel()` et `setPixel32()`, sont souvent lentes, surtout sur les périphériques mobiles. Si possible, utilisez des méthodes qui extraient tous les pixels en une seule fois. Pour lire des pixels, utilisez la méthode `getVector()`, qui est plus rapide que la méthode `getPixels()`. Dans la mesure du possible, utilisez également des API qui s'appuient sur des objets Vector car, selon toute probabilité, leur exécution sera plus rapide.

Expressions régulières

Utilisez des méthodes de la classe String, telles que `indexOf()`, `substr()` ou `substring()`, plutôt que des expressions régulières pour définir des opérations de recherche et d'extraction de chaînes de base.

Il est possible d'exécuter certaines opérations à l'aide d'une expression régulière ou des méthodes de la classe String. Pour déterminer si une chaîne en contient une autre, par exemple, vous pouvez utiliser la méthode `String.indexOf()` ou une expression régulière. Cependant, lorsqu'une méthode de la classe String est disponible, elle s'exécute plus rapidement que l'expression régulière équivalente et n'exige pas la création d'un autre objet.

Utilisez un groupe autre que capture ("`(?:xxxx)`") plutôt qu'un groupe ("`(xxxx)`") dans une expression régulière pour regrouper les éléments sans isoler le contenu du groupe dans le résultat.

Dans les expressions régulières modérément complexes, vous regroupez souvent des parties de l'expression. Dans le modèle d'expression régulière suivant, par exemple, les parenthèses créent un groupe autour du texte « ab ». Le quantificateur « + » s'applique donc au groupe et non à un seul caractère :

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

Le contenu de chaque groupe est « capturé », par défaut. Les données résultant de l'exécution de l'expression régulière peuvent contenir le contenu de chaque groupe du modèle. La capture de ces résultats de groupe prend plus longtemps et exige plus de mémoire, car des objets destinés à les contenir sont créés. Une autre syntaxe consiste à insérer les signes point d'interrogation et deux-points après la parenthèse ouvrante pour utiliser des groupes autres que capture. Cette syntaxe spécifie que les caractères se comportent comme un groupe mais ne sont pas capturés dans le résultat :

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

La syntaxe de groupes autres que capture est plus rapide et nécessite moins de mémoire que la syntaxe de groupes standard.

Si une expression régulière ralentit l'exécution du code, envisagez de lui en substituer une autre.

Il est parfois possible d'utiliser plusieurs modèles d'expression régulière pour tester ou identifier un même modèle de texte. Pour diverses raisons, certains modèles s'exécutent plus rapidement que d'autres. Si vous constatez qu'une expression régulière ralentit l'exécution du code plus qu'il n'est nécessaire, définissez d'autres modèles d'expression régulière permettant d'arriver au même résultat. Testez-les pour identifier le plus rapide.

Optimisations diverses

Pour un objet TextField, utilisez la méthode `appendText()` plutôt que l'opérateur `+=`.

Lorsque vous utilisez la propriété `text` de la classe TextField, préférez la méthode `appendText()` à l'opérateur `+=`. Cette méthode garantit de meilleures performances.

Le code suivant, par exemple, utilise l'opérateur `+=` et il faut 1 120 ms pour exécuter la boucle :

```
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
```

Dans l'exemple ci-dessous, nous avons remplacé l'opérateur `+=` par la méthode `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
```

Le code s'exécute à présent en 847 ms.


Si possible, mettez à jour les champs de texte en dehors des boucles.

Il est possible d'optimiser encore plus ce code par le biais d'une technique simple. La mise à jour du champ de texte dans chaque boucle met trop à contribution les ressources de traitement internes. Il suffit de concaténer une chaîne et de l'attribuer à la valeur du champ de texte en dehors de la boucle pour réduire considérablement la durée d'exécution du code, qui correspond maintenant à 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
```

Lors de l'utilisation de texte HTML, la première technique est tellement lente qu'elle renvoie parfois une exception `Timeout` dans Flash Player, dans certaines situations. Ce peut être le cas, par exemple, lorsque le matériel sous-jacent est trop lent.

Remarque : *Adobe® AIR® ne renvoie pas cette exception.*

```
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 );
```

Attribuez une valeur à une chaîne en dehors de la boucle et le code s'exécute en 29 ms seulement :

```
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
```

Remarque : *dans Flash Player 10.1 et AIR 2.5, la classe String a été améliorée de façon à ce que les chaînes soient moins gourmandes en mémoire.*


Evitez si possible d'utiliser l'opérateur crochet.

L'utilisation de l'opérateur crochet peut ralentir les performances. Pour éviter de vous en servir, vous pouvez stocker votre référence dans une variable locale. Le code suivant illustre une utilisation peu performante de l'opérateur crochet :

```
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 version optimisée de cet exemple, ci-dessous, contient moins d'opérateurs crochet :

```
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
```

Placez le code en ligne, si possible, pour réduire le nombre d'appels de fonctions qu'il contient.

Les appels de fonctions sont gourmands en ressources. Réduisez autant que faire se peut le nombre de ces appels en plaçant le code en ligne. Ce faisant, vous optimiserez vraiment les performances. Tenez néanmoins compte du fait qu'il peut être plus difficile de réutiliser du code en ligne et que la taille du fichier SWF risque d'augmenter. Il est facile de déplacer en ligne des appels de fonctions tels que les méthodes de la classe Math. Le code suivant s'appuie sur la méthode `Math.abs()` pour calculer des valeurs absolues :

```
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
```

Vous pouvez réaliser manuellement le calcul effectué par `Math.abs()` et le placer en ligne :

```
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
```

En conséquence du déplacement en ligne de l'appel de fonction, le code est plus de quatre fois plus rapide. Cette technique est préconisée dans de nombreux cas, mais pensez aux effets possibles sur la réutilisation et la maintenabilité du code.

Remarque : *la taille du code a une grande incidence sur l'exécution globale du lecteur. Si l'application comprend beaucoup de code ActionScript, il faut longtemps à la machine virtuelle pour vérifier le code et effectuer une compilation juste à temps (JIT). Les recherches de propriétés peuvent être lentes, en raison de la complexité accrue des hiérarchies d'héritage et parce que les mémoires cache ont tendance à être plus sollicitées. Pour réduire la taille du code, évitez d'utiliser la structure Adobe® Flex®, la bibliothèque de structures TLF ou toute bibliothèque ActionScript tierce de grande taille.*


Evitez également les boucles d'évaluation des instructions.

Une autre mesure d'optimisation consiste à ne pas évaluer une instruction dans une boucle. Le code suivant effectue des itérations sur un tableau mais il n'est pas optimisé, car la longueur du tableau est évaluée à chaque fois :

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

Il est préférable de stocker la valeur et de la réutiliser :

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

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

Utilisez l'ordre inverse dans les boucles while.

Une boucle while en ordre inverse est beaucoup plus rapide :

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

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

Ces astuces proposent quelques techniques d'optimisation d'ActionScript et illustrent l'impact d'une simple ligne de code sur les performances et la mémoire. Il en existe toutefois beaucoup d'autres. Pour plus d'informations, voir le lien suivant : http://www.rozengain.com/blog/2007/05/01/some-actionscript-30-optimizations/ (disponible en anglais uniquement).

Chapitre 5 : Performances de rendu

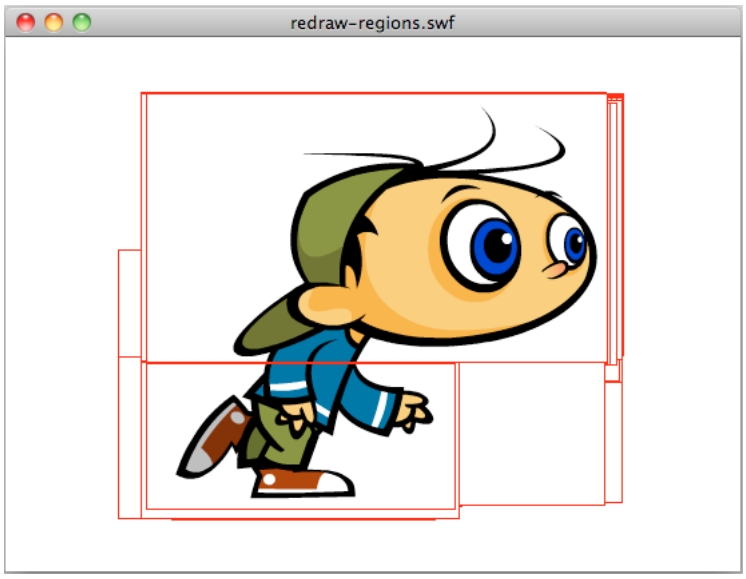
Options de retraçage

Utilisez toujours l'option Afficher les zones de retraçage lors de la création du projet.

Pour améliorer le rendu, il est important d'utiliser l'option Afficher les zones de retraçage lors de la création du projet. Cette option permet d'afficher les zones dont Flash Player effectue le rendu et qu'il traite. Pour activer cette option, sélectionnez Afficher les zones de retraçage dans le menu contextuel de la version de débogage de Flash Player.

Remarque : *l'option Afficher les zones de retraçage n'est pas disponible dans Adobe AIR ou dans la version commerciale de Flash Player. (Dans Adobe AIR, le menu contextuel est disponible uniquement dans les applications de bureau ; il ne comporte aucune option intégrée ou standard semblable à Afficher les zones de retraçage.)*

L'image suivante illustre un objet MovieClip animé simple sur le scénario, l'option Afficher les zones de retraçage étant activée :



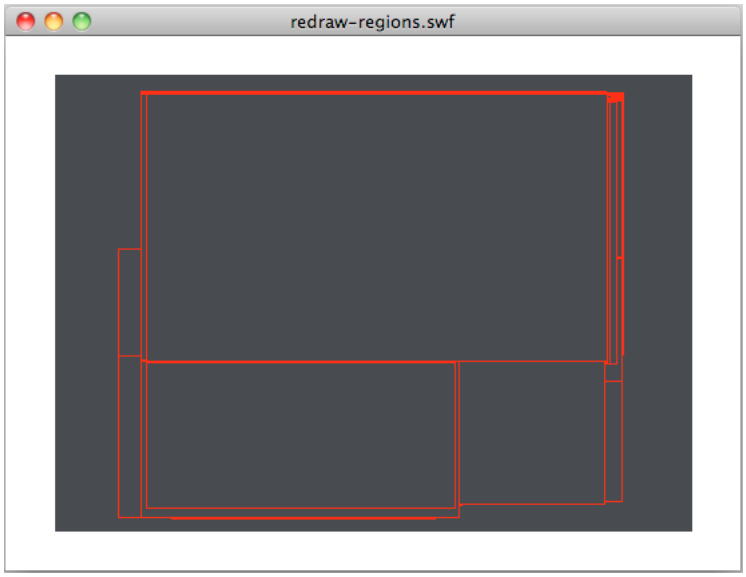
Option Afficher les zones de retraçage activée

Vous pouvez également activer cette option par programmation à l'aide de la méthode `flash.profiler.showRedrawRegions()` :

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

Dans les applications Adobe AIR, cette méthode est l'unique moyen d'activer l'option Afficher les zones de retraçage.

Utilisez l'option Afficher les zones de retraçage pour identifier les opportunités d'optimisation. N'oubliez pas que, même s'ils ne sont pas affichés, les objets d'affichage consomment néanmoins de nombreuses ressources de l'unité centrale, car leur rendu est toujours effectué. Ce concept est illustré ci-dessous. Une forme vectorielle noire couvre le personnage animé qui court. L'image montre que l'objet d'affichage n'a pas été supprimé de la liste d'affichage et est toujours rendu, d'où un gaspillage des ressources de l'unité centrale :



Régions retracées

Pour améliorer les performances, définissez la propriété `visible` du personnage animé masqué sur `false` ou supprimez-la de la liste d'affichage. Vous devez aussi arrêter son scénario. Ces mesures garantissent que l'objet d'affichage est figé et sollicite l'unité centrale au minimum.

Pensez à utiliser l'option Afficher les zones de retraçage pendant tout le cycle de développement. Grâce à elle, vous ne découvrirez pas en fin de projet des zones de retraçage superflues et des zones d'optimisation potentielles dont vous ne vous étiez pas rendu compte.

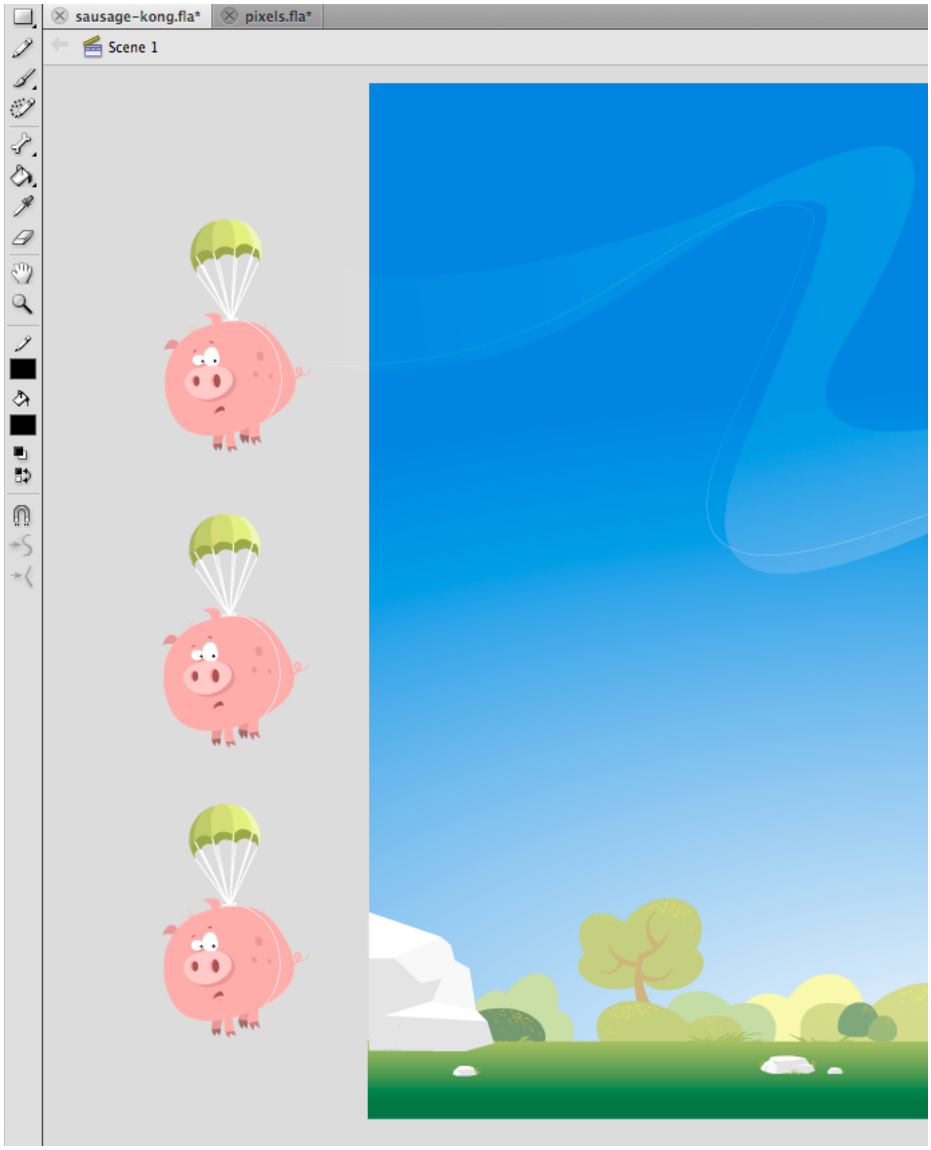
Voir aussi

« Figement et libération d'objets » à la page 28

Contenu hors-scène

Evitez de placer du contenu hors de la scène. Contentez-vous de placer les objets sur la liste d'affichage, lorsque cela est nécessaire.

Dans la mesure du possible, essayez de ne pas placer de contenu graphique hors de la scène. Les concepteurs et les développeurs ont pour habitude de placer les éléments hors de la scène pour réutiliser les actifs pendant la durée de vie de l'application. La figure suivante illustre cette technique courante :



Contenu hors-scène

Même si les éléments hors-scène ne sont pas visibles à l'écran et ne sont pas rendus, ils restent néanmoins présents dans la liste d'affichage. Le moteur d'exécution continue de soumettre ces éléments à des tests internes pour vérifier qu'ils sont toujours hors-scène et que l'utilisateur n'interagit pas avec eux. Vous devez donc éviter, dans la mesure du possible, de placer des objets hors de la scène en vous contentant de les supprimer de la liste d'affichage.

Qualité des clips

Utilisez le paramètre de qualité de scène approprié pour améliorer le rendu.

Lorsque vous développez du contenu pour des périphériques mobiles dont l'écran est de petite taille, tels que des téléphones, la qualité d'image est moins importante que pour les applications de bureau. Le choix du paramètre de qualité de scène adapté peut améliorer les performances de rendu.

Les paramètres disponibles sont les suivants :

- `StageQuality.LOW` : favorise la vitesse de lecture par rapport à l'aspect et n'utilise pas l'anticrènelage. Ce paramètre n'est pas pris en charge dans la version de bureau d'Adobe AIR ou dans Adobe AIR pour TV.
- `StageQuality.MEDIUM` : applique un certain degré d'anticrènelage, mais ne lisse pas les bitmaps mis à l'échelle. Cette valeur est la valeur par défaut d'AIR sur les périphériques mobiles ; elle n'est pas prise en charge dans les versions AIR de bureau ou dans AIR pour TV.
- `StageQuality.HIGH` (paramètre par défaut sur le bureau) : favorise l'aspect par rapport à la vitesse de lecture et applique systématiquement l'anticrènelage. Si le fichier SWF ne contient aucune animation, les bitmaps mis à l'échelle sont lissés ; dans le cas contraire, ils ne le sont pas.
- `StageQuality.BEST` : assure la meilleure qualité d'affichage et ne tient pas compte de la vitesse de lecture. Toute la sortie est anticrènelée et les bitmaps mis à l'échelle sont toujours lissés.

`StageQuality.MEDIUM` permet généralement d'obtenir une qualité adéquate pour les applications sur les périphériques mobiles. Dans certains cas, il suffit même d'utiliser `StageQuality.LOW`. Depuis Flash Player 8, il est possible d'effectuer un rendu exact du texte anticrènelé, même si vous utilisez le paramètre de qualité de scène `LOW`.

Remarque : *sur certains périphériques mobiles, même si la qualité est définie sur `HIGH`, `MEDIUM` est utilisée pour obtenir de meilleures performances dans les applications Flash Player. Le paramètre de qualité HIGH fait peu de différence, car les écrans des périphériques mobiles utilisent généralement une résolution plus élevée. (La résolution peut varier en fonction du périphérique.)*

Dans l'illustration ci-après, le paramètre de qualité de scène est défini sur `MEDIUM` et le rendu du texte sur Anticrènelage pour l'animation :

Qualité de scène définie sur MEDIUM et rendu du texte sur Anticrènelage pour l'animation

Le paramètre de qualité de scène affecte la qualité du texte, car le paramètre de rendu de texte approprié n'est pas utilisé.

Le moteur d'exécution permet de définir le rendu du texte sur Anticrènelage pour la lisibilité. Grâce à elle, la qualité du texte (anticrènelé) reste parfaite quelle que soit la valeur du paramètre de qualité de scène :

Qualité de scène définie sur LOW et rendu du texte sur Anticrènelage pour la lisibilité

Il est possible d'obtenir la même qualité de rendu en définissant le rendu du texte sur Texte bitmap (sans anticrènelage) :

Qualité de scène définie sur LOW et rendu du texte sur Texte bitmap (sans anticrènelage)

Les deux derniers exemples démontrent que vous pouvez obtenir du texte de qualité supérieure, quelle que soit la valeur du paramètre de qualité de scène. Cette fonction est disponible depuis Flash Player 8 et peut être utilisée sur les périphériques mobiles. Gardez à l'esprit que Flash Player 10.1 active automatiquement `StageQuality.MEDIUM` sur certains périphériques pour améliorer les performances.

Fusion alpha

Evitez si possible d'utiliser la propriété alpha.

Lorsque vous utilisez la propriété `alpha`, évitez de créer des effets nécessitant une fusion alpha, tels que les effets de fondu. Lorsqu'un objet d'affichage utilise la fusion alpha, le moteur d'exécution doit combiner les valeurs de couleur de chaque objet d'affichage empilé et la couleur d'arrière-plan pour déterminer la couleur finale. La fusion alpha peut exiger davantage de ressources processeur que la création d'une couleur opaque, ce qui risque de diminuer les performances sur les périphériques lents. Evitez d'utiliser la propriété `alpha`, si possible.

Voir aussi

« Mise en cache sous forme de bitmap » à la page 55

« Rendu des objets de texte » à la page 69

Cadence de l'application

En règle générale, pour améliorer les performances, optez pour la plus faible cadence possible.

La cadence de l'application détermine le temps disponible pour chaque cycle « code d'application et rendu » (voir « Principes fondamentaux relatifs à l'exécution du code par le moteur d'exécution » à la page 1). Plus la cadence est élevée et plus l'animation est fluide. Toutefois, en l'absence d'animation ou d'autres modifications visuelles, une cadence élevée n'a aucune raison d'être. Une cadence élevée sollicite davantage l'unité centrale et la batterie qu'une cadence inférieure.

Suivez les conseils généraux suivants pour déterminer la cadence par défaut appropriée pour votre application :

- Si vous utilisez la structure Flex, réglez la cadence initiale sur la valeur par défaut..
- Si l'application comprend de l'animation, une cadence de 20 images par seconde au minimum est adéquate. Il est souvent inutile de dépasser 30 images par seconde.
- Si l'application ne comporte pas d'animation, une cadence de 12 images par seconde est probablement suffisante.

Notez que la « plus faible cadence possible » peut varier en fonction de l'activité en cours de l'application. Pour plus d'informations, voir « Modifiez dynamiquement de la cadence de l'application ».

Adoptez une cadence faible lorsque la vidéo constitue le seul contenu dynamique de l'application.

Le moteur d'exécution lit le contenu vidéo chargé à sa cadence native, quelle que soit la cadence de l'application. Si l'application ne comporte pas d'animation ou d'autre contenu visuel changeant rapidement, l'adoption d'une cadence faible ne ralentit pas les performances de l'interface utilisateur.

Modifiez dynamiquement la cadence de l'application.

Vous définissez la cadence initiale de l'application dans les paramètres du projet ou du compilateur, mais elle n'est pas fixe. Vous pouvez la modifier à l'aide de la propriété `Stage.frameRate` (ou de la propriété `WindowedApplication.frameRate` dans Flex).

Modifiez la cadence en fonction des exigences en cours de l'application. Réduisez-la lorsque l'application n'exécute pas d'animation, par exemple. Augmentez-la juste avant une transition animée. De même, si l'application s'exécute en arrière-plan (après avoir perdu le focus), vous pouvez généralement réduire encore plus la cadence. Selon toute probabilité, l'utilisateur se concentre sur une autre application ou tâche.

Les conseils généraux suivants ont pour objet de vous permettre de déterminer la cadence appropriée en fonction de différentes activités :

- Si vous utilisez la structure Flex, réglez la cadence initiale sur la valeur par défaut..
- Lorsqu'une animation s'exécute, réglez la cadence sur 20 images par seconde au minimum. Il est souvent inutile de dépasser 30 images par seconde.
- En l'absence d'animation, une cadence de 12 images par seconde est probablement suffisante.

- Une vidéo chargée s'exécute à sa cadence native, indépendamment de la cadence de l'application. Si la vidéo est le seul contenu en mouvement de l'application, une cadence de 12 images par seconde est probablement suffisante.
- Lorsque l'application ne possède pas le focus d'entrée, une cadence de 5 images par seconde est probablement suffisante.
- Lorsqu'une application AIR n'est pas visible, une cadence de 2 images par seconde au maximum est probablement suffisante. Par exemple, cette indication s'applique lors de la réduction d'une application. Elle s'applique également sur les périphériques de bureau si la propriété `visible` de la fenêtre native est définie sur `false`.

Dans le cas des applications créées dans Flex, la classe spark.components. WindowedApplication prend en charge la modification dynamique de la cadence de l'application. La propriété `backgroundFrameRate` définit la cadence de l'application lorsque celle-ci est inactive. La valeur par défaut, 1, fait passer à 1 image par seconde la cadence d'une application créée à l'aide de la structure Spark. Vous pouvez modifier la cadence d'arrière-plan à l'aide de la propriété `backgroundFrameRate`. Vous pouvez définir la propriété sur une autre valeur ou la fixer à -1 pour désactiver la régulation automatique de la cadence.

Pour plus d'informations sur la modification dynamique de la cadence d'une application, voir les articles suivants :

- Reducing CPU usage in Adobe AIR (Article et exemple de code de Jonnie Hallman sur le pôle de développement Adobe. Disponibles en anglais uniquement)
- Writing well-behaved, efficient, AIR applications (article et exemple d'application d'Arno Gourdol ; disponibles en anglais uniquement.)

Grant Skinner a créé une classe de régulation de la cadence. Vous pouvez vous en servir dans vos applications pour réduire automatiquement la cadence lorsque l'application est en arrière-plan. Pour plus d'informations et pour télécharger le code source de la classe FramerateThrottler, voir l'article de Grant intitulé Idle CPU Usage in Adobe AIR and Flash Player (disponible en anglais uniquement) à l'adresse http://gskinner.com/blog/archives/2009/05/idle_cpu_usage.html.

Cadence adaptative

Lorsque vous compilez un fichier SWF, vous définissez la cadence (images par seconde) du clip. Dans un environnement restreint où la vitesse de l'unité centrale est faible, il arrive que la cadence baisse en cours de lecture. Pour préserver une cadence convenable côté utilisateur, le moteur d'exécution omet le rendu de certaines images. Ce processus empêche la cadence de baisser en deçà d'une valeur acceptable.

Remarque : *dans ce cas, le moteur d'exécution ne saute pas d'images, il n'effectue simplement pas le rendu du contenu de certaines d'entre elles. Le code est toujours exécuté et la liste d'affichage mise à jour, mais les mises à jour ne sont pas affichées. Il est impossible de spécifier un seuil d'images par seconde pour indiquer le nombre d'images dont le moteur d'exécution ne doit pas effectuer le rendu en vue d'assurer une cadence stable.*

Mise en cache sous forme de bitmap

Appliquez la fonction de mise en cache sous forme de bitmap au contenu vectoriel complexe, lorsque cela est possible.

La fonction de mise en cache sous forme de bitmap permet d'effectuer une bonne optimisation. Cette fonction met en cache un objet vectoriel, puis effectue en interne son rendu sous forme de bitmap. Elle utilise ensuite ce bitmap pour le rendu final. Si ce processus se traduit éventuellement par une nette amélioration des performances de rendu, il peut cependant être gourmand en mémoire. Utilisez la fonction de mise en cache sous forme de bitmap pour le contenu vectoriel complexe, tel que les dégradés ou le texte complexes.

L'activation de la fonction de mise en cache sous forme de bitmap pour un objet animé contenant des graphiques vectoriels complexes (tels que du texte et des dégradés) améliore les performances. Cependant, si la fonction de mise en cache des bitmaps est activée sur un objet d'affichage tel qu'une animation dont le scénario s'exécute, vous obtenez le résultat inverse. Sur chaque image, le moteur d'exécution doit mettre à jour le bitmap mis en cache et le retracer à l'écran, ce qui sollicite fortement l'unité centrale. La fonction de mise en cache sous forme de bitmap ne présente des avantages que lorsqu'il est possible de générer une seule fois le bitmap mis en cache, puis de l'utiliser sans le mettre à jour.

Si vous activez cette fonction pour un objet Sprite, il est possible de déplacer celui-ci sans que le moteur d'exécution soit obligé de régénérer le bitmap mis en cache. La modification des propriétés `x` et `y` de l'objet n'entraîne pas sa régénération. En revanche, toute tentative de rotation, mise à l'échelle ou modification de sa valeur alpha force le moteur d'exécution à régénérer le bitmap mis en cache, ce qui affecte les performances.

Remarque : *cette limitation ne concerne pas la propriété* `DisplayObject.cacheAsBitmapMatrix`*, que proposent AIR et l'outil Packager for iPhone. La propriété* `cacheAsBitmapMatrix` *permet de faire pivoter, mettre à l'échelle, incliner et modifier la valeur alpha d'un objet d'affichage sans déclencher une régénération de l'image bitmap.*

Un bitmap mis en cache peut utiliser plus de mémoire qu'une occurrence de clip courante. Un clip de 250 x 250 pixels sur la scène, par exemple, utilise environ 250 Ko une fois mis en cache, mais seulement 1 Ko lorsqu'il n'est pas mis en cache.

L'exemple ci-dessous est fondé sur un objet Sprite contenant une image de pomme. La classe suivante est associée au symbole de pomme :

```
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
```

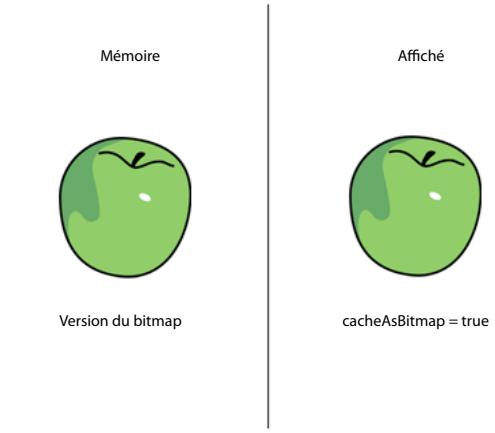
```
        {
            removeEventListener(Event.ENTER_FRAME,handleMovement);
        }

        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();
        }
    }
}
```

Le code utilise la classe Sprite plutôt que la classe MovieClip, car les pommes ne nécessitent pas un scénario chacune. Pour optimiser les performances, utilisez l'objet le plus léger possible. La classe est ensuite instanciée avec le code suivant :

```
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);
        }
    }
}
```

Lorsque l'utilisateur effectue un clic de souris, les pommes sont créées sans être mises en cache. Lorsqu'il appuie sur la touche C (code de touche 67), les vecteurs pomme sont mis en cache sous forme de bitmaps et affichés. Cette technique améliore considérablement les performances de rendu sur le bureau et les périphériques mobiles, lorsque l'unité centrale est lente.

En revanche, elle peut très rapidement se révéler très gourmande en mémoire. Dès qu'un objet est mis en cache, sa surface est capturée en tant que bitmap transparent et stockée en mémoire, comme illustré ci-dessous :

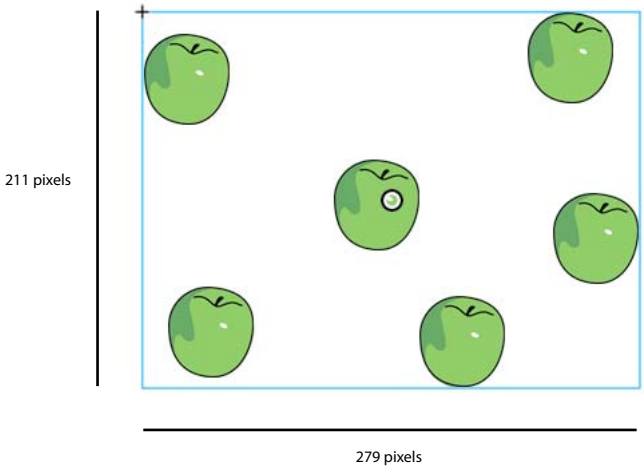


Objet et son bitmap de surface stockés en mémoire

Flash Player 10.1 et AIR 2.5 optimisent l'utilisation de la mémoire en appliquant la technique décrite à la section « Filtres et déchargement dynamique de bitmaps » à la page 20. Si un objet d'affichage mis en cache est masqué ou hors écran, le bitmap correspondant est libéré de la mémoire après une certaine période d'inutilisation.

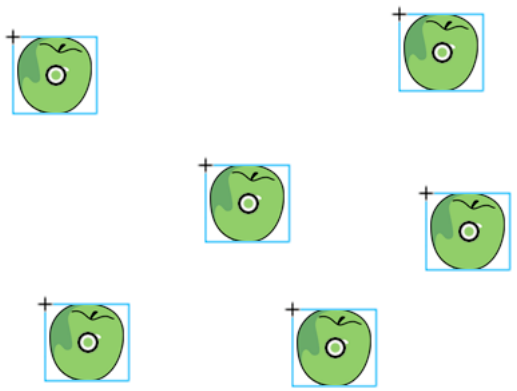
Remarque : *si la propriété* `opaqueBackground` *de l'objet d'affichage est définie sur une couleur spécifique, le moteur d'exécution considère que l'objet est opaque. Utilisé conjointement avec la propriété* `cacheAsBitmap`*, le moteur d'exécution crée un bitmap 32 bits non transparent en mémoire. Le canal alpha est défini sur 0xFF, ce qui se traduit par une amélioration des performances, car il est inutile d'appliquer une transparence pour tracer le bitmap à l'écran. Le rendu est encore plus rapide si la fusion alpha n'est pas nécessaire. Si la profondeur d'écran actuelle est limitée à 16 bits, le bitmap en mémoire est stocké sous la forme d'une image 16 bits. L'utilisation de la propriété* `opaqueBackground` *n'active pas implicitement la mise en cache sous forme de bitmap.*

Pour économiser de la mémoire, utilisez la propriété `cacheAsBitmap` et activez-la sur chaque objet d'affichage mais pas sur le conteneur. L'activation de la mise en cache sous forme de bitmap sur le conteneur génère un bitmap final beaucoup plus gros en mémoire, soit un bitmap transparent de 211 x 279 pixels utilisant environ 229 Ko de mémoire :



Activation de la mise en cache sous forme de bitmap sur le conteneur

En outre, si vous mettez en cache le conteneur, il se peut que la totalité du bitmap soit mise à jour en mémoire, si une pomme se met à se déplacer sur une image. En revanche, l'activation de la mise en cache sous forme de bitmap sur des occurrences individuelles consiste à mettre en cache six surfaces de 7 Ko, qui consomment uniquement 42 Ko de mémoire :



Activation de la mise en cache sous forme de bitmap sur les occurrences

Si vous accédez à chaque occurrence de la pomme par le biais de la liste d'affichage et appelez la méthode `getChildAt()`, les références sont stockées dans un objet Vector pour un accès plus rapide :

```
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);
        }
    }
}
```

Souvenez-vous que la mise en cache sous forme de bitmap améliore le rendu, à condition de ne pas faire pivoter, mettre à l'échelle ou modifier le contenu mis en cache sur chaque image. Cette amélioration ne se produit pas pour toute transformation autre qu'une translation sur les axes x et y. Flash Player met en effet à jour la copie du bitmap mis en cache pour chaque transformation de l'objet d'affichage. Cela peut se traduire par une forte sollicitation de l'unité centrale et de la batterie et une baisse des performances. Pour rappel, cette limitation ne concerne pas la propriété `cacheAsBitmapMatrix`, que proposent AIR ou l'outil Packager for iPhone

Le code suivant agit sur la valeur alpha dans la méthode de mouvement, ce qui a pour effet de modifier l'opacité de la pomme à chaque image :

```
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'utilisation de la fonction de mise en cache sous forme de bitmap entraîne un ralentissement des performances. A chaque modification de la valeur alpha, le moteur d'exécution doit mettre à jour le bitmap mis en cache en mémoire.

Les filtres s'attendent à ce que les bitmaps soient mis à jour à chaque déplacement de la tête de lecture d'un clip mis en cache. Par conséquent, l'application d'un filtre entraîne automatiquement la définition de la propriété `cacheAsBitmap` sur `true`. La figure suivante illustre une animation :

Clip animé

Il est préférable de ne pas appliquer de filtres à du contenu animé, car ils risquent d'entraîner des problèmes de performances. Dans l'exemple suivant, le concepteur ajoute un filtre d'ombre portée :

Clip animé doté du filtre Ombre portée

Si le scénario que contient le clip s'exécute, le bitmap doit être régénéré. Cette opération est également nécessaire si le contenu subit toute modification autre qu'une simple transformation x ou y. Le moteur d'exécution doit retracer le bitmap sur chaque image. Cette opération sollicite fortement les ressources de l'unité centrale, entraîne des performances médiocres et consomme plus de batterie.

Paul Trani fournit des exemples d'utilisation de Flash Professional et d'ActionScript pour optimiser les graphiques à l'aide de bitmaps dans les vidéos de formation suivantes :

- Optimizing Graphics (disponible en anglais uniquement)
- Optimizing Graphics with ActionScript (disponible en anglais uniquement)

Matrices de transformation des bitmaps mis en cache dans AIR

Définissez la propriété `cacheAsBitmapMatrix` lorsque vous utilisez des bitmaps mis en cache dans les applications AIR mobiles.

Dans le profil AIR mobile, vous pouvez affecter un objet Matrix à la propriété `cacheAsBitmapMatrix` d'un objet d'affichage. Lorsque vous définissez cette propriété, vous pouvez appliquer une transformation bidimensionnelle à l'objet sans régénérer le bitmap mis en cache. Vous pouvez en outre modifier la propriété alpha sans régénérer le bitmap mis en cache. Il est également nécessaire de définir la propriété `cacheAsBitmap` sur `true` et de vérifier qu'aucune propriété 3D de l'objet n'est définie.

La définition de la propriété `cacheAsBitmapMatrix` génère le bitmap mis en cache même si l'objet d'affichage est en dehors de l'écran ou masqué, ou si sa propriété `visible` est définie sur `false`. La réinitialisation de la propriété `cacheAsBitmapMatrix` à l'aide d'un objet Matrix contenant une transformation différente régénère également le bitmap mis en cache.

La transformation de matrice que vous appliquez à la propriété `cacheAsBitmapMatrix` est également appliquée à l'objet d'affichage lors du rendu de ce dernier dans le cache de bitmaps. Ainsi, si la transformation contient une échelle de 2x, la taille du rendu bitmap est deux fois supérieure à celle du rendu vectoriel. Le moteur de rendu applique la transformation inverse au bitmap mis en cache afin que l'affichage final ait un aspect identique. Vous pouvez réduire la taille du bitmap mis en cache afin de limiter la consommation de mémoire, au détriment de la fidélité du rendu. Inversement, vous pouvez agrandir le bitmap pour augmenter la qualité du rendu, bien que cela risque d'accroître la consommation de mémoire. Quoi qu'il en soit, vous devez en général utiliser une matrice d'identité, c'est-à-dire une matrice qui n'applique aucune transformation, afin d'éviter toute modification de l'apparence, comme indiqué dans l'exemple suivant :

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

Une fois la propriété `cacheAsBitmapMatrix` définie, vous pouvez mettre à l'échelle, incliner, faire pivoter et translater l'objet sans déclencher la régénération du bitmap.

Vous pouvez également modifier la valeur alpha dans une plage de valeurs comprises entre 0 et 1. Si vous modifiez la valeur alpha via la propriété `transform.colorTransform` avec une transformation de couleur, la valeur alpha utilisée dans l'objet de transformation doit être comprise entre 0 et 255. Modifier autrement la transformation de couleur entraîne la régénération du bitmap mis en cache.

Veillez à toujours définir la propriété `cacheAsBitmapMatrix` chaque fois que vous définissez `cacheAsBitmap` sur `true` dans le contenu créé pour les périphériques mobiles. Vous devez toutefois prendre en compte l'inconvénient suivant. Après la rotation, la mise à l'échelle ou l'inclinaison d'un objet, le rendu final peut présenter des artefacts de crènelage ou de mise à l'échelle par rapport à un rendu vectoriel normal.

Mise en cache manuelle sous forme de bitmap

Utilisez la classe BitmapData pour créer un comportement personnalisé de mise en cache des bitmaps.

L'exemple ci-après réutilise une version bitmap pixellisée unique d'un objet d'affichage et fait référence à un même objet BitmapData. Lors de la mise à l'échelle de chaque objet d'affichage, l'objet BitmapData original en mémoire ni n'est ni mis à jour ni redessiné. Cette technique permet d'économiser les ressources de l'unité centrale et accélère l'exécution des applications. Lors de la mise à l'échelle de l'objet d'affichage, le bitmap contenu est étiré.

La classe `BitmapApple` mise à jour se présente comme ceci :

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

La valeur alpha est toujours modifiée sur chaque image. Le code suivant transmet la mémoire tampon source d'origine à chaque occurrence de 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);
}
```

Cette technique consomme peu de mémoire, car un seul bitmap mis en cache est utilisé en mémoire et partagé par toutes les occurrences de `BitmapApple`. En outre, malgré les modifications apportées aux occurrences de `BitmapApple` (modification de la propriété alpha, rotation ou mise à l'échelle, par exemple), le bitmap source original n'est jamais mis à jour. Cette technique empêche un ralentissement des performances.

Pour obtenir au final un bitmap lisse, définissez la propriété `smoothing` sur `true` :

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

    smoothing = true;

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

Le réglage du paramètre de qualité de scène peut également améliorer les performances. Définissez-le sur `HIGH` avant la pixellisation, puis sur `LOW` après :

```
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 modification de la qualité de scène avant et après le dessin du vecteur dans un bitmap constitue une puissante technique de création de contenu anticrènelé à l'écran. Cette technique peut être performante quelle que soit la qualité de scène finale. Vous pouvez, par exemple, obtenir un bitmap anticrènelé à partir de texte anticrènelé, même si la qualité de scène est définie sur `LOW`. Cette technique n'est pas compatible avec la propriété `cacheAsBitmap`. Dans ce cas, la définition de la qualité de scène sur `LOW` entraîne la mise à jour de la qualité des vecteurs, ce qui met à jour les surfaces bitmap en mémoire et la qualité finale.

Isolation de comportements

Dans la mesure du possible, isolez les événements, tels que `Event.ENTER_FRAME`, dans un même gestionnaire.

Vous pouvez optimiser encore plus le code en isolant l'événement `Event.ENTER_FRAME` de la classe `Apple` dans un gestionnaire unique. Cette technique permet d'économiser les ressources de l'unité centrale. L'exemple suivant illustre cette technique différente, selon laquelle la classe BitmapApple ne gère plus le comportement de mouvement :

```
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;
        }
}
```

Le code suivant instancie les pommes et gère leur mouvement dans un même gestionnaire :

```
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;
            }
        }
}
```

Un seul événement `Event.ENTER_FRAME` gère donc le mouvement, à la place de 200 gestionnaires déplaçant chaque pomme. Il est facile de mettre toute l'animation en pause, ce qui peut être pratique dans un jeu.

Un jeu simple, par exemple, peut utiliser le gestionnaire suivant :

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

L'étape suivante consiste à faire interagir les pommes avec la souris ou le clavier, ce qui nécessite de modifier la 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 en résulte des occurrences de `BitmapApple` interactives, à l'instar des objets Sprite classiques. Les occurrences sont toutefois liées à un bitmap unique, qui n'est pas rééchantillonné lors de la transformation des objets d'affichage.

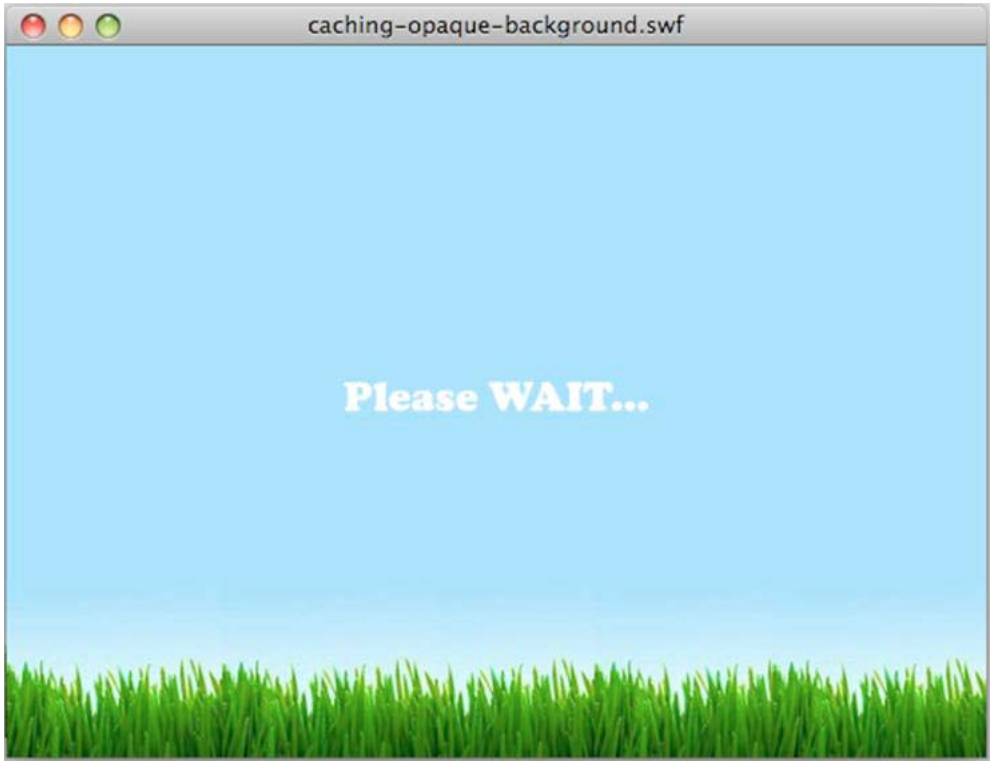
Rendu des objets de texte

Conjuguez la fonction de mise en cache sous forme de bitmap et la propriété `opaqueBackground` pour améliorer les performances de rendu du texte.

Flash Text Engine propose d'excellentes optimisations. Cependant, de nombreuses classes sont nécessaires pour afficher une seule ligne de texte. Pour créer un champ de texte modifiable à l'aide de la classe TextLine exige donc beaucoup de mémoire et de nombreuses lignes de code ActionScript. Cette classe est mieux adaptée au texte statique non modifiable. Elle assure dans ce cas un rendu plus rapide et consomme moins de mémoire.

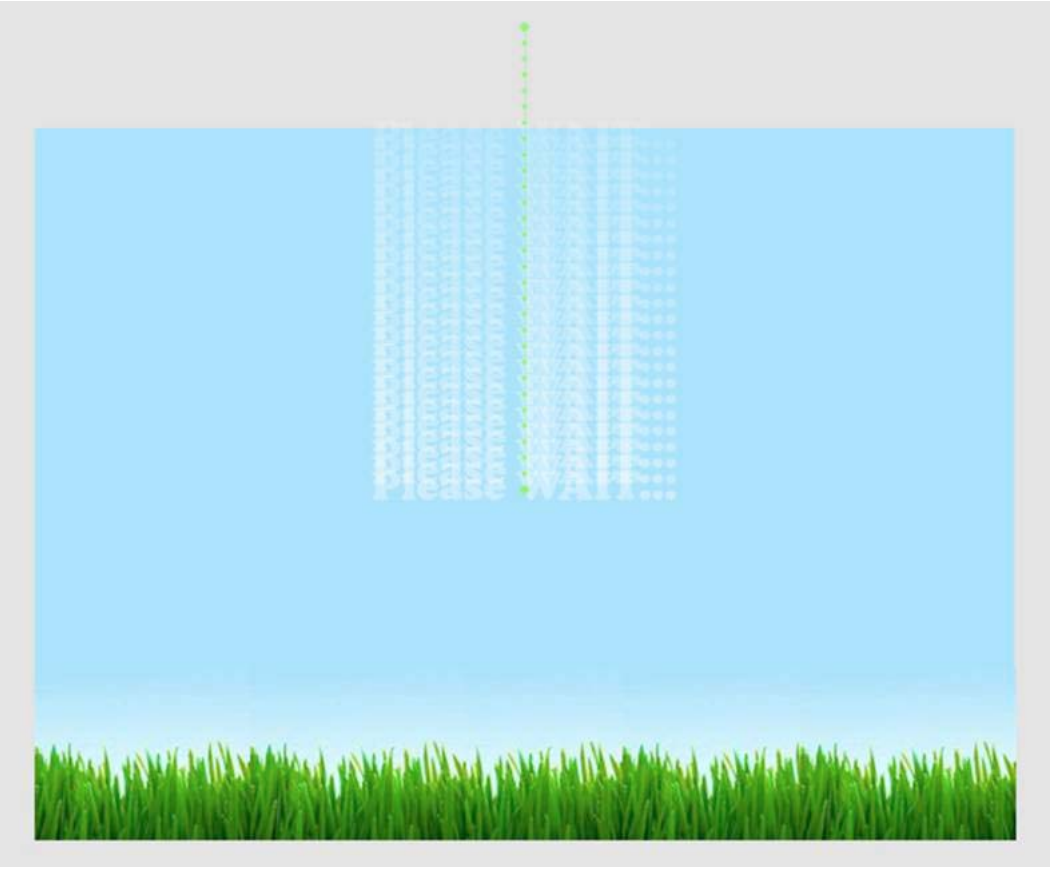
La fonction de mise en cache sous forme de bitmap permet de mettre en cache du contenu vectoriel sous forme de bitmaps pour améliorer les performances de rendu. Elle convient au contenu vectoriel complexe et au texte dont le rendu nécessite un traitement préliminaire.

L'exemple suivant montre comment conjuguer la fonction de mise en cache sous forme de bitmap et la propriété `opaqueBackground` pour améliorer les performances de rendu. La figure ci-dessous illustre un écran de bienvenue standard présenté à l'utilisateur lors du chargement de ressources :



Ecran de bienvenue

La figure suivante illustre l'accélération appliquée à l'objet TextField par programmation. Le texte est accéléré lentement du haut de la scène vers le centre de celle-ci :



Accélération de texte

Le code suivant crée l'accélération. La variable `preloader` stocke l'objet cible actif pour réduire les recherches de propriétés, qui peuvent avoir une incidence sur les performances :

```
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 );
}
```

Dans cet exemple, vous pourriez placer la fonction `Math.abs()` en ligne pour réduire le nombre des appels de fonction et améliorer encore plus les performances. Il est recommandé de définir les propriétés `destX` et `destY` sur le type int pour obtenir des valeurs à virgule fixe. L'utilisation du type int assure un accrochage aux pixels parfait sans avoir à arrondir manuellement les valeurs par le biais de méthodes lentes telles que`Math.ceil()` ou `Math.round()`. Ce code n'arrondit pas les coordonnées à des valeurs entières car, en conséquence d'un arrondissement constant, le déplacement de l'objet n'est pas fluide. Ses mouvements peuvent être saccadés, car les coordonnées sont accrochées aux entiers arrondis les plus proches sur chaque image. Cette technique peut cependant être utile pour définir la position finale d'un objet d'affichage. N'utilisez pas le code suivant :

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

Le code suivant est beaucoup plus rapide :

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

Il est possible d'optimiser encore plus le code précédent en utilisant des opérateurs de décalage au niveau du bit pour diviser les valeurs :

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

Grâce à la fonction de mise en cache sous forme de bitmap, le moteur d'exécution peut utiliser des bitmaps dynamiques, ce qui facilite le rendu des objets. Dans le présent exemple, le clip contenant l'objet TextField est en cache :

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

Pour améliorer les performances, vous pouvez aussi supprimer la transparence alpha. Celle-ci surcharge le moteur d'exécution lors de la création d'images bitmap transparentes, comme dans le code précédent. Pour contourner cette difficulté, vous pouvez utiliser la propriété `opaqueBackground` et spécifier une couleur comme arrière-plan.

Lorsque vous utilisez la propriété `opaqueBackground`, la surface bitmap créée en mémoire occupe encore 32 bits. Le décalage alpha est toutefois défini sur 255 et aucune transparence n'est appliquée. La propriété `opaqueBackground` ne permet donc pas de réduire la consommation de mémoire mais, lorsqu'elle est alliée à la fonction de mise en cache sous forme de bitmap, elle améliore les performances de rendu. Le code suivant conjugue toutes les possibilités d'optimisation :

```
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'animation est maintenant optimisée. La mise en cache sous forme de bitmap l'est également en conséquence de la suppression de la transparence. Sur les périphériques mobiles, vous pouvez éventuellement faire passer la qualité de scène de `LOW` à `HIGH` (et inversement) au cours des différents états de l'animation tout en utilisant la fonction de mise en cache sous forme de 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 );
    }
}
```

Dans ce cas, le moteur d'exécution doit cependant régénérer la surface bitmap de l'objet TextField afin qu'il corresponde au paramètre de qualité de scène actif. Il est donc préférable de ne pas modifier ce paramètre lorsque vous utilisez la fonction de mise en cache sous forme de bitmap.

Cet exemple se prête également à une technique de mise en cache sous forme de bitmap manuelle. Pour simuler la propriété `opaqueBackground`, il est possible de dessiner le clip sur un objet BitmapData non transparent ; le moteur d'exécution n'est alors pas obligé de régénérer la surface bitmap.

Cette technique est adaptée au contenu qui ne change pas à terme. En revanche, s'il est possible de modifier le contenu du champ de texte, faites si possible appel à une autre stratégie. Imaginons qu'un champ de texte soit constamment mis à jour pour indiquer le pourcentage de l'application qui est chargé. Si le champ de texte, ou l'objet d'affichage le contenant, a été mis en cache sous forme de bitmap, sa surface doit être générée chaque fois que le contenu change. Il est impossible d'utiliser la mise en cache sous forme de bitmap manuelle dans ce cas, car le contenu de l'objet d'affichage change sans cesse. Vous seriez alors forcé d'appeler manuellement la méthode `BitmapData.draw()` pour mettre à jour le bitmap mis en cache.

Pour rappel, depuis Flash Player 8 (et AIR 1.0), quelle que soit la valeur de la qualité de scène, un champ de texte dont le rendu est défini sur Anticrènelage pour la lisibilité reste parfaitement anticrènelé. Cette technique est moins gourmande en mémoire mais elle sollicite plus l'unité centrale et le rendu est un peu moins rapide qu'avec la fonction de mise en cache sous forme de bitmap.

Le code suivant repose sur cette technique :

```
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 );
    }
}
```

Il est déconseillé d'utiliser l'option Anticrènelage pour la lisibilité pour le texte en mouvement. Lorsqu'elle est conjuguée à une mise à l'échelle du texte, le texte tente de rester aligné, ce qui produit un effet de décalage. Toutefois, si le contenu de l'objet d'affichage change constamment et qu'il est nécessaire de mettre le texte à l'échelle, vous pouvez améliorer les performances sur les applications mobiles en définissant la qualité sur `LOW`. Au terme du mouvement, redéfinissez la qualité sur `HIGH`.

Processeur graphique

Rendu sur GPU dans les applications Flash Player

Une nouvelle fonction importante de Flash Player 10.1 lui permet d'utiliser le processeur graphique pour effectuer le rendu du contenu graphique sur les périphériques mobiles. Auparavant, l'unité centrale effectuait seule le rendu des graphiques. L'utilisation du processeur graphique optimise le rendu des filtres, des bitmaps, de la vidéo et du texte. Gardez à l'esprit que le rendu par le processeur graphique n'est pas toujours aussi précis que le rendu logiciel. Lorsque vous utilisez le rendu matériel, la définition du contenu à l'écran est moins nette. En outre, une limitation de Flash Player 10.1 peut empêcher le rendu à l'écran des effets Pixel Bender. Lors de l'utilisation de l'accélération matérielle, ces effets sont parfois rendus sous la forme d'un carré noir.

Flash Player 10 possédait une fonction d'accélération du processeur graphique. Il n'utilisait néanmoins pas le processeur pour calculer les graphiques, mais simplement pour les envoyer à l'écran. Flash Player 10.1 utilise le processeur graphique pour le calcul des graphiques, d'où une amélioration sensible de la vitesse de rendu. La charge de travail de l'unité centrale est également réduite, ce qui est utile sur les périphériques aux ressources limitées, tels que les périphériques mobiles.

Le mode processeur graphique est automatiquement activé lors de l'exécution de contenu sur les périphériques mobiles, en vue d'assurer les meilleures performances possibles. Bien qu'il ne soit plus nécessaire de définir `wmode` sur `gpu` pour que le processeur graphique effectue le rendu, l'accélération matérielle est désactivée si vous réglez `wmode` sur `opaque` ou `transparent`.

Remarque : *la version de bureau de Flash Player utilise toujours l'unité centrale pour le rendu logiciel, parce qu'il existe une grande variété de pilotes pour le bureau et qu'ils peuvent accentuer les différences de rendu. Le rendu sur le bureau et sur certains périphériques mobiles peut également être différent.*

Rendu sur GPU dans les applications AIR mobiles

Pour activer l'accélération matérielle des images dans une application AIR, insérez `<renderMode>gpu</renderMode>` dans le descripteur d'application. Il est impossible de modifier les modes de rendu à l'exécution. Sur les ordinateurs de bureau, le paramètre `renderMode` est ignoré ; l'accélération des images via le processeur graphique n'est actuellement pas prise en charge.

Limitations du mode de rendu sur GPU

Il existe certaines limitations lors de l'utilisation du mode de rendu sur GPU dans AIR 2.5 :

- Si le processeur graphique ne parvient pas à effectuer le rendu d'un objet, ce dernier ne s'affiche pas. Il n'existe aucune autre alternative au rendu sur GPU.
- Les modes de fusion suivants ne sont pas pris en charge : Calque, Alpha, Effacement, Superposition, Lumière crue, Eclaircir et Obscurcir.
- Les filtres ne sont pas pris en charge.
- PixelBender n'est pas pris en charge.
- De nombreux processeurs graphiques disposent d'une taille de texture maximale de 1024 x 1024. Dans ActionScript, cela se traduit par la taille de rendu final maximale d'un objet d'affichage après toutes les transformations.
- Adobe ne recommande pas l'utilisation du mode de rendu sur GPU dans les applications AIR compatibles avec la vidéo.

- En mode de rendu sur GPU, les champs de texte ne sont pas toujours déplacés vers un emplacement visible lors de l'ouverture du clavier virtuel. Pour vous assurer que le champ de texte est visible lorsque l'utilisateur saisit du texte, effectuez l'une des opérations suivantes. Placez le champ de texte dans la moitié supérieure de l'écran ou déplacez-le vers la moitié supérieure de l'écran lorsqu'il reçoit le focus.
- Le mode de rendu sur GPU est désactivé sur certains périphériques sur lesquels il ne fonctionne pas correctement. Voir les notes de version du développeur AIR pour obtenir les dernières informations.

Normes de bonnes pratiques en matière de rendu sur GPU

Les conseils suivants permettent d'accélérer le rendu sur GPU :

- Limitez le nombre d'éléments visibles sur la scène. Le rendu de chaque élément et son impact sur les éléments qui l'entourent prennent un certain temps. Si vous ne souhaitez plus afficher un objet d'affichage, définissez sa propriété `visible` sur `false`. Ne vous contentez pas de le déplacer hors de la scène, placez-le derrière un autre objet afin de le masquer ou définissez sa propriété `alpha` sur 0. Si vous n'avez plus besoin de l'objet d'affichage, supprimez-le de la scène à l'aide de la propriété `removeChild()`.
- Réutilisez les objets au lieu de les créer et les détruire.
- Créez des bitmaps dont la taille est inférieure à 2^n x 2^m bits, mais proche de cette valeur. Si les dimensions ne sont pas nécessairement exprimées sous la forme 2 à la puissance n, elles ne doivent ni s'en éloigner, ni dépasser cette valeur. Ainsi, le rendu d'une image de 31 x 15 pixels est plus rapide que celui d'une image de 33 x 17 pixels. (32 et 16 sont des puissances de 2, et 31 et 15 sont des valeurs légèrement inférieures.)
- Dans la mesure du possible, définissez le paramètre repeat sur false lorsque vous appelez la méthode `Graphic.beginBitmapFill()`.
- Ne dessinez pas inutilement. Appliquez la couleur d'arrière-plan en tant qu'arrière-plan. Ne superposez pas les formes de grande taille. A chaque pixel à dessiner correspond une charge de traitement.
- Evitez les formes à pointes longues et fines, les bords qui se recoupent ou un grand nombre de détails le long des bords. Le rendu de ces formes est en effet plus long que celui des objets d'affichage aux bords lisses.
- Limitez la taille des objets d'affichage.
- Activez cacheAsBitMap et cacheAsBitmapMatrix pour les objets d'affichage dont les images ne sont pas mises à jour fréquemment.
- Evitez de créer des graphiques par le biais de l'API de dessin d'ActionScript (classe Graphics). Dans la mesure du possible, créez ces objets de façon statique au moment de la création.
- Mettez à l'échelle les actifs bitmaps à la taille finale avant de les importer.

Mode de rendu sur GPU dans les applications AIR 2.0.3 mobiles

Le rendu sur GPU est soumis à davantage de restrictions dans les applications AIR mobiles créées avec l'outil Packager for iPhone. Le rendu sur GPU n'est efficace que sur les bitmaps, les formes pleines et les objets d'affichage dont la propriété `cacheAsBitmap` est définie. Le processeur peut par ailleurs effectuer le rendu des objets qui pivotent ou sont mis à l'échelle, et dont les propriétés `cacheAsBitmap` et `cacheAsBitmapMatrix` sont activées. Le processeur est utilisé en tandem pour d'autres objets d'affichage, ce qui entraîne généralement des performances de rendu médiocres.

Conseils pour l'optimisation des performances du rendu sur GPU

Bien que le rendu sur GPU améliore nettement les performances du contenu SWF, la conception du contenu joue une rôle primordial. N'oubliez pas que les paramètres ayant prouvé leur efficacité dans le rendu logiciel ne sont parfois pas adaptés au rendu sur GPU. Les conseils suivants peuvent vous aider à optimiser les performances du rendu sur GPU sans entraver celles du rendu logiciel.

Remarque : *Les périphériques mobiles qui prennent en charge le rendu matériel accèdent souvent au contenu SWF via Internet. Il est donc fortement recommandé de tenir compte des conseils suivants lors de la création de contenu SWF afin d'obtenir les meilleurs résultats possibles sur tous les écrans.*

- Evitez d'utiliser les modes `wmode=transparent` ou `wmode=opaque` dans les paramètres embed HTML, car ils peuvent réduire les performances. Ils peuvent également avoir une incidence négative sur la synchronisation audio/vidéo lors du rendu logiciel et matériel. Par ailleurs, de nombreuses plates-formes ne prennent pas en charge le rendu sur GPU lorsque ces modes sont activés, ce qui réduit les performances de façon significative.
- Utilisez uniquement le mode normal et le mode de fusion alpha. Evitez d'utiliser d'autres modes de fusion, notamment le mode de fusion des calques. Il est impossible de reproduire fidèlement tous les modes de fusion lors du rendu sur GPU.
- Lorsqu'un GPU effectue le rendu d'images vectorielles, il rompt ces dernières en maillages composés de petits triangles avant de les dessiner. Ce processus est appelé « réduction en mosaïque ». La réduction en mosaïque implique un faible coût de performances, qui s'accroît au fur et à mesure qu'augmente la complexité de la forme. Pour minimiser l'impact sur les performances, évitez les formes morphes, que le rendu sur GPU réduit en mosaïque sur chaque image.
- Evitez les courbes qui se croisent, les régions fines courbées (comme une fine lune croissante), ainsi que les détails compliqués le long des bords d'une forme. Ces formes sont trop complexes pour que le GPU les réduise en maillages triangulaires. Pour comprendre pourquoi, prenons deux vecteurs : un carré de 500 × 500 et une lune croissante de 100 × 10. Un GPU peut facilement effectuer le rendu du carré, car il est composé uniquement de deux triangles. En revanche, de nombreux triangles sont nécessaires pour décrire la courbe de la lune croissante. Le rendu de la forme est donc plus compliqué même s'il implique moins de pixels.
- Evitez les changements d'échelle trop importants, car ils peuvent également obliger le GPU à réduire une nouvelle fois les images en mosaïque.
- Dans la mesure du possible, évitez de dessiner à l'excès, c'est-à-dire de créer des couches de plusieurs éléments graphiques de façon à les masquer les uns des autres. Grâce au rendu logiciel, chaque pixel est dessiné une seule fois. Ainsi, pour le rendu logiciel, l'application n'affecte pas les performances, quel que soit le nombre d'éléments graphiques qui se chevauchent à un emplacement de pixel. A l'inverse, le rendu logiciel dessine chaque pixel pour chaque élément, que d'autres éléments masquent ou pas cette région. Si deux rectangles se chevauchent, le rendu matériel dessine la région chevauchée deux fois alors que le rendu logiciel ne la dessine qu'une seule fois.

  Ainsi, l'excès de dessin sur le bureau (qui utilise le rendu logiciel) n'a normalement aucun impact sur les performances. De nombreuses formes se chevauchant peuvent néanmoins avoir une incidence négative sur les périphériques qui utilisent le rendu sur GPU. Il est donc conseillé de supprimer les objets de la liste d'affichage au lieu de les masquer.
- Evitez d'utiliser un grand rectangle rempli comme arrière-plan. Définissez plutôt la couleur d'arrière-plan de la scène.
- Dans la mesure du possible, évitez d'utiliser le mode de remplissage bitmap par défaut de la répétition de bitmaps. Utilisez plutôt le mode de verrouillage de bitmaps pour obtenir de meilleures performances.

Opérations asynchrones

Préférez les versions asynchrones aux versions synchrones des opérations, si possible.

Les opérations synchrones s'exécutent immédiatement et le code attend la fin de leur exécution avant de continuer. Elles s'exécutent donc dans la phase code d'application de la boucle d'image. Si une opération synchrone est longue, elle étire la taille de la boucle d'image et l'affichage risque de se bloquer ou d'être saccadé.

Une opération asynchrone, en revanche, ne s'exécute pas nécessairement immédiatement. Votre code et d'autres lignes de code d'application du thread d'exécution actif continuent de s'exécuter. Le moteur d'exécution exécute l'opération dès que possible tout en essayant de parer aux problèmes de rendu. Dans certains cas, l'exécution se déroule en arrière-plan et ne fait pas partie de la boucle d'image. Au terme de l'opération, le moteur d'exécution distribue un événement que le code peut écouter en vue d'exécuter d'autres tâches.

Les opérations asynchrones sont planifiées et divisées pour parer aux problèmes de rendu. Elles permettent donc d'accroître la réactivité de l'application (voir « Perception et réalité en matière de performances » à la page 3 pour plus d'informations).

Les opérations asynchrones nécessitent néanmoins plus de temps système. Dans la pratique, leur exécution peut donc durer plus longtemps, surtout lorsqu'elles sont très courtes.

Dans le moteur d'exécution, de nombreuses opérations sont synchrones ou asynchrones de par leur nature, et il est impossible de choisir leur mode d'exécution. Adobe Air, en revanche, propose trois types d'opérations que vous pouvez exécuter de manière synchrone ou asynchrone, au choix :

- Opérations des classes File et FileStream

  Il est possible d'exécuter un grand nombre des opérations de la classe File de manière synchrone ou asynchrone. Ainsi, il existe une version asynchrone des méthodes de copie ou de suppression de fichier ou de répertoire et d'affichage du contenu d'un répertoire. Le nom de la version asynchrone est identifié par le suffixe « Async ». Pour supprimer un fichier de manière asynchrone, par exemple, appelez la méthode `File.deleteFileAsync()` et non la méthode `File.deleteFile()`.

  Lorsque vous accédez à un fichier en lecture ou en écriture à l'aide d'un objet FileStream, c'est la façon dont vous ouvrez celui-ci qui détermine si les opérations de lecture ou d'écriture sont synchrones ou asynchrones. Utilisez la méthode `FileStream.openAsync()` pour les opérations asynchrones. L'écriture des données est exécutée en mode asynchrone. La lecture des données est effectuée par blocs, elles sont donc disponibles progressivement. En revanche, le mode synchrone de l'objet FileStream lit intégralement le fichier avant de poursuivre l'exécution du code.

- Opérations de base de données SQL locale

  Lorsque vous utilisez une base de données SQL locale, toutes les opérations exécutées via un objet SQLConnection sont en mode synchrone ou asynchrone. Pour spécifier leur exécution en mode asynchrone, ouvrez la connexion à la base de données à l'aide de la méthode `SQLConnection.openAsync()` plutôt que de la méthode `SQLConnection.open()`. Les opérations de base de données asynchrones s'exécutent en arrière-plan. Le moteur de base de données ne s'exécute pas dans la boucle d'image du moteur d'exécution, si bien que les opérations de base de données sont peu susceptibles de donner lieu à des problèmes de rendu.

  Vous trouverez d'autres stratégies d'amélioration des performances avec la base de données SQL locale à la section « Performances de la base de données SQL » à la page 91.

- Shaders autonomes de Pixel Bender

  La classe ShaderJob permet d'exécuter une image ou un ensemble de données via un shader de Pixel Bender et d'accéder aux données résultantes brutes. Lorsque vous appelez la méthode `ShaderJob.start()`, le shader s'exécute de manière asynchrone par défaut. L'exécution a lieu en arrière-plan, pas dans la boucle d'image du moteur d'exécution. Pour imposer l'exécution en mode synchrone de l'objet ShaderJob (ce qui n'est pas recommandé), transmettez la valeur `true` au premier paramètre de la méthode `start()`.

Outre ces mécanismes intégrés permettant d'utiliser le mode asynchrone, vous pouvez aussi structurer votre code de sorte à l'exécuter en mode asynchrone et non synchrone. Si vous programmez une tâche qui risque d'être longue, vous pouvez structurer le code de sorte à exécuter la tâche par blocs. Cette division permet au moteur d'exécution d'effectuer ses opérations de rendu entre les blocs d'exécution du code, réduisant ainsi l'éventualité de problèmes de rendu.

Diverses techniques de division du code sont recensées ci-après. Elles ont pour but principal de vous permettre de programmer du code n'exécutant pas toutes les tâches en une seule fois. Vous assurez le suivi du code et de ses arrêts. Grâce à un mécanisme tel qu'un objet Timer, vous vérifiez à plusieurs reprises s'il reste des tâches et vous les exécutez par blocs jusqu'à ce qu'elles soient toutes terminées.

Pour structurer du code afin qu'il divise les tâches, il est nécessaire de respecter certains modèles établis. Les articles et bibliothèques de code ci-dessous décrivent ces modèles et contiennent du code qui vous permettra de les implémenter dans vos applications :

- Asynchronous ActionScript Execution (Article détaillé de Trevor McCauley, s'accompagnant d'exemples d'implémentation. Disponible en anglais uniquement.)
- Parsing & Rendering Lots of Data in Flash Player (Article détaillé de Jesse Warden, s'accompagnant d'exemples d'implémentation de deux techniques, « patron de conception Builder » et « threads verts ». Disponible en anglais uniquement.)
- Green Threads (Article de Drew Cummins décrivant la technique « threads verts » et s'accompagnant d'un exemple de code source. Disponible en anglais uniquement)
- greenthreads (Bibliothèque de code de type open source créée par Charlie Hubbard pour l'implémentation des « threads verts » dans ActionScript. Disponible en anglais uniquement. Voir greenthreads Quick Start pour plus d'informations. Disponible en anglais uniquement.)
- Threads dans ActionScript 3, à l'adresse http://www.adobe.com/go/learn_fp_as3_threads_fr (Article d'Alex Harui, comprenant un exemple d'implémentation de la technique de « pseudo threading ». Disponible en anglais uniquement.)

Fenêtres transparentes

Dans les applications AIR de bureau, envisagez d'utiliser une fenêtre d'application rectangulaire opaque et non transparente.

Pour créer une fenêtre d'application initiale opaque d'une application AIR de bureau, définissez la valeur suivante dans le fichier descripteur XML de l'application :

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

Si la fenêtre est créée par le code d'application, créez un objet NativeWindowInitOptions et réglez sa propriété `transparent` sur `false` (paramétrage par défaut). Transmettez-le au constructeur NativeWindow pendant la création de l'objet NativeWindow :

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

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

Si la fenêtre est un composant Window Flex, veillez à ce que sa propriété de transparence soit définie sur false (paramétrage par défaut) avant d'appeler la méthode `open()` de l'objet Window.

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

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

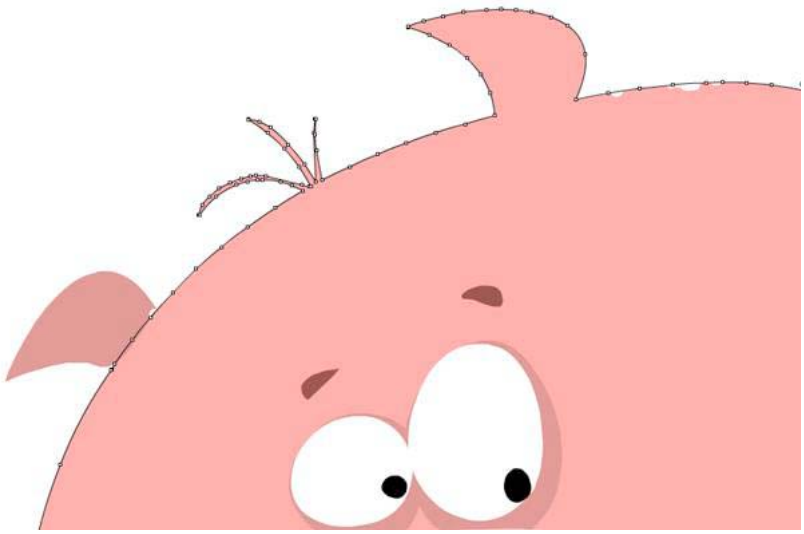
Une partie du bureau de l'utilisateur ou d'autres fenêtres d'application sont potentiellement visibles au travers d'une fenêtre transparente. Le moteur d'exécution consomme donc plus de ressources pour effectuer son rendu. Le rendu d'une fenêtre rectangulaire opaque, qu'elle utilise un chrome système ou personnalisé, est plus simple.

Optez pour une fenêtre transparente uniquement lorsqu'il est important que l'affichage ne soit pas rectangulaire ou pour afficher le contenu en arrière-plan au travers de la fenêtre.

Lissage des formes vectorielles

Lissez les formes pour optimiser les performances de rendu.

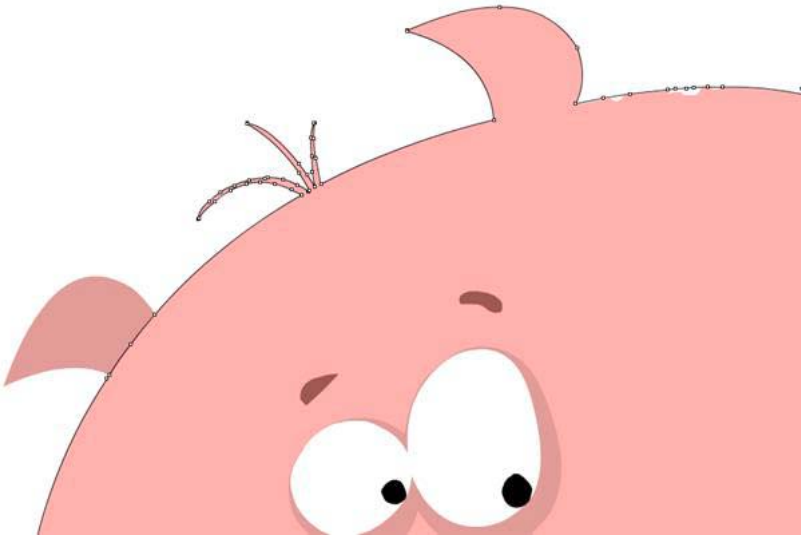
Contrairement aux bitmaps, le rendu du contenu vectoriel requiert de nombreux calculs, notamment pour les gradients et les tracés complexes qui contiennent de nombreux points de contrôle. En qualité de concepteur ou développeur, veillez à ce que les formes soient suffisamment optimisées. La figure suivante illustre les tracés non simplifiés comprenant de nombreux points de contrôle :



Tracés non optimisés

Utilisez l'outil de lissage de Flash Professional pour supprimer les points de contrôle en trop. Un outil équivalent est disponible dans Adobe® Illustrator® ; il est possible d'afficher le nombre total de points et de tracés dans le panneau Informations sur le document.

Le lissage supprime les points de contrôle en trop, ce qui réduit la taille finale du fichier SWF et améliore les performances de rendu. La figure suivante illustre les mêmes tracés après le lissage :



Tracés optimisés

Tant que vous ne simplifiez pas à l'extrême les tracés, cette optimisation ne change rien visuellement. Il est toutefois possible d'améliorer de façon significative la cadence moyenne de votre application finale en simplifiant les tracés complexes.

Chapitre 6 : Optimisation de l'interaction avec le réseau

Amélioration en vue de l'interaction avec le réseau

Flash Player 10.1 et AIR 2.5 offrent une série de nouvelles fonctions destinées à l'optimisation réseau sur toutes les plates-formes, notamment la mise en mémoire tampon circulaire et la recherche dynamique.

Mise en mémoire tampon circulaire

Lors du chargement de contenu multimédia sur des périphériques mobiles, il peut se produire des problèmes qui ne surviennent pratiquement jamais sur un ordinateur de bureau. Il est plus probable, par exemple, que la mémoire ou l'espace disque soit saturé. Lors du chargement d'une vidéo, les versions de bureau de Flash Player 10.1 et d'AIR 2.5 téléchargent et mettent en cache la totalité du fichier FLV (ou fichier MP4) sur le disque dur. Le moteur d'exécution lit ensuite la vidéo à partir du fichier mis en cache. Il est donc rare que l'espace disque vienne à manquer. Si cela se produit, le moteur d'exécution de bureau arrête la lecture de la vidéo.

Sur un périphérique mobile, il arrive beaucoup plus fréquemment que l'espace disque soit saturé. Si tel est le cas, le moteur d'exécution n'arrête pas la lecture comme c'est le cas sur la version de bureau. Il reprend l'écriture du fichier en cache depuis le début de celui-ci. L'utilisateur peut continuer à visionner la vidéo. Il lui est impossible d'effectuer une recherche dans la portion de la vidéo qui a été récrite, sauf au début du fichier. La mise en mémoire tampon circulaire ne démarre pas par défaut. Il est possible de la démarrer au cours de la lecture et au début de celle-ci si la taille de la vidéo est supérieure à l'espace disque ou la mémoire RAM disponible. Le moteur d'exécution nécessite au moins 4 Mo de mémoire RAM ou 20 Mo d'espace disque pour pouvoir utiliser la mise en mémoire tampon circulaire.

Remarque : *si l'espace disque est suffisant sur le périphérique, la version mobile du moteur d'exécution se comporte comme la version de bureau. Une mémoire tampon en RAM est utilisée comme méthode de secours si le périphérique ne possède pas de disque ou si le disque est saturé. Vous pouvez limiter la taille du fichier en cache et de la mémoire tampon en RAM lors de la compilation. La structure de certains fichiers MP4 est telle qu'il est nécessaire de télécharger intégralement le fichier avant que la lecture puisse commencer. Le moteur d'exécution détecte ces fichiers et bloque leur téléchargement si l'espace disque est insuffisant. Dans ce cas, il est donc impossible de les lire. Il peut être préférable de ne pas demander le téléchargement de ces fichiers.*

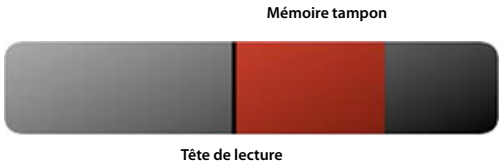
En tant que développeur, pensez qu'il est uniquement possible d'effectuer une recherche au sein du flux en cache. Il arrive que `NetStream.seek()` échoue si le décalage est hors limites, auquel cas un événement `NetStream.Seek.InvalidTime` est distribué.

Recherche dynamique

Remarque : *la fonction de recherche dynamique requiert Adobe® Flash® Media Server 3.5.3.*

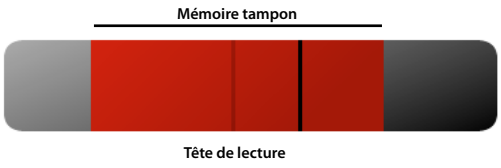
Flash Player 10.1 et AIR 2.5 proposent un nouveau comportement, la recherche dynamique, qui améliore l'expérience de l'utilisateur lors de la lecture de vidéos en flux continu. Si l'utilisateur recherche une destination au sein de la mémoire tampon, le moteur d'exécution réutilise celle-ci pour assurer une recherche instantanée, ce qui n'était pas le cas dans les versions précédentes du moteur d'exécution. Par exemple, si un utilisateur lisait une vidéo à partir d'un serveur de diffusion en continu, que le délai de mise en tampon (`NetStream.bufferTime`) était défini à 20 secondes et que l'utilisateur effectuait une recherche en avant de 10 secondes, le moteur d'exécution supprimait toutes les données en mémoire tampon plutôt que de réutiliser les 10 secondes déjà chargées. Le moteur d'exécution devait alors demander de nouvelles données au serveur beaucoup plus fréquemment, d'où des performances de lecture médiocres sur les connexions lentes.

La figure ci-dessous illustre le comportement de la mémoire tampon dans la version précédente du moteur d'exécution. La propriété `bufferTime` détermine le nombre de secondes à précharger à l'avance pour qu'il soit possible d'utiliser la mémoire tampon sans arrêter la vidéo en cas de perte de la connexion :

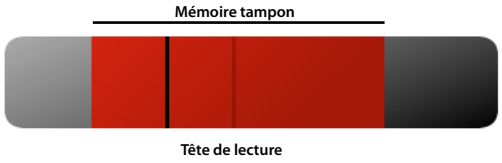


Comportement de la mémoire tampon avant la fonction de recherche dynamique

Grâce à la fonction de recherche dynamique, le moteur d'exécution utilise désormais la mémoire tampon lors d'une recherche en avant et en arrière au sein de la vidéo. Ce nouveau comportement est illustré ci-dessous :



Recherche en avant avec la fonction recherche dynamique



Recherche en arrière avec la fonction recherche dynamique

La recherche dynamique réutilise la mémoire tampon pendant une recherche en avant ou en arrière, garantissant ainsi une lecture plus fluide et rapide. Pour les éditeurs de vidéo, ce nouveau comportement se traduit par des économies de bande passante. Cependant, si la recherche dépasse les limites de la mémoire tampon, le comportement standard est appliqué et le moteur d'exécution demande de nouvelles données au serveur.

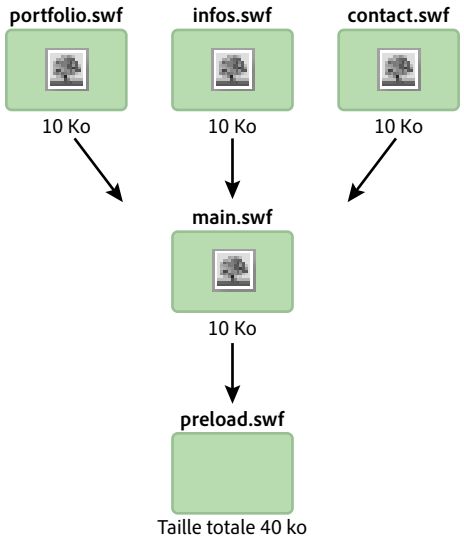
Remarque : *ce comportement ne s'applique pas au téléchargement de vidéo progressif.*

Pour utiliser la recherche dynamique, définissez `NetStream.inBufferSeek` sur `true`.

Contenu externe

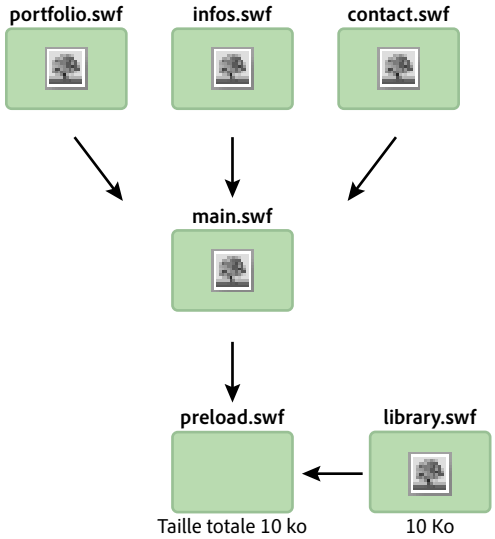
Divisez l'application en plusieurs fichiers SWF.

Les périphériques mobiles ont parfois un accès limité au réseau. Pour charger rapidement le contenu, divisez l'application en plusieurs fichiers SWF. Si possible, réutilisez la logique de programmation et les actifs dans l'ensemble de l'application. Imaginons, par exemple, une application divisée en plusieurs fichiers SWF, comme illustré ci-après :



Application divisée en plusieurs fichiers SWF

Dans cet exemple, chaque fichier SWF contient sa propre copie d'un même bitmap. Il est possible d'éviter cette duplication en utilisant une bibliothèque partagée à l'exécution, comme illustré ci-dessous :



Utilisation d'une bibliothèque partagée à l'exécution

Une bibliothèque partagée à l'exécution est chargée pour que tous les autres fichiers SWF aient accès au bitmap. La classe ApplicationDomain stocke toutes les définitions de classe chargées et les met à disposition, lors de l'exécution, par le biais de la méthode `getDefinition()`.

Une bibliothèque partagée à l'exécution peut également contenir toute la logique de programmation. Il est possible de mettre à jour l'application tout entière sans la recompiler. Le code suivant charge une bibliothèque partagée à l'exécution et extrait la définition que contient le fichier SWF lors de l'exécution. Vous pouvez appliquer cette technique aux polices, bitmaps, sons ou toute 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.");
}
```

Pour faciliter l'extraction de la définition, chargez les définitions de classe dans le domaine d'application du fichier SWF concerné :

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

Pour utiliser les classes que contient le fichier SWF chargé, il suffit maintenant d'appeler la méthode `getDefinition()` sur le domaine d'application actif. Vous pouvez également accéder aux classes par appel de la méthode `getDefinitionByName()`. Parce qu'elle charge les polices et les actifs de grande taille une seule fois, cette technique permet d'économiser de la bande passante. Les actifs ne sont jamais exportés dans d'autres fichiers SWF. Une seule restriction : il est nécessaire de tester et d'exécuter l'application par le biais du fichier loader.swf. Ce fichier charge d'abord les actifs, puis les différents fichiers SWF qui constituent l'application.

Erreurs d'entrée/sortie

Prévoyez des gestionnaires d'événement et des messages d'erreur pour les erreurs d'entrée/sortie.

Sur un périphérique mobile, le réseau n'est pas nécessairement aussi fiable que sur un ordinateur de bureau relié à une connexion Internet haut débit. Sur ce type de périphérique, l'accès au contenu mobile externe est soumis à deux contraintes : la disponibilité et la vitesse. Veillez donc à créer des actifs légers et à ajouter des gestionnaires pour chaque événement IO_ERROR afin d'alerter l'utilisateur.

Prenons un exemple : un utilisateur qui consulte votre site Web à partir de son périphérique mobile perd soudain sa connexion au réseau entre deux stations de métro. Un actif dynamique était en cours de chargement à ce moment-là. Ce cas de figure étant extrêmement rare sur un ordinateur de bureau, vous pouvez utiliser un écouteur d'événements vides pour empêcher l'affichage d'une erreur d'exécution. Sur un périphérique mobile, en revanche, un simple écouteur d'événements vides ne suffit pas pour gérer la situation.

Le code suivant ne répond pas à une erreur d'entrée/sortie. Ne l'utilisez pas tel qu'il se présente :

```
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;
}
```

Il est préférable de gérer un tel échec et de présenter un message d'erreur à l'utilisateur. Le code suivant gère correctement ce cas de figure :

```
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
}
```

Il est recommandé de proposer à l'utilisateur un moyen de charger à nouveau le contenu. Vous pouvez implémenter ce comportement dans le gestionnaire `onIOError()`.

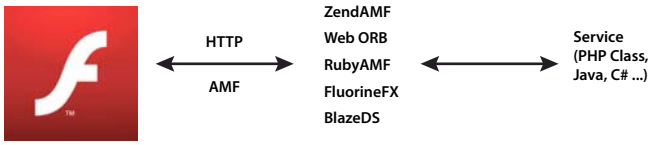
Flash Remoting

Utilisez Flash Remoting et AMF pour optimiser les échanges de données client-serveur.

Vous pouvez charger du contenu distant dans des fichiers SWF à l'aide de XML. Toutefois, XML est constitué de texte brut que le moteur d'exécution charge et analyse. Il est particulièrement adapté aux applications qui chargent un contenu de petite taille. Si vous développez une application chargeant un contenu volumineux, envisagez d'utiliser la technologie Flash Remoting et le format AMF (Action Message Format).

AMF est un format binaire par le biais duquel un serveur et le moteur d'exécution peuvent partager des données. Il réduit la taille des données et améliore la vitesse de transmission. AMF étant un format natif du moteur d'exécution, l'envoi de données AMF au moteur d'exécution permet d'éviter la sérialisation et la désérialisation, opérations gourmandes en mémoire côté client. La passerelle Flash Remoting se charge d'exécuter ces tâches. Lors de l'envoi d'un type de données ActionScript à un serveur, la passerelle gère la sérialisation du côté serveur. Elle vous envoie également le type de données correspondant. Ce type de données est une classe créée sur le serveur qui expose un ensemble de méthodes qu'il est possible d'appeler à partir du moteur d'exécution. Parmi les passerelles Flash Remoting, citons ZendAMF, FluorineFX, WebORB et BlazeDS, une passerelle Flash Remoting Java open source officielle proposée par Adobe.

La figure ci-après illustre le concept Flash Remoting :



Flash Remoting

Dans l'exemple suivant, la classe NetConnection établit une connexion à une passerelle 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 connexion à une passerelle Flash Remoting est simple. Il est toutefois possible de simplifier l'utilisation de Flash Remoting en faisant appel à la classe RemoteObject, qui figure dans le kit de développement SDK d'Adobe® Flex®.

Remarque : *vous pouvez utiliser des fichiers SWC externes, tels ceux issus de Flex framework, dans un projet Adobe® Flash® Professional. Grâce à ces fichiers, vous pouvez vous servir de la classe RemoteObject et de ses dépendances sans recourir au reste du kit SDK de Flex. Les développeurs expérimentés peuvent même communiquer directement avec une passerelle Flash Remoting par le biais de la classe Socket brute, le cas échéant.*

Opérations de réseau superflues

Mettez en cache les actifs localement après leur chargement, au lieu de les charger à partir du réseau chaque fois qu'il est nécessaire d'y accéder.

Si l'application charge des actifs (contenu multimédia ou données, par exemple), enregistrez-les sur le périphérique local pour les mettre en cache. Si les actifs ne changent pas souvent, envisagez de mettre à jour la mémoire cache à fréquence régulière. L'application peut, par exemple, rechercher une nouvelle version d'un fichier image une fois par jour, ou de nouvelles données toutes les deux heures.

Vous disposez de plusieurs méthodes pour mettre en cache les actifs, selon leur type et leur nature :

- Actifs multimédias (images et vidéo, par exemple) : enregistrez les fichiers dans le système de fichiers à l'aide des classes File et FileStream.
- Valeurs de données individuelles ou petits ensembles de données : enregistrez les valeurs en tant qu'objets partagés locaux à l'aide de la classe SharedObject.
- Ensembles de données plus volumineux : enregistrez les données dans une base de données locale ou sérialisez-les et enregistrez-les dans un fichier.

Pour plus d'informations sur la mise en cache des valeurs de données, voir open-source AS3CoreLib project (disponible en anglais uniquement). Ce projet comprend une classe ResourceCache qui effectue les opérations de chargement et de mise en cache.

Chapitre 7 : Utilisation des données multimédias

Vidéo

Pour plus d'informations sur l'optimisation de la vidéo sur les périphériques mobiles, voir l'article Optimize web content for mobile delivery (disponible en anglais uniquement) sur le site Adobe Developer Connection.

Voir en particulier les sections suivantes :

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

Ces sections contiennent des informations sur le développement de lecteurs vidéo pour les périphériques mobiles, telles que :

- Directives d'encodage vidéo
- Normes de bonne pratique
- Méthode de profilage des performances du lecteur vidéo
- Implémentation d'un lecteur vidéo de référence

StageVideo

Utilisez la classe StageVideo si vous souhaitez recourir à l'accélération matérielle pour présenter la vidéo.

Pour plus d'informations sur l'utilisation de l'objet StageVideo, voir Utilisation de la classe StageVideo pour la présentation par accélération matérielle dans le Guide du développeur d'ActionScript 3.0.

Audio

A partir de Flash Player 9.0.115.0 et d'AIR 1.0, le moteur d'exécution prend en charge la lecture des fichiers AAC (AAC Main, AAC LC et SBR). Il est possible de réaliser une optimisation simple en utilisant des fichiers AAC plutôt que des fichiers MP3. A débits égaux, le format AAC garantit une meilleure qualité et des fichiers moins volumineux que le format MP3. La réduction de la taille de fichier permet d'économiser de la bande passante, facteur important sur les périphériques mobiles qui n'offrent pas des connexions Internet haut débit.

Décodage audio via le matériel

Similaire au décodage vidéo, le décodage audio sollicite fortement l'unité centrale. Il est possible de l'optimiser en tirant parti du matériel disponible sur le périphérique. Flash Player 10.1 et AIR 2.5 peuvent détecter et utiliser les pilotes audio matériels en vue d'améliorer les performances lors du décodage de fichiers AAC (profils LC, HE/SBR) ou mp3 (PCM n'est pas pris en charge). L'unité centrale est alors beaucoup moins sollicitée et donc disponible pour d'autres opérations. La batterie est elle aussi moins utilisée.

Remarque : *lors de l'utilisation du format AAC, le profil MP (Main Profile) AAC n'est pas géré sur les périphériques, car les pilotes matériels requis ne sont pas pris en charge sur la plupart de ces derniers.*

Le décodage de l'audio est transparent pour l'utilisateur et le développeur. Lorsque le moteur d'exécution commence à lire les flux audio, il vérifie d'abord le matériel, comme il le fait pour la vidéo. Si un pilote matériel est disponible et le format audio pris en charge, le décodage audio a lieu. Toutefois, même lorsqu'il est possible de gérer le décodage du flux AAC ou mp3 entrant via le matériel, il arrive que celui-ci ne puisse pas traiter tous les effets. Il arrive parfois, par exemple, que ses limitations ne permettent pas de traiter le mixage et le rééchantillonnage.

Chapitre 8 : Performances de la base de données SQL

Structure d'application visant à améliorer les performances de la base de données

Ne modifiez pas la propriété `text` d'un objet SQLStatement après son exécution. Utilisez plutôt une occurrence de SQLStatement pour chaque instruction SQL et définissez des valeurs différentes par le biais de paramètres d'instruction.

Avant d'exécuter une instruction SQL, l'environnement d'exécution la prépare (la compile) pour déterminer les étapes effectuées en interne pour l'exécuter. Lorsque vous appelez `SQLStatement.execute()` sur une occurrence de SQLStatement qui n'a encore jamais été exécutée, l'instruction est automatiquement préparée avant son exécution. Lors des prochains appels à la méthode `execute()`, et tant que la propriété `SQLStatement.text` ne change pas, l'instruction est déjà préparée. Son exécution est donc plus rapide.

Pour optimiser au maximum la réutilisation d'une instruction, si des valeurs doivent être modifiées entre ses exécutions, personnalisez-la à l'aide de paramètres d'instruction (qui sont spécifiés à l'aide de la propriété de tableau associatif `SQLStatement.parameters`) . Contrairement à la modification de la propriété `text` de l'occurrence de SQLStatement, lorsque vous modifiez les valeurs des paramètres d'instruction, le moteur d'exécution n'a pas besoin de préparer à nouveau l'instruction.

Lorsque vous réutilisez une occurrence de SQLStatement, l'application doit conserver une référence à cette occurrence une fois celle-ci préparée. A cet effet, déclarez la variable en tant que variable de domaine de classe et non en tant que variable de domaine de fonction. Pour ce faire, structurez l'application de sorte que l'instruction SQL soit enveloppée dans une seule classe. Un groupe d'instructions exécutées en combinaison peut également être enveloppé dans une même classe (cette technique repose sur l'utilisation du patron de conception Commande). Définies en tant que variables de membre de la classe, les occurrences persistent tant que l'occurrence de la classe enveloppe existe dans l'application. Au minimum, vous pouvez simplement définir une variable contenant l'occurrence de SQLStatement à l'extérieur d'une fonction de sorte que cette occurrence reste en mémoire. Par exemple, déclarez l'occurrence de SQLStatement en tant que variable de membre d'une classe ActionScript ou en tant que variable autre qu'une fonction dans un fichier JavaScript. Vous pouvez ensuite définir les valeurs de paramètres de l'instruction et appeler sa méthode `execute()` lorsque vous souhaitez véritablement exécuter la requête.

Utilisez des index de base de données pour accélérer l'exécution des opérations de comparaison et de tri.

Lorsque vous créez un index pour une colonne, la base de données enregistre une copie des données de celle-ci. Les données sont triées par ordre numérique ou alphabétique. La base de données est ainsi en mesure de faire correspondre des données (utilisation de l'opérateur d'égalité, par exemple) et de trier les résultats à l'aide de la clause `ORDER BY`, et ce, rapidement.

Les index de base de données sont toujours tenus à jour, ce qui entraîne un léger ralentissement des opérations de modification des données (INSERT ou UPDATE) effectuées sur cette table. L'augmentation de la vitesse de récupération des données peut cependant être significative. En raison de ce compromis, évitez tout simplement d'indexer toutes les colonnes d'une table. Adoptez plutôt une stratégie de définition des index. Suivez les directives ci-dessous pour planifier votre stratégie d'indexation :

- Indexez les colonnes utilisées pour la jointure des tables, dans les clauses WHERE ou ORDER BY.

- Si vous utilisez fréquemment des colonnes ensemble, placez-les dans un même index.
- Spécifiez le classement COLLATE NOCASE pour l'index d'une colonne contenant des données texte que vous récupérez en ordre alphabétique.

Envisagez de pré-compiler les instructions SQL pendant les périodes d'inactivité de l'application.

Lors de sa première exécution, une instruction SQL est plus lente, car le texte SQL est préparé (compilé) par le moteur de base de données. Comme la préparation et l'exécution d'une instruction peut être une opération exigeante, il est judicieux de précharger les données initiales, puis d'exécuter les autres instructions en arrière-plan :

1. Chargez les données dont l'application a d'abord besoin..
2. Exécutez les autres instructions au terme des opérations de démarrage initial de l'application ou lors d'une autre période « d'inactivité » de celle-ci.

Supposons, par exemple, que l'application n'accède pas du tout à la base de données lors de l'affichage de son écran initial. Attendez donc que cet écran soit affiché avant d'établir la connexion à la base de données. Enfin, créez les occurrences de SQLStatement et exécutez toutes celles qui sont disponibles.

A l'inverse, supposons que l'application affiche immédiatement certaines données à son démarrage, par exemple le résultat d'une requête particulière. Dans ce cas, continuez et exécutez l'occurrence de SQLStatement pour cette requête. Dès que les données initiales sont chargées et affichées, créez des occurrences de SQLStatement pour les autres opérations de base de données et, si possible, exécutez ultérieurement les autres instructions nécessaires.

En pratique, si vous réutilisez des occurrences de SQLStatement, le système ne doit consacrer du temps supplémentaire à la préparation de l'instruction qu'une seule fois. L'impact sur les performances globales est probablement mineur.

Groupez plusieurs opérations de modification des données SQL dans une même transaction.

Supposons que vous exécutiez un grand nombre d'instructions SQL impliquant l'ajout ou la modification de données (instructions `INSERT` ou `UPDATE`). Vous pouvez augmenter significativement les performances en exécutant toutes les instructions dans une transaction explicite. Si vous ne commencez pas une transaction de façon explicite, chacune des instructions s'exécute dans sa propre transaction créée automatiquement. A l'issue de l'exécution de chaque transaction (chaque instruction), le moteur d'exécution écrit les données résultantes dans le fichier de la base de données sur disque.

Regardez à présent ce qu'il se passe si vous créez explicitement une transaction et exécutez les instructions dans le contexte de cette transaction. L'environnement d'exécution effectue toutes les modifications en mémoire, puis écrit simultanément toutes les modifications dans le fichier de la base de données lorsque la transaction est validée. L'écriture de données sur le disque est généralement la partie la plus longue de l'opération. Par conséquent, écrire sur le disque une seule fois plutôt qu'une fois par instruction SQL peut améliorer significativement les performances.

Lorsque les résultats de la requête `SELECT` sont volumineux, traitez-les par blocs à l'aide des méthodes `execute()` (avec le paramètre `prefetch`) et `next()` de la classe SQLStatement.

Supposons que vous exécutiez une instruction SQL qui extrait un jeu de résultats volumineux. L'application traite ensuite chaque ligne de données dans une boucle. Elle formate les données ou s'en sert pour créer des objets, par exemple. Le traitement des données est susceptible d'être long, ce qui peut entraîner des problèmes de rendu (écran bloqué ou non réactif, par exemple). Pour parer à ces écueils, vous pouvez diviser les tâches par blocs (voir « Opérations asynchrones » à la page 76). L'API de la base de données SQL facilite la division du traitement des données.

La méthode `execute()` de la classe SQLStatement possède un paramètre facultatif, `prefetch` (le premier). Si vous le définissez, il spécifie le nombre de lignes de résultats que renvoie la base de données au terme de l'exécution :

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

Une fois le premier jeu de résultats renvoyé, vous pouvez appeler la méthode `next()` pour poursuivre l'exécution de l'instruction et extraire un autre jeu de lignes de résultat. A l'instar de la méthode `execute()`, la méthode `next()` gère le paramètre `prefetch`, qui permet de spécifier le nombre maximal de lignes à renvoyer :

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

Vous pouvez continuer d'appeler la méthode `next()` jusqu'à ce que toutes les données soient chargées. Comme l'illustre l'exemple précédent, vous pouvez déterminer ce moment en vérifiant la propriété `complete`de l'objet SQLResult, qui est créée chaque fois que la méthode `execute()` ou `next()` prend fin.

Remarque : *utilisez le paramètre `prefetch` et la méthode `next()` pour diviser le traitement des résultats d'une requête. En revanche, ne vous en servez pas pour obtenir un sous-ensemble de résultats. Si seul un sous-jeu des résultats d'une instruction vous intéresse, utilisez la clause `LIMIT` de l'instruction `SELECT`. Si le jeu de résultats est volumineux, rien ne vous empêche de diviser son traitement à l'aide du paramètre `prefetch` et de la méthode `next()`.*

Envisagez d'utiliser plusieurs objets SQLConnection asynchrones avec une seule base de données pour exécuter simultanément plusieurs instructions.

Lorsqu'un objet SQLConnection est connecté à une base de données à l'aide de la méthode `openAsync()`, il s'exécute en arrière-plan et non dans le thread d'exécution principal. En outre, chaque objet SQLConnection s'exécute dans son propre thread en arrière-plan. Si vous utilisez plusieurs objets SQLConnection, vous pouvez exécuter simultanément plusieurs instructions SQL de manière efficace.

Cette technique présente néanmoins quelques inconvénients. En particulier, chaque objet SQLConnection supplémentaire requiert de la mémoire. En outre, les exécutions simultanées sollicitent toujours plus le processeur, surtout sur les machines dotées d'une unité centrale ou d'un cœur d'unité centrale unique. C'est pourquoi cette technique n'est pas recommandée sur les périphériques mobiles.

Autre inconvénient, vous risquez de perdre les avantages que présente potentiellement cette technique, car un objet SQLStatement est lié à un objet SQLConnection unique. Il est donc impossible de réutiliser l'objet SQLStatement si l'objet SQLConnection associé est en cours d'utilisation.

Si vous décidez d'utiliser plusieurs objets SQLConnection connectés à une seule base de données, souvenez-vous que chacun d'eux exécute ses instructions dans une transaction qui lui est propre. Vous devrez tenir compte de ce facteur dans tout code qui effectue des modifications (ajout, modification ou suppression de données, par exemple).

Paul Robertson a créé une bibliothèque de code de type open source qui permet de bénéficier des avantages liés à l'utilisation de plusieurs objets SQLConnection, tout en réduisant au minimum les inconvénients potentiels. Cette bibliothèque utilise un pool d'objets SQLConnection et gère les objets SQLStatement associés. Elle garantit que les objets SQLStatement sont ainsi réutilisés et que plusieurs objets SQLConnection sont disponibles pour exécuter simultanément plusieurs instructions. Pour plus d'informations et pour télécharger la bibliothèque, aller à http://probertson.com/projects/air-sqlite/ (disponible en anglais uniquement).

Optimisation des fichiers de base de données

Evitez de modifier le schéma de la base de données.

Dans la mesure du possible, évitez de modifier le schéma (structure des tables) d'une base de données après avoir ajouté des données dans ses tables. Normalement, un fichier de base de données est structuré avec les définitions de ses tables au début du fichier. Lorsque vous ouvrez une connexion à une base de données, l'environnement d'exécution charge ces définitions. Lorsque vous ajoutez des données dans les tables d'une base de données, ces données sont ajoutées dans le fichier après les données de définition des tables. Cependant si vous modifiez le schéma, les nouvelles données de définition de la table sont mélangées aux données de celle-ci dans le fichier de base de données. L'insertion d'une colonne à une table ou l'ajout d'une table, par exemple, peuvent entraîner le mélange des types de données. Si les données de définition de la table ne figurent pas toutes au début du fichier de base de données, il faut plus longtemps pour établir une connexion à la base de données. En effet, il faut plus longtemps au moteur d'exécution pour lire les données de définition de la table aux différents emplacements du fichier.

Pour optimiser une base de données après avoir modifié son schéma, utilisez la méthode `SQLConnection.compact()`.

Si vous devez modifier le schéma, vous pouvez appeler la méthode `SQLConnection.compact()` à l'issue des modifications. Cette opération restructure le fichier de la base de données de sorte que les données de définition des tables soient regroupées au début du fichier. L'opération `compact()` peut cependant se révéler assez longue, en particulier au fur et à mesure que la taille du fichier de la base de données augmente.

Traitement superflu de la base de données à l'exécution

Utilisez un nom de table complet (comprenant le nom de la base de données) dans l'instruction SQL.

Spécifiez systématiquement le nom de la base de données conjointement avec chaque nom de table dans une instruction. (Utilisez « main » s'il s'agit de la base de données principale.) Le code suivant inclut le nom de base de données explicite `main` :

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

Le fait de spécifier de façon explicite le nom de la base de données évite au moteur d'exécution d'avoir à rechercher la table correspondante dans chaque base de données connectée. Cela lui évite également toute possibilité de se tromper de base de données. Respectez cette règle même lorsque l'occurrence de SQLConnection n'est connectée qu'à une seule base de données. En effet, en arrière-plan, l'occurrence de SQLConnection est également connectée à une base de données temporaire accessible par l'intermédiaire d'instructions SQL.

Utilisez des noms de colonne explicites dans les instructions SQL `INSERT` et `SELECT`.

Les exemples suivants illustrent l'utilisation de noms de colonne explicites :

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

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

Comparez les exemples précédents aux suivants. Evitez ce style de code :

```
-- 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
```

S'ils ne sont pas explicites, il est plus difficile pour le moteur d'exécution d'identifier les noms de colonne. Si une instruction `SELECT` utilise un caractère générique au lieu de colonnes explicites, le moteur d'exécution récupère plus de données. Ces données doivent être traitées et engendrent des occurrences d'objet inutiles.

Evitez de joindre une même table plusieurs fois dans une même instruction, à moins que vous ne la compariez à elle-même.

Au fur et à mesure que les instructions SQL croissent, il peut vous arriver, par mégarde, de joindre plusieurs fois une table de base de données à la requête. Vous pourriez souvent obtenir le même résultant en utilisant la table à une seule reprise. Vous êtes susceptible de joindre une même table plusieurs fois si vous utilisez une ou plusieurs vues dans une requête. Vous pourriez, par exemple, joindre une table à une requête, ainsi qu'une vue contenant les données de la table. Ces deux opérations donneraient lieu à plusieurs jointures.

Syntaxe SQL performante

Pour inclure une table dans une requête, utilisez l'instruction `JOIN` (dans la clause `FROM`) plutôt qu'une sous-requête dans la clause `WHERE`. Ce conseil est valable même si vous souhaitez utiliser les données de la table à des fins de filtrage, pas dans le jeu de résultats.

Il est plus efficace d'effectuer une jointure de plusieurs tables dans la clause `FROM` que d'utiliser une sous-requête dans la clause `WHERE`.

Evitez les instructions SQL qui ne tirent pas parti des index. Il s'agit d'instructions utilisant des fonctions agrégées dans une sous-requête, d'une instruction `UNION` dans une sous-requête ou d'une clause `ORDER BY` conjuguée à une instruction `UNION`.

Un index peut grandement accélérer le traitement d'une requête `SELECT`. Certaines syntaxes SQL, cependant, empêchent la base de données d'exploiter les index, la forçant à effectuer les opérations de recherche et de tri sur les données elles-mêmes.

Dans la mesure du possible, évitez l'opérateur `LIKE`, surtout avec un caractère générique à gauche, comme dans `LIKE('%XXXX%')`.

L'opérateur `LIKE` prend en charge les recherches par caractères génériques. Il est donc plus lent que les comparaisons exactes. En particulier, si la chaîne de recherche commence par un caractère générique, la base de données ne peut absolument pas utiliser les index dans la recherche. Elle est obligée d'examiner tout le texte de chaque ligne de la table.


Dans la mesure du possible, évitez l'opérateur `IN`. Si vous connaissez d'avance les valeurs possibles, il est possible de remplacer l'opérateur `IN` par `AND` ou `OR` pour accélérer l'exécution.

La seconde des deux instructions suivantes s'exécute plus vite, car elle utilise des expressions d'égalité simples conjuguées à `OR`, et non les instructions `IN()` ou `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
```


Envisagez d'utiliser d'autres formes d'une instruction SQL pour améliorer les performances.

Comme illustré dans les exemples précédents, la syntaxe d'une instruction SQL peut affecter les performances de la base de données. Il existe souvent plusieurs manières d'écrire une instruction SQL `SELECT` pour récupérer un jeu de résultats particulier. Une technique s'exécutera parfois sensiblement plus vite qu'une autre. Outre les suggestions précédentes, vous trouverez des informations supplémentaires sur les différentes instructions SQL et leurs performances dans les ressources dédiées du langage SQL.

Performances des instructions SQL


Comparez directement différentes versions d'une instruction SQL pour déterminer la plus rapide.

Pour comparer les performances de plusieurs versions d'une instruction SQL, le mieux consiste à les tester directement avec la base de données et les données.

Les outils de développement suivants indiquent les durées d'exécution des instructions SQL. Servez-vous en pour comparer la vitesse des différentes versions des instructions :

- Run! (Outil de test et de création de requêtes SQL AIR, développé par Paul Robertson. Disponible en anglais uniquement.)
- Lita (SQLite Administration Tool, développé par David Deraedt. Disponible en anglais uniquement.)

Chapitre 9 : Test de performances et déploiement

Test de performances

Un certain nombre d'outils permettent de tester les performances des applications. Vous pouvez utiliser les classes Stats et PerformanceTest, développées par des membres de la communauté Flash, ainsi que le profileur d'Adobe® Flash® Builder™ et l'outil FlexPMD.

Classe Stats

Pour tester votre code à l'exécution à l'aide de la version commerciale du moteur d'exécution, sans outil externe, vous pouvez utiliser la classe Stats développée par mr. doob de la communauté Flash. Vous pouvez télécharger la classe Stats à l'adresse suivante : https://github.com/mrdoob/Hi-ReS-Stats.

La classe Stats permet d'assurer le suivi des éléments ci-dessous :

- Nombre d'images rendues par seconde (plus ce nombre est élevé, mieux c'est).
- Durée de rendu d'une image, en millisecondes (plus ce nombre est faible, mieux c'est).
- Quantité de mémoire utilisée par le code. Si cette quantité augmente à chaque image, il est possible que votre application ait une fuite de mémoire. Il est important d'identifier ce problème potentiel.
- Quantité maximale de mémoire utilisée par l'application.

Une fois la classe Stats téléchargée, vous pouvez l'utiliser avec le code compact ci-dessous :

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

En utilisant la compilation conditionnelle dans Adobe® Flash® Professional ou Flash Builder, vous pouvez activer l'objet Stats :

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

Il vous suffit de basculer la valeur de la constante `DEBUG` pour activer ou désactiver la compilation de l'objet Stats. Vous pouvez appliquer cette technique pour remplacer toute logique de code que vous ne souhaitez pas compiler dans votre application.

Classe PerformanceTest

Pour profiler l'exécution du code ActionScript, Grant Skinner a développé un outil qu'il est possible d'intégrer à un flux de travail de test unitaire. Vous transmettez une classe personnalisée à la classe PerformanceTest, qui exécute une série de tests sur le code. La classe PerformanceTest facilite le test des performances de différentes techniques. Vous pouvez télécharger la classe PerformanceTest à l'adresse suivante : http://www.gskinner.com/blog/archives/2009/04/as3_performance.html.

Profileur Flash Builder

Flash Builder comprend un profileur permettant de tester les performances de votre code, de manière très détaillée.

Remarque : *pour accéder à ce profileur, utilisez la version de débogage de Flash Player, sans quoi vous obtiendrez un message d'erreur.*

Le profileur s'utilise également avec du contenu créé dans Adobe Flash Professional. A cet effet, chargez dans Flash Builder le fichier SWF compilé, depuis un projet ActionScript ou Flex, et exécutez le profileur sur le fichier. Pour plus d'informations sur le profileur, voir « Profilage des applications Flex » dans le guide Utilisation de Flash Builder 4.

FlexPMD

Les services techniques Adobe offrent un outil, appelé FlexPMD, qui permet d'évaluer la qualité du code ActionScript 3.0. FlexPMD est un outil ActionScript, similaire à JavaPMD. FlexPMD améliore la qualité du code ActionScript en évaluant un répertoire source ActionScript 3.0 ou Flex. Il détecte les mauvaises pratiques de programmation, telles que le code inutilisé, trop complexe ou trop long et l'utilisation incorrecte du cycle de vie des composants Flex.

FlexPMD est un projet Open Source d'Adobe disponible à l'adresse suivante : http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD. Un module d'extension Eclipse est également disponible à l'adresse suivante : http://opensource.adobe.com/wiki/display/flexpmd/FlexPMD+Eclipse+plugin.

FlexPMD facilite l'évaluation du code et permet de s'assurer qu'il est correct et optimisé. FlexPMD est un outil d'autant plus puissant qu'il est extensible. En tant que développeur, vous pouvez créer vos propres jeux de règles pour évaluer tout code. Libre à vous, par exemple, de créer un jeu de règles pour détecter l'utilisation excessive de filtres ou toute autre mauvaise pratique que vous souhaitez identifier.

Déploiement

Lors de l'exportation de la version définitive de votre application dans Flash Builder, vous devez vous assurer qu'il s'agit bien de la version commerciale (validée). L'exportation d'une version commerciale supprime toutes les informations de débogage du fichier SWF. Le fichier SWF est ainsi moins volumineux et l'application s'exécute plus rapidement.

Pour exporter la version commerciale de votre projet, accédez au panneau Projet de Flash Builder et sélectionnez l'option Exporter vers une version validée.

Remarque : *lors de la compilation du projet dans Flash Professional, il est impossible de choisir entre la version de débogage et la version commerciale. Le fichier SWF compilé est, par défaut, une version commerciale.*