

# OPTIMISER LES PERFORMANCES POUR LA PLATEFORME ADOBE FLASH



Ce livret vous a été remis en complément du [Programme AS3 Formation](#).

**Il vous est personnel et ne peut pas être redistribué.**

Si vous souhaitez partager directement l'adresse du site web, où j'explique en vidéo comment créer simplement une application flash robuste et évolutive, visitez :

**[Tutoriels Vidéos gratuits pour les Technologies Adobe Flash, Adobe AIR, ActionScript 3](#)**

## Livret de Formation

Ecrit par Matthieu Deloison  
**Formateur ActionScript**

Ce livret est un recueil de mes lectures sur différents blogs comme celui de Thibault Imbert ([www.bytearray.org](http://www.bytearray.org)), celui de Grant Skinner ([www.gskinner.com](http://www.gskinner.com)) et bien d'autres encore...

J'ai mis en pratiques l'ensemble de ces conseils dans mes applications sur des projets professionnels et personnels.

Ce sont **des conseils pratiques et vérifiés dans de "vraies applications flash"**.

## Table des matières

|   |           |
|---|-----------|
| OPTIMISER LES PERFORMANCES.....                                     | 1         |
| POUR LA PLATEFORME.....   | 1         |
| ADOBE FLASH.....  | 1         |
| <b>INTRODUCTION.....</b>  | <b>5</b>  |
| L'Optimisation des Performances pour la Plateforme Adobe Flash..... | 5         |
| <b>L'EXECUTION DU CODE.....</b>                                     | <b>6</b>  |
| Les performances perçues et les performances réelles.....           | 7         |
| Ciblez vos optimisations.....                                       | 9         |
| <b>GERER LA MEMOIRE.....</b>  | <b>11</b> |
| Les Display Objects.....  | 11        |
| Les types primitifs.....  | 12        |
| Réutiliser les objets.....  | 14        |
| Libérer la mémoire.....   | 19        |
| <b>UTILISATION DES BITMAPS.....</b>                                 | <b>22</b> |
| La qualité des Bitmaps.....   | 22        |
| BitmapData single reference.....                                    | 24        |
| Les filtres et le déchargement dynamique de bitmap.....             | 34        |
| Utilisation des effets 3D.....                                      | 36        |
| <b>REDUIRE L'UTILISATION DU CPU.....</b>                            | <b>38</b> |
| Pause et reprise.....   | 38        |
| La gestion des instances.....                                       | 40        |
| Le mode veille.....   | 42        |
| Création et suppression des objets.....                             | 44        |
| Les évènements Activate et deactivate.....                          | 48        |
| <b>LES PERFORMANCES ACTIONSCRIPT 3.0.....</b>                       | <b>50</b> |
| L'API de dessin.....  | 50        |
| La capture et diffusion d'évènements.....                           | 51        |
| Travailler avec les Pixels.....                                     | 55        |
| <b>OPTIMISATIONS PLUS FINES.....</b>                                | <b>58</b> |
| TextField.....  | 58        |
| Les boucles avec TextField.....                                     | 60        |
| Remplacer l'opérateur []......                                      | 62        |

|   |           |
|---|-----------|
| Limiter l'appel à des fonctions.....      | 64        |
| Optimiser les boucles.....                | 66        |
| Les boucles inverses.....                 | 67        |
| <b>OPTIMISATION DE L'APPLICATION.....</b> | <b>68</b> |
| Du contenu externe.....                   | 68        |
| Flash Remoting.....                       | 70        |
| Utiliser le cache de Player Flash.....    | 73        |
| <b>CONCLUSION.....</b>                    | <b>74</b> |
| <b>C'EST MAINTENANT !.....</b>            | <b>75</b> |
| <b>PRISE DE NOTES.....</b>                | <b>76</b> |

# INTRODUCTION

Version 1.1

## **L'Optimisation des Performances pour la Plateforme Adobe Flash**

**Les applications Flash**, y compris Adobe AIR, **fonctionnent sur plusieurs plateformes** comme : le bureau, les smartphones, les tablettes PC et les télévisions.

A travers plusieurs exemples et cas concrets, vous découvrirez avec ce document **les meilleures pratiques pour programmer des applications flash**.

Ce document aborde les sujets suivants :

- **La consommation de la mémoire.**
- **L'utilisation des ressources CPU.**
- **L'amélioration des performances de l'ActionScript 3.**
- L'amélioration de la vitesse du rendu.
- Le travail avec l'audio et la vidéo.
- Des tests et des déploiements d'application.

Plusieurs de ces optimisations peuvent s'appliquer aux applications sur tous les périphériques, incluant l'exécution AIR, le Player Flash. Quelques exceptions et ajouts supplémentaires sont pour les appareils mobiles (smartphones, tablettes).

La majorité de ces optimisations concernent les nouvelles fonctionnalités du Player Flash 10.1 et AIR 2.5. Toutefois, plusieurs de ces optimisations peuvent se mettre en œuvre sur les anciennes versions du Player Flash et d'AIR.

## L'EXECUTION DU CODE

Un élément très important pour comprendre comment améliorer les performances d'une application est de comprendre comment la plateforme flash exécute le code.

Le code exécute en boucle le code de chaque frame. Dans ce cas là, une frame représente une portion de temps, déterminé par le framerate (nombre d'image par seconde) de l'application.

Le temps accordé pour chaque frame dépend directement du framerate.

Par exemple, si vous spécifié un framerate de 30 images par secondes, l'exécution du code sur chaque frame est de 33 ms.

Vous pouvez renseigner le framerate de votre application au moment de la compilation avec Adobe Flash CS5 ou le SDK Flex.

Vous pouvez également renseigner un framerate différent, directement dans le code

**Plus le framerate de votre application est bas, plus les performances seront élevées.**

En fonction du type d'application, **vous pouvez utiliser un framerate de 12fps.**

Par exemple pour les applications sans animation, la lecture de vidéo ou audio, les interfaces utilisateurs avec seulement des boutons de contrôles... Dans ces cas là un framerate de 12 fps est largement suffisant.

Pour les autres types d'applications comme **les jeux flash, les séquences animées**, les interfaces utilisateurs avec des effets de tweens... **un framerate de 30 à 40 fps est nécessaire** pour fournir une expérience satisfaisante à l'utilisateur.

Tout en sachant que le Player Flash 10.2 ne peut pas gérer un framerate supérieur à 60.

Le Player Flash 11 offre encore de meilleures performances et peut gérer un framerate supérieur à 60 fps.

## Les performances perçues et les performances réelles

Au final, **votre application est jugé par les utilisateurs.**

Les développeurs peuvent mesurer les performances de l'application, combien de temps met pour s'exécuter telle ou telle partie, combien d'instances d'objets sont créées...

Toutefois, ces différentes mesures importent peu les utilisateurs.

Les utilisateurs utilisent d'autres critères pour définir la performance d'une application.

Par exemple, est ce qu'une application est rapide, fluide et avec un temps de réponse rapide à mes demandes ?

Est ce que l'application utilise trop les ressources de ma machine ?

**Posez-vous ces questions pour définir la performance perçue de votre application :**

- Est ce que les animations sont fluides ou saccadées ?
- Est ce que le contenu vidéo paraît fluide ou saccadé ?
- Est ce que l'audio est en lecture continue, ou avec une fonction pause / lecture ?
- Est ce que la fenêtre se bloque où se vide pendant les calculs longs.
- Lorsque vous renseignez du texte, est ce fluide ou avec des saccades ?
- Si vous cliquez sur un élément, la réponse est-elle instantanée ou avec un délai ?
- Est ce que le ventilateur du CPU devient bruyant lorsque l'application s'exécute ?
- Sur un ordinateur portable ou un smartphone, est ce que la batterie se décharge rapidement pendant l'exécution de l'application ?
- Est ce que les autres applications deviennent lentes pendant que la notre s'exécute ?

**Il est très important de comprendre la différence entre les performances perçues et les performances réelles.**

Les solutions pour obtenir les meilleures performances perçues sont différentes que pour obtenir de la performance pure.

Assurez-vous de ne pas exécuter trop de codes lors de l'exécution de l'application, ni trop de mise à jour graphiques (animations, effets...).

Dans certains cas, il est nécessaire de diviser votre application en plusieurs parties, entre chaque partie, des mises à jour de l'écran.

**Les techniques fournies dans ce document sont axées sur la performance du code à l'exécution et comment sont perçues ces performances par l'utilisateur.**



## Ciblez vos optimisations

Des améliorations de performances n'impliquent pas forcément une amélioration significative pour les utilisateurs.

Il est important de **se concentrer sur l'optimisation des performances sur fonctions spécifiques qui posent problèmes** dans votre application.

**Plusieurs optimisations sont de bonnes pratiques** et peuvent toujours être mises en œuvres.

Pour les autres optimisations, cela dépend des besoins de votre application et de l'utilisateur.

Par exemple, une application fournit toujours de meilleures performances si elle n'utilise pas d'animation, de vidéo ou d'effets graphiques.

Toutefois, une des raisons de l'utilisation de la plateforme Flash pour créer des applications, est la possibilité d'utiliser des médias, les possibilités graphiques pour concevoir des applications « rich ».

Prenez en compte le niveau d'interactions / effets graphiques, en fonction des caractéristiques de la machines, des appareils sur lesquels s'exécute votre application.

Le premier conseil est d'**éviter d'optimiser « trop tard »**.

Certaines optimisations nécessitent une réécriture du code qui devient difficile à lire ou à faire évoluer. Ce code une fois optimisé est difficile à maintenir dans le temps.

Concernant ce type d'optimisations, il est toujours préférable d'attendre et de déterminer les sections de code nécessitant ce type d'optimisation.

Dans l'idéal, **en réduisant la consommation mémoire d'une application, cela augmente les performances de celle-ci**. Toutefois, cela n'est pas toujours possible.

Par exemple, si une application se bloque durant certaines opérations, la solution est toujours de diminuer les calculs effectués sur chaque frame.

En diminuant les calculs, cela prend moins de ressources pour accomplir le calcul demandé.

Effectivement, il est possible pour l'utilisateur de ne pas percevoir le temps supplémentaire (accordé aux calculs), si l'application continue de répondre à ses interactions et qu'elle ne se bloque plus.

Une chose à savoir à propos de l'optimisation est d'**effectuer des tests de performances**.

## GERER LA MEMOIRE

**L'économie de la mémoire est très important** dans le développement d'une application.

Pour les appareils mobiles (comme les smartphones), c'est la priorité, il est indispensable de limiter la consommation des ressources mémoires par l'application.

### Les Display Objects

*Choisissez le Display Object adéquate.*

ActionScript 3 propose une large gamme de display objects.

**Une des optimisations la plus simple est de limiter l'utilisation de la mémoire en sélectionnant le type de display object approprié.**

- Pour des formes simples, sans interaction, utilisez des objets Shape.
- Pour des objets interactifs, qui n'ont pas besoin de timeline, préférez les objets Sprite.
- Pour les animations qui utilisent la timeline, utilisez les objets MovieClip.

**Choisissez toujours le type d'objet le plus efficace pour votre application.**

Le code ci-dessous montre l'utilisation de la mémoire pour les différents types d'objets.

```
trace(getSize(new Shape()));  
// output: 236  
trace(getSize(new Sprite()));  
// output: 412  
trace(getSize(new MovieClip()));  
// output: 440
```

La méthode **getSize()** montre le nombre de bytes utilisé par un objet en mémoire.

Vous voyez que l'utilisation d'un objet **MovieClip** à la place d'un objet **Shape**, **consomme près du double de mémoire**, surtout si les fonctionnalités d'un MovieClip sont inutiles.

## Les types primitifs

*Utilisez la méthode `getSize()` pour tester votre code et déterminer l'objet le plus efficace pour la tâche.*

Tous les types primitifs utilisent de 4 à 8 bytes en mémoire, sauf String.

**En utilisant un type primitif correct, il est facile 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
```

Pour un nombre, c'est 8 bytes qui sont réservés par la Machine Virtuel ActionScript (AVM) si il n'y a aucune valeur de renseignée.

Tous les autres types primitifs sont stockés sur 4 bytes.

```
// Primitive types
var a:Number = 5;
trace(getSize(a));
// output: 4
a = Number.MAX_VALUE;
trace(getSize(a));
// output: 8
```

Le fonctionnement est différent pour le type String.

La mémoire réservée est en fonction de la taille de la String (chaîne de caractères).

```
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
```

Utilisez la méthode **getSize()** pour tester votre code et ainsi choisir le type primitif le plus efficace pour la tâche à exécuter.

## Réutiliser les objets

*Réutilisez les objets lorsque c'est possible, plutôt que d'en créer d'autres.*

Une autre solution facile pour optimiser la mémoire est de **réutiliser les objets et d'éviter d'en créer de nouveaux** lorsque c'est possible.

```
const MAX_NUM:int = 18;
const COLOR:uint = 0xCCCCCC;
var area:Rectangle;
for (var:int = 0; i < MAX_NUM; i++)
{
    // ne réutilisez pas ce code
    area = new Rectangle(i,0,1,10);
    myBitmapData.fillRect(area,COLOR);
}
```

La recréation de l'objet Rectangle à chaque boucle utilise plus de mémoire, avec un temps de traitement plus long, à cause de la création d'un nouveau objet à chaque boucle.

**Utilisez plutôt ce type de code :**

```
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);
}
```

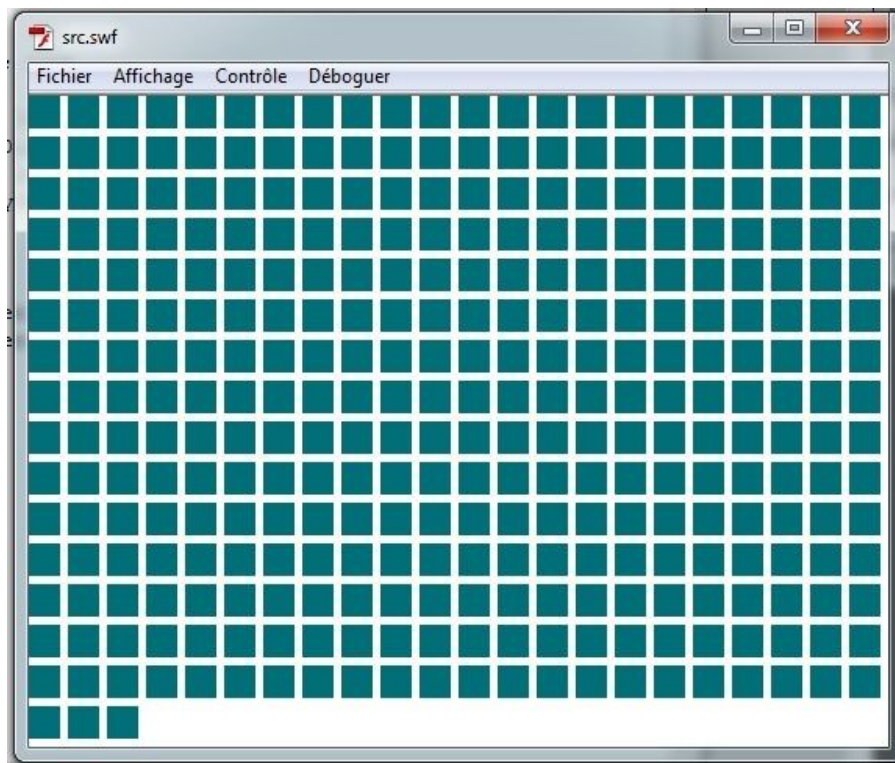
**L'exemple précédent utilise un seul objet, avec un impact très faible sur la mémoire.**

L'exemple suivant montre une **économie importante de mémoire en réutilisant un objet BitmapData**.

```
var myImage:BitmapData;
var myContainer:Bitmap;
const MAX_NUM:int = 333;
for (var i:int = 0; i < MAX_NUM; i++)
{
    // création d'un bitmap de 20 x 20 pixels, non-transparent
    myImage = new BitmapData(20,20,false,0x006f77);
    // création d'un container pour chaque instance BitmapData
    myContainer = new Bitmap(myImage);
    // ajout à la display list
    addChild(myContainer);
    // positionne le container
    myContainer.x = (myContainer.width + 8) * Math.round(i % 22);
    myContainer.y = (myContainer.height + 8) * int(i / 22);
}
```

**Note :** lorsque vous utilisez des valeurs positives, arrondir un nombre avec `int` est plus rapide que d'utiliser la méthode `Math.floor()`.

L'image ci-dessous vous montre le résultat :



Une version optimisée est de créer une seule instance BitmapData puis d'utiliser plusieurs références Bitmap pour obtenir un résultat identique :

```
// création d'un seul bitmap de 20 x 20 pixels, 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++)
{
    // création d'un container avec un référence BitmapData
    myContainer = new Bitmap(myImage);
    // ajout à la display list
    addChild(myContainer);
    // positionne le container
    myContainer.x = (myContainer.width + 8) * Math.round(i % 20);
    myContainer.y = (myContainer.height + 8) * int(i / 20);
}
```

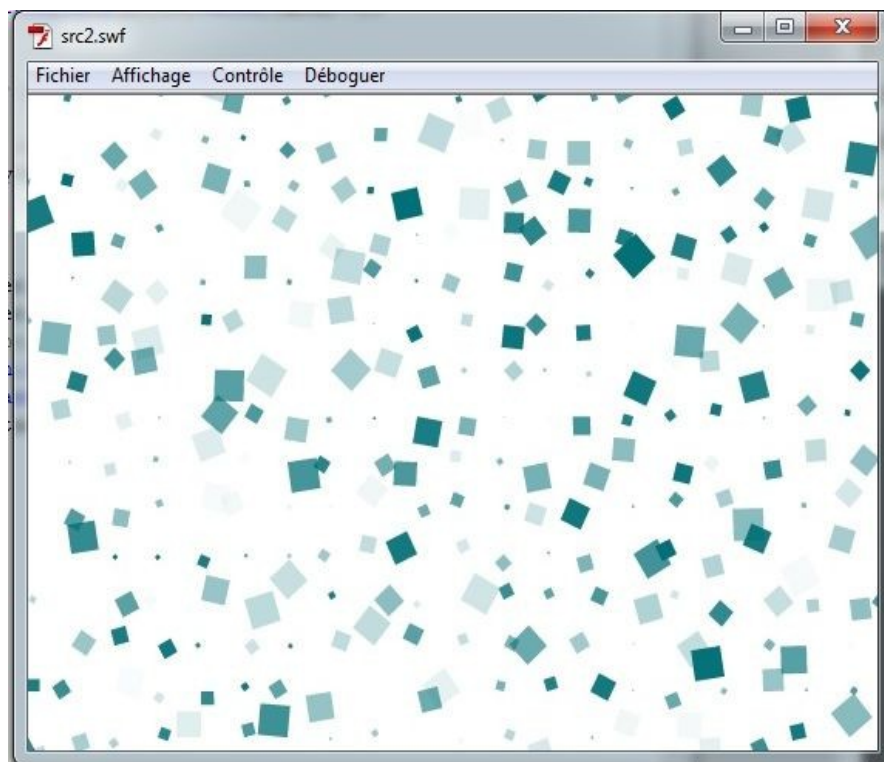


Cette technique économise environ 700 KB de mémoire, ce qui est très significatif pour les appareils mobiles (smartphones).

Chaque container Bitmap peut être manipuler sans modifier le BitmapData d'origine, tout cela en utilisant les propriétés du Bitmap.

```
// création d'un seul bitmap de 20 x 20 pixels, non-transparent
var myImage:BitmapData = new BitmapData(20,20,false,0x06f77);
var myContainer:Bitmap;
const MAX_NUM:int = 333;
for (var i:int = 0; i< MAX_NUM; i++)
{
    // création d'un container avec un référence BitmapData
    myContainer = new Bitmap(myImage);
    // ajout à la display list
    addChild(myContainer);
    // positionne le container
    myContainer.x = (myContainer.width + 8) * Math.round(i % 22);
    myContainer.y = (myContainer.height + 8) * int(i / 22);
    // définition d'une rotation, alpha et scaleX individuel
    myContainer.rotation = Math.random()*360;
    myContainer.alpha = Math.random();
    myContainer.scaleX = myContainer.scaleY = Math.random();
}
```

Et voici le résultat :



## Libérer la mémoire

*Supprimez toutes les références des objets pour vous assurer du passage du garbage collector.*

Vous ne pouvez pas exécuter le garbage collector directement dans le Player Flash.

Pour être certains que qu'un **objet est libéré par le garbage collector**, il faut **supprimer toutes les références à cet objet**.

**Note : vous pouvez appeler directement le garbage collector dans Adobe AIR** et la version debug du Flash Player.

Par exemple, le code suivant met une référence de Sprite à null :

```
var mySprite:Sprite = new Sprite();  
// La référence est égale à null, le garbage collector le retire de la mémoire  
mySprite = null;
```

Rappelez-vous, lorsqu'un objet est null, il n'est pas toujours supprimé de la mémoire.

Parfois, le garbage collector ne s'exécute pas, si la mémoire disponible est considérée comme suffisante.

**Le garbage collector n'est pas prévisible.**

Lorsque le garbage collector s'exécute, il trouve des objets non supprimés.

Il détecte les objets inutilisés, en trouvant des objets référencés par d'autres, mais que l'application n'utilise plus. Tous les objets inutilisés détectés de cette façon sont supprimés.

Dans les grosses applications, ce processus peut utiliser beaucoup de CPU et peut affecter les performances et donc créer un ralentissement dans l'application.

**Essayez de limiter les passages du garbage collector en réutilisant les objets** dès que c'est possible.

Également, **mettez les références à null**, lorsque c'est possible, ainsi le garbage collector s'exécute plus rapidement pour trouver ces objets.

**Note** : mettre un display objet à null n'implique pas sa suppression. L'objet continue de consommer des ressources CPU, même après le passage du garbage collector.

**Assurez-vous d'avoir désactivé (désabonné) votre objet** avant de mettre sa référence à null.

**Le garbage collector peut être exécuter** en appelant la méthode **System.gc()**, disponible dans Adobe AIR et la version debug du Player Flash.

**Note** : si l'objet utilisé est un diffuseur d'évènements, d'autres objets peuvent le référencer. Alors, supprimez les écouteurs en utilisant la méthode **removeEventListener()** avant de mettre sa référence à **null**.

La mémoire utilisée par un Bitmap peut être réduite instantanément.

Par exemple, la classe BitmapData possède une méthode **dispose()**.

L'exemple suivant crée une instance BitmapData de 1,8 MB.

L'utilisation de la mémoire grimpe de 1,8 MB.

```
trace(System.totalMemory / 1024);  
// output: 43100  
// création d'une instance BitmapData  
var image:BitmapData = new BitmapData(800, 600);  
trace(System.totalMemory / 1024);  
// output: 44964
```

Ensuite, le BitmapData est supprimé manuellement.

```
trace(System.totalMemory / 1024);  
// output: 43100  
// création d'une instance BitmapData  
var image:BitmapData = new BitmapData(800, 600);  
trace(System.totalMemory / 1024);  
// output: 44964  
image.dispose();  
image = null;  
trace(System.totalMemory / 1024);  
// output: 43084
```

Pour que la méthode **dispose()** supprime les pixels de la mémoire, la référence doit être mise à **null**.

Toujours appeler la méthode **dispose()** et mettre la référence à **null** lorsque vous n'avez plus besoin de l'objet BitmapData, alors la mémoire est libérée immédiatement.

## UTILISATION DES BITMAPS

**L'utilisation de vecteurs à la place des bitmaps est une bonne solution pour économiser la mémoire.**

Par contre, l'utilisation de vecteurs, surtout les complexes, augmente fortement les besoins en ressources CPU.

**L'utilisation des bitmaps est une bonne solution pour optimiser le rendu**, par qu'à l'exécution de l'application, il y a besoin de moins de ressources pour dessiner des pixels sur l'écran que pour le rendu de contenu vectoriel.

### La qualité des Bitmaps

Pour une meilleure utilisation de la mémoire, les images opaques 32 bits sont réduites à 16 bits lorsque le Player Flash détecte un écran 16 bits.

Cette baisse de qualité consomme moitié moins de ressources mémoire, et le rendu des images est plus rapide.

Cette fonctionnalité est seulement disponible avec le Flash Player 10.1 pour Windows Mobile.

**Note :** avant le Player Flash 10.1, tous les pixels créés en mémoire étaient stockés sur 32 bits (4 bytes).

Un logo simple de 300x300 pixels utilisait 350 KB de mémoire ( $300 \times 300 \times 4 / 1024$ ).

Avec cette nouvelle option, le même logo opaque utilise seulement 175 KB.

Si le logo est transparent, il ne peut pas être réduit à 16 bits, il utilise donc la même mémoire (350 KB).

Cette fonctionnalité peut s'appliquer seulement pour les bitmaps embarquées ou les images chargées au démarrage (PNG, GIF, JPG).

Sur les appareils mobiles, il est difficile de différencier un rendu d'image en 16 bits et le même rendu en 32 bits.

Avec une simple image composée de quelques couleurs, il n'y a pas de différence visible.

Pour la majorité des images plus complexes, il est difficile de détecter une différence.

Toutefois, il peut y avoir une dégradation des couleurs lorsqu'il y a un zoom sur une image, une image 16 bits peut être moins lisse qu'une en 32 bits.

## BitmapData single reference

Une optimisation importante est d'**utiliser, aussi souvent que possible, les instances BitmapData.**

Le Player Flash 10.1 et AIR 2.5 ajoute une nouvelle fonctionnalité, pour toutes les plateformes, qui s'appelle « BitmapData single reference ».

Lorsque vous créez des instances BitmapData à partir d'une image embarquée, une seule version de ce Bitmap est utilisée toutes les instances BitmapData.

Si un bitmap est modifié plus tard, cela donne un bitmap unique dans la mémoire.

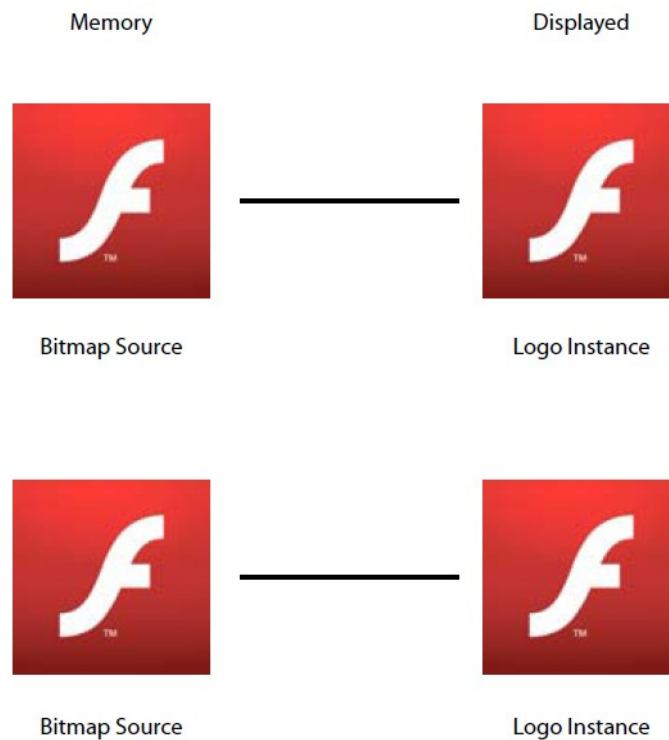
L'image embarquée peut provenir de la bibliothèque ou du tag [embed].

**Note :** le contenu existant profite de cette nouvelle fonctionnalité, parce que le Player Flash 10.1 et AIR 2.5 réutilise automatiquement les bitmaps.

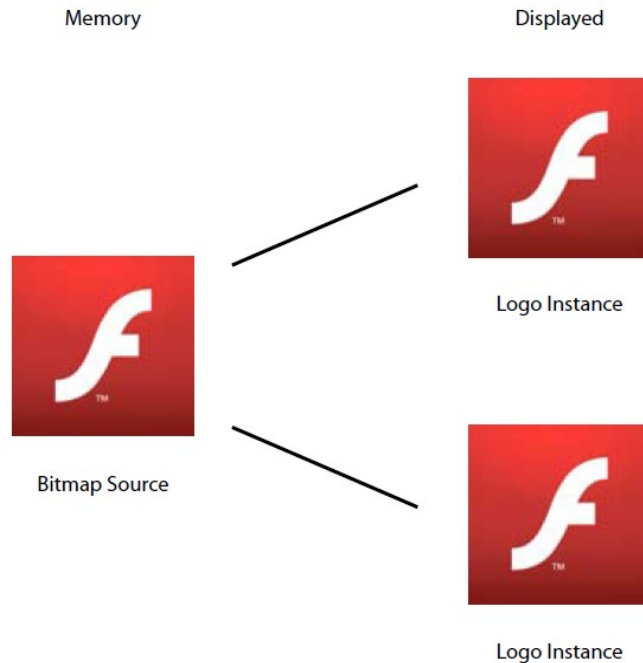


Lorsqu'une image est embarquée, une bitmap associée est créée en mémoire.

Avant le Player Flash 10.1 et AIR 2.5, chaque instance donnée avait une bitmap séparée en mémoire.



Dans le Flash Player 10.1 et AIR 2.5, lorsque de multiples instances de bitmap sont créées, une seule version de bitmap est utilisée pour toutes les instances BitmapData.

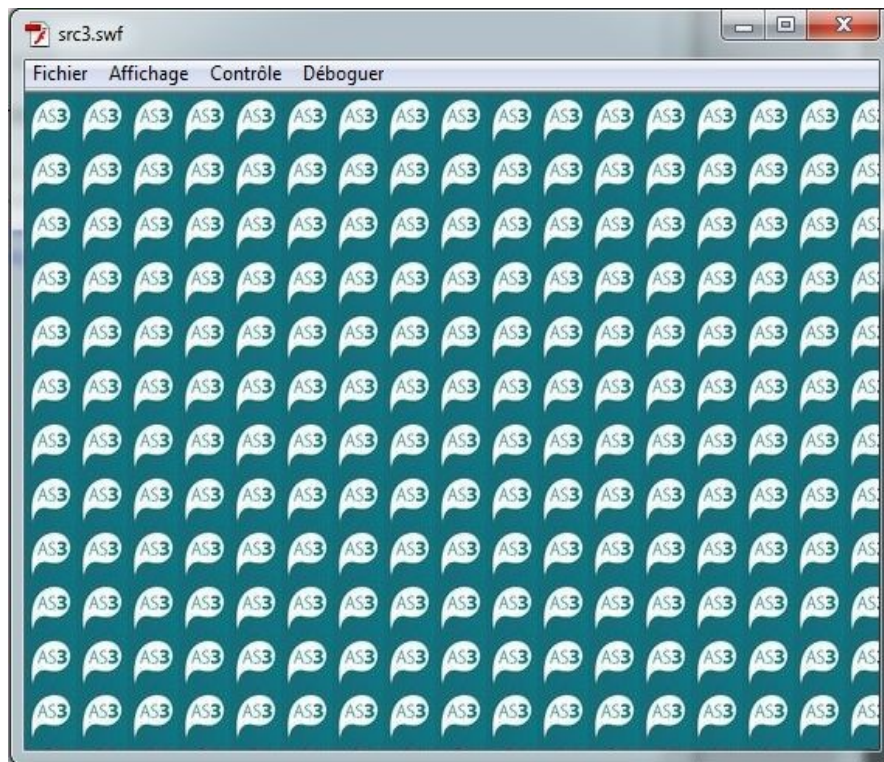


Cette approche réduit efficacement la mémoire utilisée par une application avec beaucoup de bitmaps.

L'exemple ci-dessous crée plusieurs instances du symbol **as3** :

```
const MAX_NUM:int = 18;
var oLogo:BitmapData;
var bitmap:Bitmap;
for (var i:int = 0; i<MAX_NUM; i++)
{
    for (var j:int = 0; j<MAX_NUM; j++)
    {
        oLogo = new LogoAS3(0,0);
        bitmap = new Bitmap(oLogo);
        bitmap.x = j * oLogo.width;
        bitmap.y = i * oLogo.height;
        addChild(bitmap)
    }
}
```

Et voici le résultat de ce morceau de code :



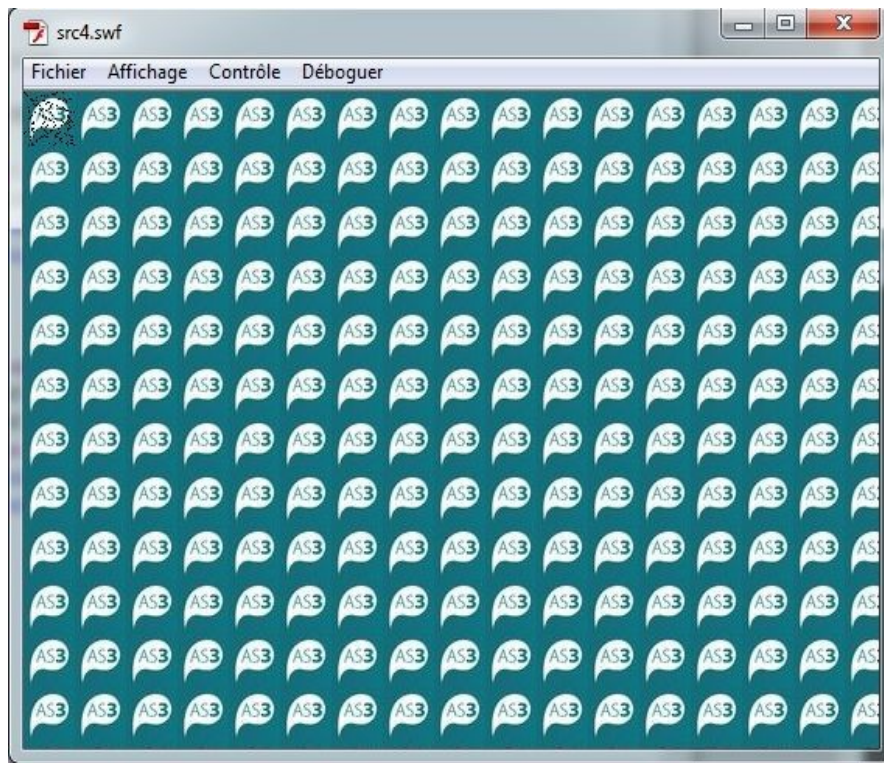
Avec le Player Flash 10, cette application utilise environ 1 008 KB en mémoire.

Avec le Player Flash 10.1, sur le bureau ou un appareil mobile, cette application utilise seulement 4 KB de mémoire.

Le code suivant modifie une seule instance BitmapData :

```
const MAX_NUM:int = 18;
var oLogo:BitmapData;
var bitmap:Bitmap;
for (var i:int = 0; i<MAX_NUM; i++)
{
    for (var j:int = 0; j<MAX_NUM; j++)
    {
        oLogo = new LogoAS3(0,0);
        bitmap = new Bitmap(oLogo);
        bitmap.x = j * oLogo.width;
        bitmap.y = i * oLogo.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, 0xffffffff);
```

L'image ci-dessous montre la modification d'une instance as3 :

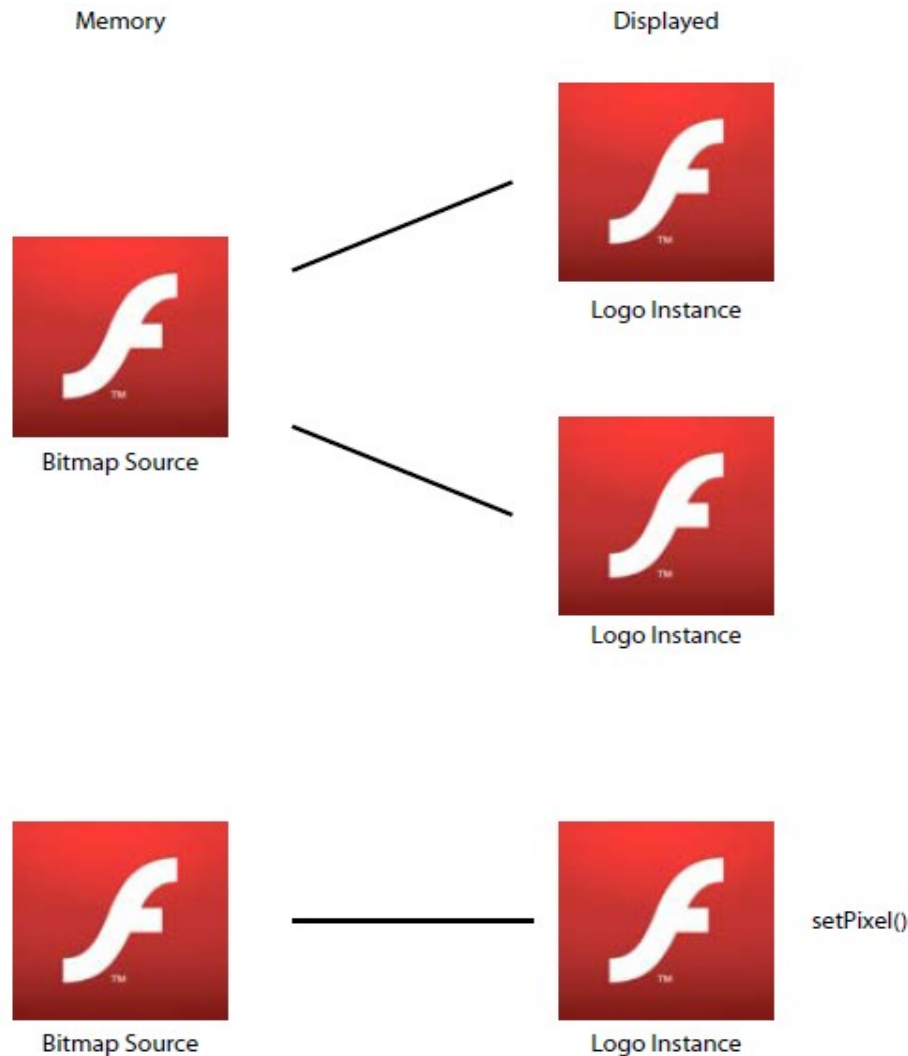


En interne, le runtime assigne et crée automatiquement la bitmap en mémoire, pour prendre en compte les modifications de pixels.



Lorsqu'une méthode de la classe BitmapData est appelée, comme la modification de pixels, une nouvelle instance est créée en mémoire et les autres instances ne sont pas mises à jour.

La figure ci-dessous vous explique le concept :



Si un logo AS3 est modifié, une nouvelle copie est créée dans la mémoire.

Le résultat, est que l'application utilise, environ, 8 KB de mémoire, avec le Player Flash 10.1 et AIR 2.5.

Dans l'exemple précédent, chaque bitmap était disponible individuellement pour des modifications.

Pour créer un seul effet, la méthode `beginBitmapFill()` est la plus appropriée.

```
var container:Sprite = new Sprite();  
var source:BitmapData = new Star(0,0);  
// remplit le contenu avec le BitmapData d'origine  
container.graphics.beginBitmapFill(source);  
container.graphics.drawRect(0,0,stage.stageWidth,stage.stageHeight);  
addChild(container);
```

Cette technique fournit un résultat identique avec une seule instance `BitmapData`

Pour la rotation du logo en continue, au lieu d'accéder à chaque instance du logo, utilisez un objet `Matrix` en rotation sur chaque frame.

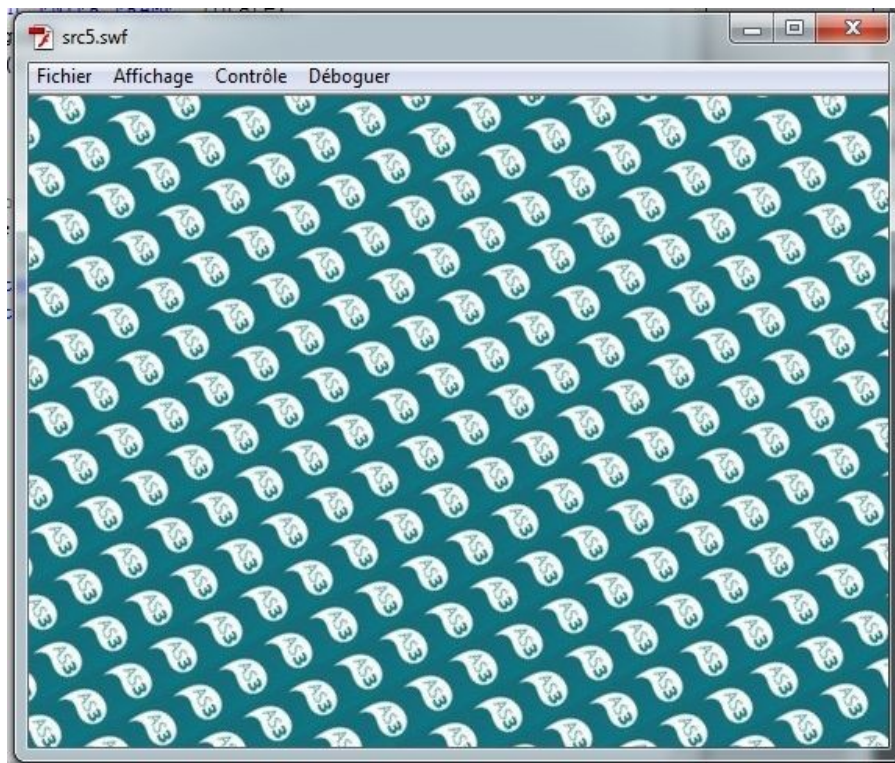
Affectez l'objet Matrix à la méthode `beginBitmapFill()`.

```
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
{
    matrix.rotate(angle); // rotation du logo
    container.graphics.clear(); // efface le contenu
    // Remplit le contenu avec le BitmapData d'origine
    container.graphics.beginBitmapFill(source,matrix,true,true);
    container.graphics.drawRect(0,0,stage.stageWidth,stage.stageHeight);
}
```

En utilisant cette technique, il n'y a pas besoin de boucle en ActionScript pour créer l'effet. Le runtime effectue cela en interne.



L'image suivante donne le résultat obtenu :



Avec cette technique, la mise à jour du BitmapData d'origine met à jour automatiquement ceux qui l'utilisent sur le stage, ce qui en fait une technique puissante.

**Note :** lorsque vous utilisez plusieurs instances de la même image, le dessin dépend de la classe associée au bitmap d'origine dans la mémoire.

Si aucune classe n'est associée, les images sont dessinées comme des formes.

## Les filtres et le déchargement dynamique de bitmap

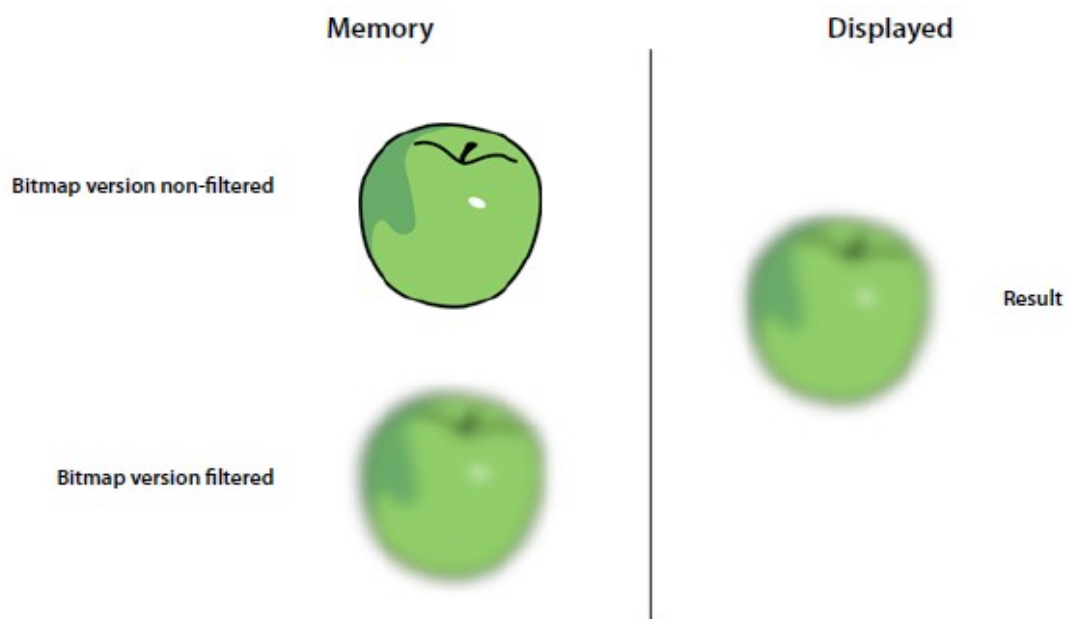
*Evitez les filtres, inclut les filtres produit par Pixel Blender.*

**Essayez d'utiliser le moins possible les filtres**, y compris les filtres sur les appareils mobiles avec Pixel Blender.

Lorsqu'un filtre est appliqué à un display object, le runtime crée 2 bitmaps dans la mémoire.

Ces bitmaps ont chacune la taille d'un display object.

La première est une version normale de l'image, la deuxième une version le filtre appliqué.



Lorsque les propriétés d'un filtre sont modifiées, les 2 bitmaps sont mis à jour pour créer le bitmap de résultat.

Ce processus utilise des ressources CPU et les 2 bitmaps utilisent une taille en mémoire significative.

Le Player Flash 10.1 et AIR 2.5 possède un nouveau système de filtre sur toutes les plateformes.

Si le filtre n'est pas modifié pendant 30 secondes, ou si il est caché de l'écran, la mémoire utilisée par le bitmap non filtré est libérée.

Cette fonctionnalité permet d'économiser la mémoire utilisée par les filtres sur toutes les plateformes.

Par exemple, prenez un objet texte avec un filtre de flou appliqué. Le texte utilisé dans ce cas, pour un simple affichage n'est pas modifié.

Après 30 secondes, la version bitmap non filtrée est supprimée de la mémoire.

Le même principe de libération de mémoire est mis en œuvre pour un texte caché pendant plus de 30 secondes, ou en dehors de l'écran.

Lorsqu'une propriété d'un filtre est modifiée, la version bitmap non filtrée est recrée.

Cette fonctionnalité s'appelle le déchargement dynamique de bitmap.

**Malgré ces optimisations, soyez prudent avec les filtres, ils ont besoin de beaucoup de ressources CPU lorsqu'ils sont modifiés.**

**Une meilleure solution est d'utiliser des bitmaps créés avec un logiciel comme Adobe Photoshop, pour simuler des filtres.**

**Évitez d'utiliser la création de bitmaps au démarrage d'ActionScript.**

En utilisant des bitmaps « externes », cela aide le runtime à réduire l'utilisation de ressources CPU, surtout lorsque les propriétés des filtres ne changent pas continuellement.

Dans la mesure du possible, **créez les effets dont vous avez besoin sur un bitmap dans un logiciel comme Photoshop.**

Vous pouvez afficher le bitmap au runtime (démarrage) sans utiliser de processus important, et donc cela peut être plus rapide.

## Utilisation des effets 3D

*Préférez la création manuelle des effets 3D.*

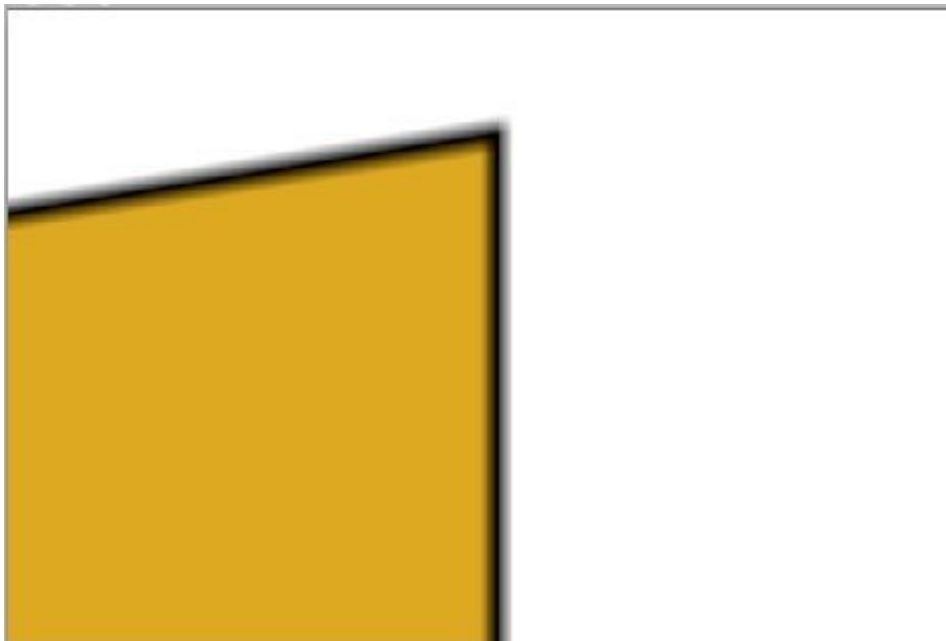
Le Player Flash 10 et AIR 1.5 possède un moteur 3D qui vous permet de créer des effets de perspective sur les display objects.

Vous pouvez appliquer ces transformations en utilisant les propriétés **rotationX** et **rotationY** ou avec la méthode **drawTriangles()** de la classe Graphics.

Vous pouvez modifier la profondeur avec la propriété **z**.

Rappelez-vous que chaque transformation de perspective d'un display object est considéré comme un bitmap et nécessite plus de mémoire.

L'image ci-dessous montre l'anti aliasing produit lors de l'utilisation d'une perspective :



L'anti-aliasing est le résultat d'un contenu vectoriel considéré dynamiquement comme un bitmap.

L'anti-aliasing arrive lorsque vous utilisez des effets 3D.

Si vous créez les effets manuellement, vous pouvez réduire l'utilisation de la mémoire.

Toute fois, les nouvelles fonctionnalités 3D de Flash Player 10 et AIR 1.5 rend la création de texture plus facile comme avec la méthode **drawTriangles()**.

Lorsque le CPU effectue les transformations 3D, considérez que l'application de celles-ci nécessite 2 bitmaps dans la mémoire.

Une bitmap est l'originale et la deuxième représente la version en perspective.

Les transformations 3D fonctionnent comme les filtres.

Utilisez les propriétés 3D avec précaution lorsque le CPU effectue des transformations 3D.

## REDUIRE L'UTILISATION DU CPU

Le Player Flash possède 2 nouvelles fonctionnalités pour vous aider à économiser les ressources CPU.

**Elles concernent la pause et la reprise du contenu SWF lorsqu'il n'est plus dans le focus, et en limitant le nombre d'instance du Player Flash sur une page.**

### Pause et reprise

*La fonctionnalité pause et reprise ne s'applique pas aux applications Adobe AIR.*

Pour optimiser l'utilisation du CPU et de la batterie, le Player Flash 10.1 ajoute une nouvelle fonctionnalité sur les appareils mobiles (netbooks, smartphones) pour les instances inactives.

**Cette fonctionnalité permet de limiter l'utilisation du CPU en mettant en pause puis reprise le fichier SWF lorsque son contenu est en dehors ou sur l'écran.**

Grâce à cette fonctionnalité, le Player Flash libère autant de mémoire que possible, en supprimant tous les objets qui peuvent être recréés lorsque le contenu est remis en lecture.

Le SWF est considéré comme « éteint » lorsque le contenu est en dehors de l'écran, du focus.

2 possibilité pour que le contenu du SWF soit considéré en dehors de l'écran :

- L'utilisateur scrolle la page (avec une barre défilante type scrollbar) et le contenu SWF se retrouve en dehors de l'écran, donc caché.

Dans ce cas, si il y a de l'audio ou de la vidéo, ils sont toujours en lecture, mais le rendu graphique est stoppé.

Si il n'y a ni audio, ni vidéo en lecture, pour vous assurer que l'exécution de l'ActionScript ne soit pas en pause, mettez le paramètre HTML **hasPriority** à true.

Rappelez-vous que le rendu du contenu SWF est mis en pause lorsque le contenu est en dehors de l'écran ou caché, en fonction de la valeur du paramètre HTML **hasPriority**.

- Un onglet est ouvert dans le navigateur, ce qui met le SWF en fond.

Dans ce cas, en fonction du paramètre HTML **hasPriority**, le contenu du SWF est ralenti à 2 fps. L'audio et la vidéo sont arrêtés et il n'y a aucun calcul de rendu tant le contenu du SWF ne redevient pas visible.



## La gestion des instances

*Note : la gestion des instances n'est pas possible avec les applications Adobe AIR.*

Utilisez le paramètre HTML **hasPriority** pour décaler le chargement des différents fichiers SWF.

Le Player Flash possède un nouveau paramètre HTML qui s'appelle **hasPriority**.

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

Ce paramètre limite le nombre d'instances du Player Flash démarrée sur la même page.  
En limitant le nombre d'instances, cela permet d'économiser le CPU et la batterie.

Le principe est de définir des priorités dans les contenus SWF sur une page.

Prenons un exemple simple : un utilisateur parcourt un site web et la page d'accueil a 3 fichiers SWF différents. Un seul est visible, un autre partiellement et le dernier n'est pas visible sur la page (besoin de scroller).

Les deux premières animations démarrent normalement, la dernière est décalée car elle n'est pas visible.

Ce fonctionnement est celui par défaut lorsque le paramètre **hasPriority** n'est pas renseigné ou est à **false**.



Pour être certain qu'un swf démarre, même s'il n'est pas visible, mettez le paramètre **hasPriority** à true.

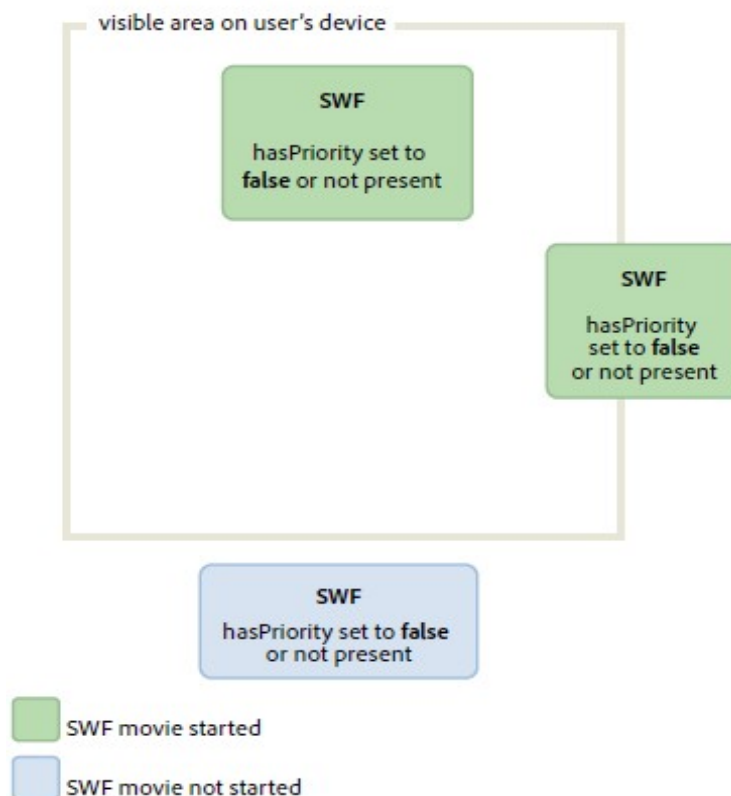
Faites attention à la valeur de **hasPriority**, un fichier SWF qui n'est pas visible pour l'utilisateur a toujours son rendu en pause.

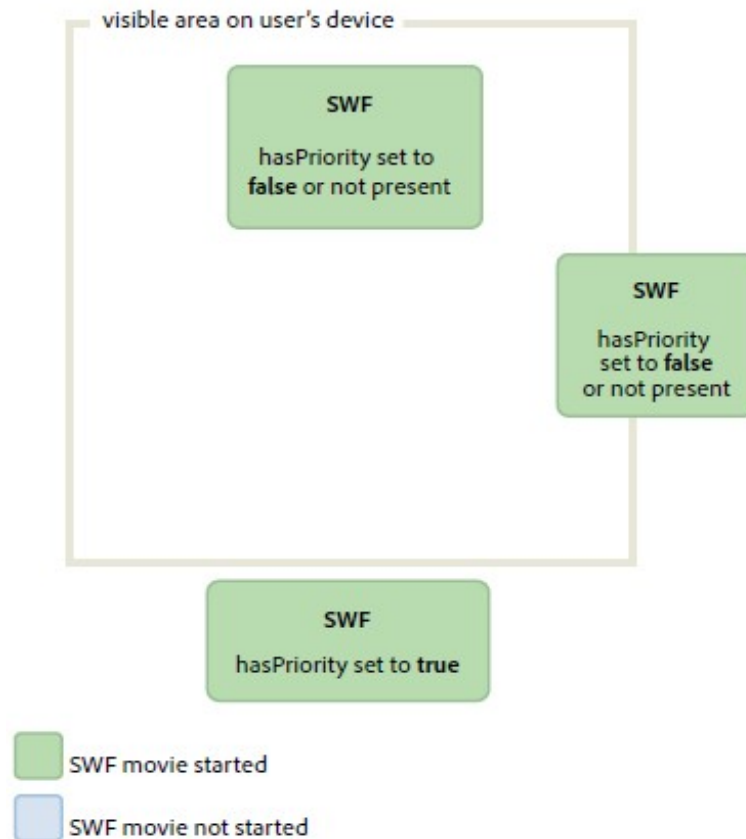
**Note :** si les ressources CPU sont faibles Les instances du Player Flash ne démarrent pas automatiquement sauf si le paramètre **hasPriority** est mis à **true**.

Si de nouvelles instances sont créées en javascript, après le chargement de la page, ces instances ne tiennent pas compte du paramètre **hasPriority**.

Egalement, les fichiers SWF peuvent être démarrés par un click. Cette fonctionnalité est communément appelée "click to play".

Les images suivantes montrent l'influence du paramètre **hasPriority** à différentes valeurs.





## Le mode veille

Le Player Flash 10.1 et AIR 2.5 implémentent une nouvelle fonctionnalité sur les appareils mobiles pour économiser des ressources CPU et la durée de vie de la batterie.

Par exemple, si l'exécution d'une application mobile est interrompue et arrête d'utiliser l'appareil, le runtime le détecte la mise en veille.

Alors le framerate passe à 4 fps et le rendu est mis en pause.

Pour les applications AIR, le mode veille démarre lorsque l'application est mise en tâche de fond.

Le code ActionScript continue de s'exécuter en mode veille, comme si la propriété **stage.frameRate** est mise à 4 fps. Mais l'étape de rendu est sauté, donc l'utilisateur ne voit pas que l'application tourne à 4 fps.

Un framerate de 4 fps est choisi, au lieu de 0, car il permet toutes les communications sortantes (NetStream, Socket et NetConnection).

Le passage à 0 fps ferme toutes les connections.

Un rafraichissement toutes les 250 ms (4 fps) a été choisi car les constructeurs utilise cette fréquence de rafraichissement.

**Note** : lorsque le runtime est en mode veille, la propriété **stage.frameRate** renvoie le framerate d'origine du fichier SWF, au lieu de 4 fps.

Lorsque le système (du smartphone) sort du mode veille, le rendu reprend. Le framerate reprend sa valeur d'origine.

**Note** : Aucun évènement ActionScript n'est envoyé lorsque le runtime passe en mode veille.

## Création et suppression des objets

*Créez et supprimez vos objets proprement en utilisant les évènements `REMOVED_FROM_STAGE` et `ADDED_TO_STAGE`.*

**Pour optimiser votre code, supprimez toujours vos objets.**

La suppression est très importante pour tous les objets surtout pour les display objects.

Surtout **si un display object n'est plus affiché dans la display list et en attente du garbage collector, il continue de consommer des ressources CPU.**

Par exemple, il peut être appelé par **`Event.ENTER_FRAME`**.

Il est donc important de supprimer les objets avec les évènements **`Event.REMOVED_FROM_STAGE`** et **`Event.ADDED_TO_STAGE`**.

Pour que la suppression d'un objet s'effectue correctement par le garbage collector, sa référence ne doit plus être appelée dans le code.

Faites attention à vos évènements, notamment qui peut **`Event.ENTER_FRAME`** appeler un display object.

Prenons un exemple :

```
// Affiche ou retire un logo en fonction du bouton cliqué
showBtn.addEventListener(MouseEvent.CLICK,showIt);
removeBtn.addEventListener(MouseEvent.CLICK,removeIt);
function showIt(e:MouseEvent):void
{
    addChild (logoAS3);
}
function removeIt(e:MouseEvent):void
{
    if (contains(logoAS3)) removeChild(logoAS3);
}
// Ecoute des évènement Event.ADDED_TO_STAGE et Event.REMOVED_FROM_STAGE
// sur logoAS3
logoAS3.addEventListener(Event.ADDED_TO_STAGE,activate);
logoAS3.addEventListener(Event.REMOVED_FROM_STAGE,deactivate);

function activate(e:Event):void
{
    // A chaque frame, appelle d'une fonction qui fait bouger le logoAS3.
    e.currentTarget.addEventListener(Event.ENTER_FRAME,handleMovement);
}
function deactivate(e:Event):void
{
    // Suppression de l'évènement Event.ENTER_FRAME
    // logoAS3 arrête de se déplacer
    // la consommation du CPU baisse
    e.currentTarget.removeEventListener(Event.ENTER_FRAME,handleMovement);
    e.currentTarget.stop();
}
```

Lorsque l'utilisateur appuie sur le bouton showBtn, logoAS3 est ajouté sur la scène et

l'évènement **Event.ADDED\_TO\_STAGE** se déclenche.

Cela appelle la fonction activate pour envoyer un évènement **Event.ENTER\_FRAME** et donc déplacer le logoAS3 à chaque image.

Lorsque l'utilisateur appuie sur le bouton removeBtn, logoAS3 est supprimé de la display list, ce qui déclenche l'évènement **Event.REMOVED\_FROM\_STAGE**.

Puis un appel de la fonction deactivate qui supprime l'évènement **Event.ENTER\_FRAME** sur le logoAS3 et arrête le déplacement du logoAS3.

Ainsi la référence logoAS3 pourra être complètement supprimée au moment du passage du garbage collector.

En attendant l'exécution du garbage collector, l'objet logoAS3 continue d'utiliser des ressources CPU même s'il n'est plus affiché dans la display list. Sa consommation de ressources CPU est limitée du fait de la suppression complète des références à cet objet.

Avec le Player Flash 10 et AIR 1.5, si la tête de lecture rencontre une frame vide, les display object sont automatiquement supprimés.

Ce concept de suppression est également très important avec du contenu chargé dynamiquement via la classe Loader.

Une méthode essentielle appelée **unloadAndStop()** permet de décharger un fichier SWF et de supprimer toutes les références à l'intérieur de ce fichier SWF, tout en obligeant le passage du garbage collector.

Voici un exemple d'utilisation de la méthode **unloadAndStop()** :

```
var loader:Loader = new Loader();
loader.load ( new URLRequest ( "site.swf" ) );
addChild ( loader );
stage.addEventListener ( MouseEvent.CLICK, unloadSWF );
function unloadSWF ( e:MouseEvent ):void
{
    // Décharge le fichier the SWF avec une désactivation automatique des objets
    loader.unloadAndStop();
}
```

Les actions suivantes sont effectuées suite à l'appel de la méthode **unloadAndStop()** :

- Les sons sont stoppés.
- Les écouteurs enregistrés dans le fichier SWF sont supprimés.
- Les objets Timer sont stoppés.
- Les périphériques (comme la caméra et le micro) sont enlevés.
- Tous les MovieClip sont stoppés.
- L'envoi des événements **Event.ENTER\_FRAME**, **Event.FRAME\_CONSTRUCTED**, **Event.EXIT\_FRAME**, **Event.ACTIVATE** and **Event.DEACTIVATE** sont stoppés.

## Les évènements Activate et deactivate

*Utilisez les évènements `Event.ACTIVATE` et `Event.DEACTIVATE` pour détecter l'inactivité d'une application et l'optimiser.*

Deux évènements **Event.ACTIVATE** et **Event.DEACTIVATE** peuvent vous aider dans l'optimisation d'une application et ainsi utiliser le moins de ressources CPU possible.

Ces évènements vous permettent de détecter lorsque l'application perd le focus.

En les écoutant, il est possible d'optimiser votre application en l'adaptant (pour les calculs, les animations...).

Le code suivant écoute ces évènements et change le framerate dynamiquement à 0 lorsque l'application perd le focus.



Une application perd le focus lorsque l'utilisateur change d'onglet ou met l'application en tâche de fond (en mode réduit).

```
var originalFrameRate:uint = stage.frameRate;
var standbyFrameRate:uint = 0;
stage.addEventListener ( Event.ACTIVATE, onActivate );
stage.addEventListener ( Event.ACTIVATE, onDeactivate );
function onActivate ( e:Event ):void
{
    // met le framerate d'origine
    stage.frameRate = originalFrameRate;
}
function onDeactivate ( e:Event ):void
{
    // met framerate à 0
    stage.frameRate = standbyFrameRate;
}
```

Lorsque l'application reprend le focus, le framerate est remis à sa valeur d'origine.

En plus de changer le framerate dynamiquement, vous pouvez tout à fait effectuer d'autres optimisations comme : la suppression d'objets, l'arrêt d'animations...

## LES PERFORMANCES ACTIONSCRIPT 3.0

### L'API de dessin

*Utilisez l'API de dessin pour une exécution rapide du code.*

Le Player Fkash 10 et AIR 1.5 fournissent **une nouvelle API de dessin, qui permet une meilleure performance lors de l'exécution du code.**

Cette nouvelle API ne donnent pas d'amélioration de performances de rendu mais elle réduit énormément le nombre de lignes de code à écrire.

Quelques lignes pour donner une meilleure performance lors de l'exécution ActionScript.

La nouvelle API de dessin propose ces méthodes :

- **drawPath()**
- **drawGraphicsData()**
- **drawTriangles()**

Voici un exemple de code avant cette nouvelle API de dessin :

```
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 );
```

Et un autre exemple avec l'utilisation de la nouvelle API de dessin :

```
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()** fournit des améliorations de performances identiques.

## La capture et diffusion d'évènements

Le système évènementiel de l'ActionScript 3.0 propose un concept de capture et diffusion d'évènements.

En utilisant cet avantage de diffusion d'évènements, cela peut vous aider à optimiser l'exécution de votre code ActionScript.

Vous pouvez enregistrer votre évènement sur un objet, au lieu de plusieurs, pour améliorer les performances.

Dans l'exemple suivant, prenez l'exemple d'un jeu, où l'utilisateur doit cliquer le plus vite possible sur des logos pour les détruire.

Le jeu supprime chaque logo de l'écran lorsqu'il est cliqué et ajoute des points à l'utilisateur.

Ecoutez l'évènement **MouseEvent.CLICK** diffusé par chaque logo, avec le code suivant :

```
const MAX_NUM:int = 10;
var sceneWidth:int = stage.stageWidth;
var sceneHeight:int = stage.stageHeight;
var currentLogo:InteractiveObject;
var currentLogoClicked:InteractiveObject;
for ( var i:int = 0; i< MAX_NUM; i++ )
{
    currentLogo = new LogoAS3();
    currentLogo.x = Math.random()*sceneWidth;
    currentLogo.y = Math.random()*sceneHeight;
    addChild ( currentLogo );
    // Ecoute l'évènement MouseEvent.CLICK sur chaque logo
    currentLogo.addEventListener ( MouseEvent.CLICK, onLogoClick );
}
function onLogoClick ( e:MouseEvent ):void
{
    currentLogoClicked = e.currentTarget as InteractiveObject;
    currentLogoClicked.removeEventListener(MouseEvent.CLICK, onLogoClick );
    removeChild ( currentLogoClicked );
}
```

Ce code appelle la méthode **addEventListener()** sur chaque instance de logo. Et supprime chaque écouteur lorsqu'un logo est cliqué, avec la méthode **removeEventListener()**.

L'ActionScript 3.0 fournit une solution de capture et diffusion d'évènements qui permet d'écouter les évènements directement à partir de l'objet parent.

Du coup, il est possible d'optimiser le code en diminuant le nombre d'appel aux méthodes **addEventListener()** et **removeEventListener()**.

L'exemple de code suivant montre l'ajout de l'évènement directement à l'objet parent :

```
const MAX_NUM:int = 10;
var sceneWidth:int = stage.stageWidth;
var sceneHeight:int = stage.stageHeight;
var currentLogo:InteractiveObject;
var currentLogoClicked:InteractiveObject;
var container:Sprite = new Sprite();
addChild ( container );
// Ecoute l'évènement MouseEvent.CLICK sur le parent (container des logos)
// Mise à true du 3ème paramètre de l'évènement pour la phase de capture
container.addEventListener ( MouseEvent.CLICK, onLogoClick, true );
for ( var i:int = 0; i< MAX_NUM; i++ )
{
    currentLogo = new Apple();
    currentLogo.x = Math.random()*sceneWidth;
    currentLogo.y = Math.random()*sceneHeight;
    container.addChild ( currentLogo );
}
function onLogoClick ( e:MouseEvent ):void
{
    currentLogoClicked = e.target as InteractiveObject;
    container.removeChild ( currentLogoClicked );
}
```

Ce code est ainsi simplifié et optimisé avec un seul appel de la méthode **addEventListener()**.

Les écouteurs ne sont plus enregistrés sur les instances de logo, du coup il n'y a plus besoin de supprimer l'évènement lorsque le logo est cliqué.

La fonction qui reçoit l'évènement peut encore être optimisé en supprimant la propagation de l'évènement.

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

## Travailler avec les Pixels

Utilisez la méthode `setVector()` pour dessiner des pixels.

Lorsque vous ajoutez des pixels, **une optimisation simple est d'utiliser la méthode adéquate de la classe `BitmapData`.**

Une solution rapide pour dessiner des pixels est d'utiliser la méthode `setVector()`.

```
var width:int = 200;// dimensions de l'image
var height:int = 200;
var total:int = width*height;
// Pixel colors Vector
var pixels:Vector.<uint> = new Vector.<uint>(total, true);
for ( var i:int = 0; i< total; i++ )
{
    pixels[i] = Math.random()*0xFFFFFFFF;// sauvegarde la couleur de chaque pixel
}
// création d'un objet BitmapData non-transparent
var myImage:BitmapData = new BitmapData ( width, height, false );
var imageContainer:Bitmap = new Bitmap ( myImage );
myImage.setVector ( myImage.rect, pixels );// dessine les pixels
addChild ( imageContainer );
```

Lorsque vous utilisez des méthodes lentes comme `setPixel()` ou `setPixel32()`, utilisez les méthodes `lock()` et `unlock()` pour une exécution plus rapide.

Dans le code suivant, en utilisant les méthodes **lock()** et **unlock()**, cela améliore les performances :

```
var buffer:BitmapData = new BitmapData(200,200,true,0xFFFFFFFF);
var bitmapContainer:Bitmap = new Bitmap(buffer);
var positionX:int;
var positionY:int;
buffer.lock();// mise à jour en lock
var starting:Number=getTimer();
for (var i:int = 0; i<2000000; i++)
{
    positionX = Math.random()*200;// positions aléatoires
    positionY = Math.random()*200;
    buffer.setPixel32( positionX, positionY, 0x66990000 );// 40% de pixels transparents
}
buffer.unlock();// mise à jour unlock
addChild( bitmapContainer );
trace( getTimer () - starting );
// output : 670
```

La méthode **lock()** de la classe BitmapData "bloque" une image contre les modifications. Les objets qui la référence, ne peuvent pas la mettre à jour lorsque l'objet BitmapData est mis à jour.

Par exemple, si un objet Bitmap référence un objet BitmapData, vous pouvez "bloquer" (lock) l'objet BitmapData, le modifier et le débloquent (unlock).

L'objet Bitmap n'est pas mis à jour tant que l'objet BitmapData n'est pas débloquent.



Pour améliorer les performances, utilisez toujours cette méthode avec **unlock()** avec et après les appels aux méthodes **setPixel()** ou **setPixel32()**.

L'appel de **lock()** et **unclock()** empêche l'écran de se mettre à jour inutilement.

Les méthodes qui parcourent les pixels comme **getPixel()**, **getPixel32()**, **setPixel()** et **setPixel32()** sont très lentes, surtout sur les appareils mobiles.

Dans la mesure du possible, utilisez ces méthodes pour récupérer tous les pixels avec un seul appel.

Pour lire les pixels, utilisez la méthode **getVector()**, qui est plus rapide que la méthode **getPixel()**.

Aussi, rappelez-vous d'utiliser cette API avec des vecteurs lorsque c'est possible, pour une exécution plus rapide.

## OPTIMISATIONS PLUS FINES

### TextField

*Pour un objet TextField, utilisez toujours la méthode `appendText()`, à la place de l'opérateur `+=`.*

Lorsque vous travaillez avec la propriété **text** de l'objet **TextField**, utilisez toujours la méthode **appendText()**, à la place de l'opérateur **+=**.

L'utilisation de la méthode **appendText()** donne une amélioration des performances.

Dans l'exemple ci-dessous, en utilisant l'opérateur **+=**, l'opération prend 1120ms :

```
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, l'opérateur **+=** est remplacé par **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 en 847 ms.

## Les boucles avec TextField

*Lorsque c'est possible, mettez à jour votre texte en dehors des boucles.*

Le code précédent peut encore être optimisé en utilisant une technique simple.

En mettant à jour le texte à chaque boucle, cela prend beaucoup de calculs interne.

Simplement, en concaténant une chaîne de caractères, et à l'affecter au TextField, en dehors de la boucle, le code s'exécute vraiment beaucoup plus rapidement.

Le code ci-dessous s'exécute en seulement 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
```

Procédez de la même façon avec un texte HTML.

Utilisez une chaîne de caractères concaténée dans votre boucle. Puis en dehors de la boucle, affectez directement votre chaîne de caractères à votre texte HTML.

**L'exécution du code sera extrêmement plus rapide.**

Le code ci-dessous s'exécute en seulement 29 ms:

```
var myTextField:TextField = new TextField();  
addChild ( myTextField );  
myTextField.autoSize = TextFieldAutoSize.LEFT;  
var started:Number = getTimer();  
var content:String = myTextField.htmlText;  
for (var i:int = 0; i< 1500; i++ )  
{  
    content += "<b>ActionScript<b> 3";  
}  
myTextField.htmlText = content;  
trace ( getTimer() - started );  
// output : 29
```

**Note** : dans le Player Flash 10.1 et AIR 2.5, la classe **String** a été améliorée, du coup, elle utilise encore moins de mémoire.

## Remplacer l'opérateur []

*Évitez d'appeler des fonctions avec l'opérateur [].*

**L'utilisation de l'opérateur [] peut réduire les performances de votre application.**

Vous pouvez éviter de l'utiliser en stockant une référence dans une variable locale.

Le code ci-dessous montre la baisse de performance avec l'opérateur [].

```
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
```

Et voici une code optimisé, en réduisant l'utilisation de l'opérateur [].

```
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
```

Vous pouvez remarquer **un gain de performance pratiquement multiplié par 2.**

## Limiter l'appel à des fonctions

*Lorsque c'est possible, réduisez l'appel à des fonctions dans votre code.*

**L'appel de fonction coûte énormément de ressources.**

Essayez de **réduire le nombre d'appel de fonctions** en modifiant votre code. C'est une excellente solution pour améliorer les performances de votre application.

Toutefois, gardez en mémoire, que la modification de ce code est difficile à réutiliser et peut augmenter la taille de votre SWF.

Certaines fonctions, comme la classe Math, sont faciles à modifier.

L'exemple ci-dessous utilise 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
```

Le calcul de **Math.abs()** peut être effectué manuellement.



```
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 se passant de la méthode **Math.abs()**, le code devient 4 fois plus rapide lors de l'exécution.

Un gain de performance non négligeable !

Il est possible d'utiliser cette technique dans plusieurs projets.

Mais faites attention, ce type de code est difficilement maintenable et réutilisable.

## Optimiser les boucles

*Optimisez vos boucles.*

Une autre optimisation peut s'effectuer directement dans les boucles, au moment du test d'incrémentation.

Le code suivant effectue une itération dans un tableau, mais ce n'est pas optimisé car la taille du tableau est évalué à chaque itération.

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

Il est **plus performant de stocker la taille du tableau et de la réutiliser.**

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

## Les boucles inverses

*L'utilisation des boucles inverses est plus optimisée que les boucles normales.*

**Un boucle while inverse est plus optimisée qu'une boucle normale.**

```
var i:int = myArray.length;  
while (--i > -1)  
{  
    // votre code  
}
```

Ces différentes astuces permettent d'optimiser les performances de votre code ActionScript.

Cela vous montre qu'une seule ligne de code peut affecter les performances et l'utilisation de la mémoire.

D'autres optimisations ActionScript sont possibles.

**Pour plus d'informations, consultez :** <http://www.actionscript-facile.com/?p=1615>

**Optimiser le code ActionScript 3**

## OPTIMISATION DE L'APPLICATION

### Du contenu externe

*Séparez votre application en plusieurs SWF.*

Les appareils mobiles ont un accès limité au réseau.

**Pour charger votre contenu plus rapidement, séparez votre application en plusieurs SWF.**

**Essayez de réutiliser le code et les éléments graphiques le plus possible** dans votre application.

Vous pouvez diviser votre SWF en suivant cet type d'organisation :

- 1 SWF pour tous vos éléments graphiques (boutons, bannières...).
- 1 SWF pour vos animations.
- 1 SWF qui contient votre librairie de code ActionScript (Framework Pixlib, PureMVC, AS3 Facile...).
- 1 SWF contenant vos polices de caractères.
- 1 SWF qui contient vos fichiers sons.
- Si votre application SWF devient imposante, comme un jeu vidéo par exemple, vous pouvez créer un SWF par niveau de jeu. Ce SWF niveau de jeu contient le code ActionScript du niveau et des graphismes spécifiques.

En utilisant ce principe de multiples SWF, **votre application charge uniquement le contenu nécessaire pour l'utilisateur**, comme le niveau de jeu en cours.

Et au fur et à mesure de la progression du joueur, de nouveaux éléments sont chargés. C'est un peu comme le chargement d'un niveau sur une console de jeux vidéos de salon.

La classe **ApplicationDomain** stocke toutes les définitions de classes qui sont chargées. La méthode **getDefinition()** permet d'utiliser ces classes.

Voici un exemple de mise en pratique de la classe **ApplicationDomain** :

```
var loader:Loader = new Loader();// Création d'un objet Loader
loader.contentLoaderInfo.addEventListener(Event.COMPLETE, loadingComplete );
// chargement du SWF contenant des graphismes
loader.load(new URLRequest("library.swf") );
var classDefinition:String = "Logo";
function loadingComplete(e:Event ):void
{
    var objectLoaderInfo:LoaderInfo = LoaderInfo ( e.target );
    // Récupère une référence à travers le fichier SWF chargé
    var appDomain:ApplicationDomain = objectLoaderInfo.applicationDomain;
    // Vérification si la classe est disponible
    if ( appDomain.hasDefinition(classDefinition) )
    {
        // Récupération de la classe
        var importLogo:Class = Class ( appDomain.getDefinition(classDefinition) );
        var instanceLogo:BitmapData = new importLogo(0,0);// Nouvelle instance logo
        addChild ( new Bitmap ( instanceLogo ) );// ajout à la display list
    }
    else trace ("The class definition " + classDefinition + " n'est pas disponible.");
}
```

Il est très simple d'instancier une classe, une image... à partir d'un fichier SWF chargé.

## Flash Remoting

*Utilisez le Flash Remoting et l'AMF pour optimiser les communications client-serveur.*

**Vous pouvez utiliser des fichier XML pour charger du contenu distant dans votre SWF.**

Le fichier XML est ainsi chargé puis analysé par l'application.

**XML est un meilleur choix pour les applications avec peu de contenu distant à charger.**

Le fichier XML est ainsi de petite taille, il se charge et s'analyse rapidement.

Par contre, **si vous développez une application avec beaucoup de contenu externe à charger, utilisez la technologie de flash remoting et Action Message Format (AMF).**

L'AMF est un format binaire utilisé pour partager des informations entre un serveur (php) et un client (flash).

L'utilisation de l'AMF réduit la taille des données et améliore donc la vitesse de transmission.

L'AMF est un format natif de Flash, **l'envoi et la réception des données AMF évite d'utiliser beaucoup de mémoire lors de la sérialisation et la désérialisation.**

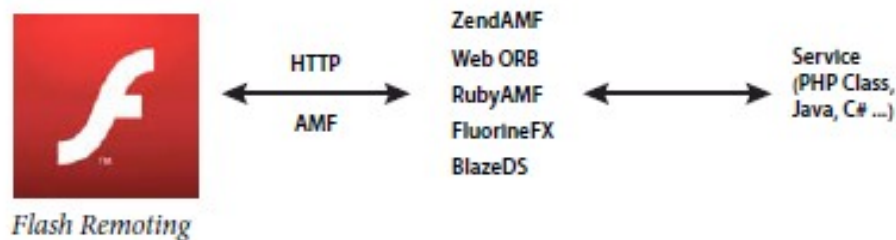
La passerelle de remoting (gateway) s'occupe de ces tâches.

Lorsque vous envoyez des données au serveur, la passerelle effectue la désérialisation pour vous sur le serveur.

La passerelle envoie également les données en conservant leur type (Number, String, Object, Array).

Pour effectuer du flash remoting, la passerelle la plus connue et gratuite est AMFPHP.

Voici une image qui illustre le principe de fonctionnement :



Un tutoriel complet sur l'échange des données entre PHP (et MySQL) et Flash est disponible sur ActionScript Facile : <http://www.actionscript-facile.com/?p=397>

**Flash remoting - échanger des données entre php mysql et flash actionscript**

Voici un exemple de code ActionScript pour la mise en place de la communication remoting :

```
var connection:NetConnection = new NetConnection (); // Création de l'objet NetConnection
// Connection de Flash à la passerelle (gateway) Remoting
connection.connect ("http://www.votreserver.com/remoting-service/gateway.php");
// Fonctions pour recevoir les données et les erreurs
function success ( incomingData:* ):void
{
    trace( incomingData );
}
function error ( error:* ):void
{
    trace( "Error occurred" );
}
// Création d'un objet pour rediriger la réponse du serveur vers les fonctions success et error
var serverResult:Responder = new Responder (success, error);
// Appel de la méthode remoting sur le serveur avec des paramètres
connection.call ("com.votreserver.HelloWorld.sayHello", serverResult, "Hello there ?");
```



## Utiliser le cache de Player Flash

*Mettez en cache vos éléments après les avoir chargés au lieu de les charger à chaque fois à travers le réseau.*

Si votre application charge des éléments comme des fichiers média ou des données, vous pouvez utiliser le cache en les sauvegardant sur l'appareil de l'utilisateur.

Pour les éléments mis à jour fréquemment, utilisez un intervalle pour mettre à jour le cache régulièrement.

Par exemple, votre application peut vérifier une fois par jour s'il existe une nouvelle version d'une image, d'un fichier XML ou mettre à jour ces données toutes les 3h00...

**Vous pouvez mettre en cache les éléments de plusieurs façon** en fonction du type et de la nature de l'élément :

- Les éléments comme les images, les vidéos, l'audio : sauvegardez-les sur le disque dur du client en utilisant les classes File et FileStream.
- Les données individuelles ou de petites tailles (fichiers XML) : sauvegardez leur valeur des objets partagés (type cookie) en utilisant la classe SharedObject.
- Les données importantes : sauvegardez ces données dans une base de données (MySQL) ou sérialisez ces données et stockez-les sur le disque dur du client.

Pour le cache des données, **le projet open source AS3CoreLib** comprend une classe ResourceCache qui s'occupe du chargement et du cache pour vous.

<https://github.com/mikechambers/as3corelib>

## **CONCLUSION**

Ce sont **des optimisations très simples** qui peuvent être appliquées à vos applications flash **pour obtenir de meilleures performances et de rapidité.**

En faisant attention à l'utilisation du processeur, **l'expérience globale de l'utilisateur sera plus fluide et agréable.**

Comme nous venons de voir, **les optimisations ne sont pas liées uniquement à ActionScript.**

La plupart des optimisations efficaces, que vous pouvez appliquer à vos applications flash, sont liées à de simples recommandations graphiques ou ds détails de code.

Vous trouverez un complément d'optimisation dans le livret de formation : **L'Optimisation des Publicités Flash.**

A télécharger gratuitement sur ActionScript Facile : <http://www.actionscript-facile.com/?p=1257>

**Optimiser les publicités flash ActionScript 3**

## C'EST MAINTENANT !

Rendez-vous sur cette page pour me dire ce que ce livre de formation vous a apporté :

<http://www.actionscript-facile.com/livret-optimiser-performances-plateforme-adobe-flash/article1220043.html>

Dites-moi dans les commentaires comment je pourrais l'améliorer.

**Profitez-en pour poster vos exemples d'applications flash optimisées !**

Je compte le compléter en fonction des évolutions du Player Flash.

Merci pour votre lecture,

Matthieu

Formateur des Développeurs ActionScript

[illegible]

Ce livret vous a été remis en complément du [Programme AS3 Formation](#).

**Il vous est personnel et ne peut pas être redistribué.**

Ce livret vous a été remis par le site [Formation Flash ActionScript 3](#)

Si vous souhaitez partager directement l'adresse du site web, où j'explique en vidéo comment créer simplement une application flash robuste et évolutive, visitez :

[Tutoriels Vidéos gratuits pour les Technologies Adobe Flash, Adobe AIR, ActionScript 3](#)

## Livret de Formation

Ecrit par Matthieu Deloison  
**Formateur ActionScript**

**Rendez-vous sur [Formation Flash ActionScript 3](#)** pour d'autres cours en vidéos.

  
MATTHIEU DELOISON