L'Optimisation des Publicités Flash

L'Optimisation des Publicités Flash

Ce livret vous a été remis par le site **ActionScript-Facile.com**.

Vous pouvez librement l'imprimer, le prêter, le distribuer ou le faire suivre à un développeur qui appréciera les conseils et stratégies de programmation actionscript présentées ici.

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 :

http://www.actionscript-facile.com/

Livret de Formation

Ecrit par Thibault Imbert : **http://www.bytearray.org/?p=1586**

Traduction par Matthieu Deloison : **http://www.actionscript-facile.com/**

L'Optimisation des Publicités Flash

Table des matières

INTRODUCTION

Projet v 1.1

Par Thibault Imbert.

L’optimisation des publicités

Avant de vous expliquer comment optimiser les bannières publicitaires, il est important de vous signaler qu’il existe différents niveaux de bannières Flash.

Voici un résumé de cette liste :

- Animation manuelle : la publicité est animée manuellement, toutes les animations sont faites sur la timeline ou la scène (sans Actionscript).
- A moitié dynamique : la publicité est composée d’animation manuelle et d’animation dynamique (en actionscript).
- Complètement dynamique : l’animation de la publicité est entièrement construite en Actionscript.

La performance pour le rendu est le facteur le plus important concernant les optimisations Flash.

Quelque soit le type de contenu que vous créez, le rendu est un des éléments qui peut être facilement amélioré et vous pouvez ainsi obtenir des améliorations de performances énormes.

C’est la raison pour laquelle ce livre se focalise sur les optimisations du rendu.

Des publicités peuvent avoir besoin de code.

Pour cette raison, nous allons découvrir, à la fin de ce livre, les erreurs typiques de codes pour toutes les versions d’Actionscript.

Parce que les publicités, créées chaque jour, utilisent différentes versions d’Actionscript. Des développeurs utilisent encore Actionscript 1, d’autres Actionscript 2 ou 3.

LES OPTIMISATIONS GRAPHIQUES

Qualité des animations et options de rendu du texte

Utiliser la qualité du stage la plus appropriée.

A chaque fois que vous développez du contenu flash, à partir des applications style “Publicités”, choisissez la qualité du Stage de l'animation, qui présente une optimisation adéquate.

En utilisant la qualité du Stage adéquate, les performances de rendu peuvent grandement être améliorée.

Ces paramètres de configuration de la qualité du Stage sont disponibles en Actionscript 3 :

- **StageQuality.LOW** : améliore la vitesse de lecture de l'apparence mais n'utilise pas l'anti-aliasing.
- **StageQuality.MEDIUM** : appliquer de l'anti-aliasing mais pas d'optimisation sur les bitmaps not smooth bitmaps.
- **StageQuality.HIGH** : (option par défaut) améliore la vitesse de lecture et utilise toujours l'anti-aliasing. Si le SWF ne contient pas d'animation, les bitmaps sont bitmaps are smoothed, si le SWF contient des animations, bitmaps are not smoothed.
- **StageQuality.BEST** : fournit la meilleure qualité et ne prend pas en compte la vitesse de lecture. Tous les éléments possèdent l'anti-aliasing et les bitmaps sont toujours lissés.

La qualité de l’application swf peut être configurée via une simple ligne de code, à partir de la propriété **quality** de l’objet **stage** :

```
stage.quality = StageQuality.MEDIUM;
```

L'équivalent en AS1/AS2 version est :

```
_quality = "medium";
```

En AS3, l'objet Stage est référencé par n'importe quel DisplayObject ajouté à la display list.

Cependant, la qualité du Stage peut être réglée à partir du panneau “paramètre de publication”.

La copie d'écran ci-dessous vous montre les options de qualité de l'animation.

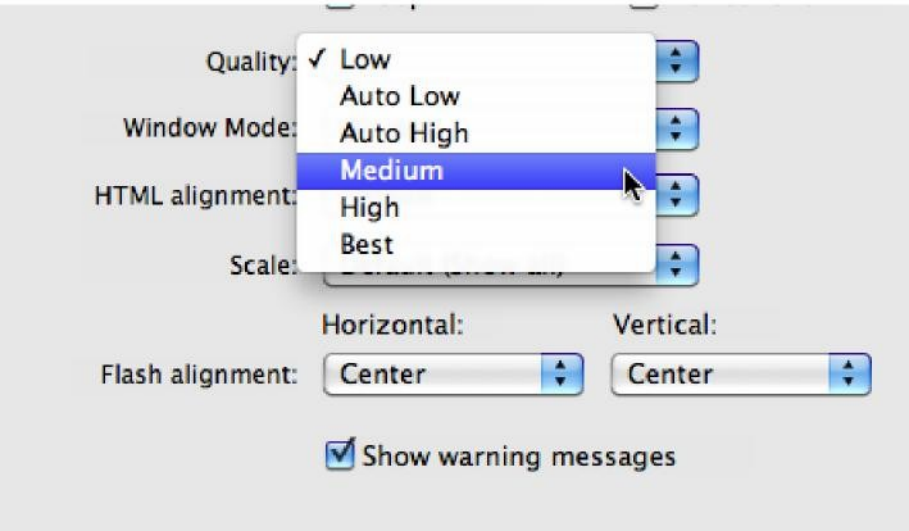

Figure 1.1

Mettez la qualité du Stage à Medium, le texte utilise l'anti-alias pour l'animation.

En utilisant la qualité Medium, cela fournit souvent une qualité suffisante si des bitmaps sont utilisées, dans quelques cas rares, la qualité Low est suffisante, mais ce n'est pas souvent.

Un truc très simple à retenir est que dans le Flash Player 8, le texte anti-aliasé peut être rendu avec précisions, même avec la qualité du Stage à Low.

Dans l'image ci-dessous, la qualité de l'animation est réglée sur Medium, et le rendu du texte anti-alias est activé pour l'animation :

Figure 1.2

La qualité du Stage est réglé à Medium, le texte utilise l'anti-alias pour l'animation.

Dans l'image ci-dessus, les paramètres de qualité du Stage affecte la qualité du texte parce que nous n'utilisons l'option de rendu de texte appropriée.

Dans le Flash Player 8, une amélioration du texte inclus une option appelée « anti-alias » pour la lisibilité. Ce réglage permet d'obtenir une qualité parfaite de votre texte anti-aliasé quelque soit la qualité du Stage et la version Actionscript utilisé.

Dans l'image ci-dessous, la qualité de l'animation est réglé sur **StageQuality.LOW** et le rendu anti-alias du texte est activé pour la lecture :

Figure 1.3
La qualité du Stage est réglée à Low, le texte utilise l'anti-alias pour la lecture.

La même qualité de rendu peut être obtenu en mettant le paramètre de rendu du texte à bitmap Text (sans anti-alias).

Dans l'image suivante, la qualité de l'animation est réglée à Low, et le rendu du texte configuré à Bitmap Text :

Figure 1.4

La qualité du Stage est réglée à Low, le texte utilise du texte bitmap.

Les 2 précédents exemples montrent clairement qu'il est possible d'obtenir un rendu de haute qualité du texte, sans utiliser le paramètre qualité du Stage.

Cette fonctionnalité est disponible depuis le Player Flash 8 et peut être utilisé dans la majorité des cas.

Rappelez-vous, depuis la version 10.1, le Player Flash peut automatiquement passer en qualité Medium sur les appareils mobiles pour augmenter les performances.

Le frame rate (le nombre d'images par seconde)

> *N'utilisez jamais un frame rate au dessus de 30 ou 40 fps. Le Player Flash 10.1 est incapable de gérer plus de 60fps.*

En général, utilisez le framerate le plus faible possible pour des meilleures performances.

Le framerate d'une application détermine combien de temps est disponible pour un cycle du rendu du code de l'application.

Un framerate élevé peut créer des animations plus fluides.

Toutefois, lorsqu'il n'y a pas d'animation ou d'effets visuels, il n'y a aucune raison d'avoir un framerate élevé.

Ci-dessous, des recommandations générales pour choisir le framerate adéquate pour votre application :

- Si votre application contient des animations, le framerate adéquate est de 20 images par secondes. Plus de 30 images par secondes n'est pas souvent nécessaire.
- Si votre application ne contient pas d'animation, un framerate de 12 images par secondes est probablement suffisant.

Le framerate le plus bas possible varie en fonction de l'activité d'affichage de votre application.

Utilisez un framerate faible lorsque la vidéo est le seul élément dynamique de votre application.

Au démarrage, le runtime lance la lecture d'une vidéo chargée, au framerate par défaut de votre application.

Si votre application ne possède pas d'animation ou de changement visuel du contenu rapide, utiliser un framerate bas ne dégrade pas l'expérience de l'utilisateur.

Lorsque l'animation est terminée, le framerate d'origine est utilisé de nouveau.

Rappelez-vous que sur les mobiles et netbooks, une nouvelle fonctionnalité est apparue avec le Flash Player 10.1, intitulée 'Pause and Resume” automatiquement lorsque le swf est considéré comme inactif (écran éteint et écran rallumé).

Sur le même principe, le but de cette fonctionnalité est de limité l'utilisation du CPU. Il y a des consignes générales à suivre pour déterminer le nombre d'images par secondes adéquate en fonction du type d'activité :

- Lorsqu'une animation est jouée,utilisez un minimum de 20 images par secondes. Plus de 30 images par secondes est souvent inutile.
- Lorsqu'aucune animation n'est jouée, un framerate de 12 images par secondes est probablement suffisant.
- Les vidéos chargées sont jouées au framerate natif de l'application. Si la vidéo est le seul contenu animé de votre application, un framerate de 12 images par secondes est probablement suffisant.
- Quand l'application n'a pas la focus (la main), un framerate de 5 images par secondes est suffisant.

En supplément de cela, le Flash Player 10.1 implémente ces nouvelles options :

- Le framerate maximum est de 60fps pour les systèmes de bureau. Il n'y a pas de maximum pour les mobiles (60 fps est trop élevé), mais nous respectons une consigne d'environ 30fps.
- Tous les players de bureau sont concernés.
- L'application AIR n'a pas de consigne.
- **EnterFrame** peut fonctionner jusqu'à 60fps.
- Les appels **LocalConnection** prennent moins de 33ms pour s'exécuter.
- Les événements **Timer** sont limités à 60 appels par secondes.

Publier un swf avec moins de 60fps est correct car le Player Flash peut gérer un maximum de 60fps.

Rappelez-vous que vous pouvez modifier dynamiquement le framerate, pour le réduire lorsqu'il n'y a pas d'animation.

Vous pouvez définir le framerate par défaut de l'application dans les paramètres de compilation ou le projet, mais le framerate n'est pas bloqué à cette valeur.

Vous pouvez changer cette valeur avec le paramètre **stage.frameRate**.

Cette fonctionnalité n'est seulement disponible qu'avace l'AS3.

Donc, la publication de votre SWF avec une cadence supérieure à 60 est inutile, vu que le plafonnement du Flash Player est à un maximum de 60 images par seconde.

Mettez à jour le framerate en fonction des besoins de votre application.

Diminuez le framerate quand votre application n'exécute aucune animation.

Le code ci-dessous vous montre un exemple de mise en pratique, à la fin de la tween, le framerate passe à 0 :

```
import aze.motion.eaze;

import aze.motion.easing.Cubic;

var standbyFrameRate:uint = 0;

// final x position var destX:uint = 200;

// création d'une animation synamique avec le framework Eaze - url :
http://code.google.com/p/eaze-tween/ eaze(myMovieClip).to(2, { x:destX })

.onComplete(handler)

.easing(Cubic.easeInOut);

function handler ():void

{

// une fois l'animation terminée

// mise à jour dynamique du framerate

stage.frameRate = standbyFrameRate;

}
```

Lorsque l'animation démarre, augmentez le framerate.

De la même façon, si votre publicité est exécutée en tâche de fond (après la perte du focus), vous pouvez diminuer le framerate.

L'utilisateur est susceptible d'être porté sur une autre application ou une tâche.

Dans le code ci-dessous, nous écoutons les évènements **Event.ACTIVATE** et **Event.DEACTIVATE**, et nous mettons à jour dynamiquement le framerate à 0 quand l'animation perd le focus (passage à un autre onglet) :

```
var originalFrameRate:uint = stage.frameRate; var standbyFrameRate:uint = 0;

stage.addEventListener ( Event.ACTIVATE, onActivate ); stage.addEventListener
( Event.DEACTIVATE, onDeactivate );

function onActivate ( e:Event ):void

{

// restauration du framerate par défaut de l'application

stage.frameRate = originalFrameRate;

}

function onDeactivate ( e:Event ):void

{

// mise à 0 du framerate

stage.frameRate = standbyFrameRate;

}
```

Note : ces événements sont disponible uniquement en ActionScript 3.

Mais attendez, il y a mieux, en optimisant les chapes correctement, vous pouvez gagnez une amélioration significative des performances, découvrons cela tout de suite.

Dans l'image ci-dessous, la qualité de l'animation est réglée sur Medium, et le rendu du texte anti-alias est activé pour l'animation :

Here is some sample text

Figure 1.2

La qualité du Stage est réglé à Medium, le texte utilise l'anti-alias pour l'animation.

Dans l'image ci-dessus, les paramètres de qualité du Stage affecte la qualtié du texte parce que nous n'utilisons l'option de rendu de texte appropriée.

Dans le Flash Player 8, une amélioration du texte inclus une option appelée « anti-alias » pour la lisibilité.

Ce paramètre permet d'obtenir une qualité parfaite de votre texte anti-alias quelque soit la qualité du Stage et la version Actionscript utilisé.

Dans l'image ci-dessous, la qualité de l'animation est réglé sur **StageQuality.LOW** et le rendu anti-alias du texte est activé pour la lecture :

Figure 1.3

La qualité du Stage est réglée à Low, le texte utilise l'anti-alias pour la lecture.

La même qualité de rendu peut être obtenu en mettant le paramètre de rendu du texte à bitmap Text (sans anti-alias).

Dans l'image suivante, la qualité de l'animation est réglée à Low, et le rendu du texte configuré à Bitmap Text :

Figure 1.4

La qualité du Stage est réglée à Low, le texte utilise du texte bitmap.

Les 2 précédents exemples montrent clairement qu'il est possible d'obtenir un rendu de haute qualité du texte, sans utiliser le paramètre qualité du Stage.

Cette fonctionnalité est disponible depuis le Flash Player 8 et peut être utilisé dans la majorité des cas.

Rappelez-vous, depuis la version 10.1, le Flash Player peut automatiquement passer en qualité Medium sur les appareils mobiles pour augmenter les performances.

Optimisation des images vectorielles

Toujours simplifier les vecteurs en enlevant les points de contrôles inutiles.

Différemment des bitmap, rappelez-vous toujours que les images vectorielles peuvent avoir besoin de beaucoup de calculs pour le rendu, surtout lors de l'utilisation de dégradé et des chemins complexes avec beaucoup de points de contrôles.

Que vous soyez designer ou développeur, assurez-vous toujours que les formes soient optimisées.

L'image ci-dessous, vous montre des formes non simplifié, avec beaucoup de points de contrôles :

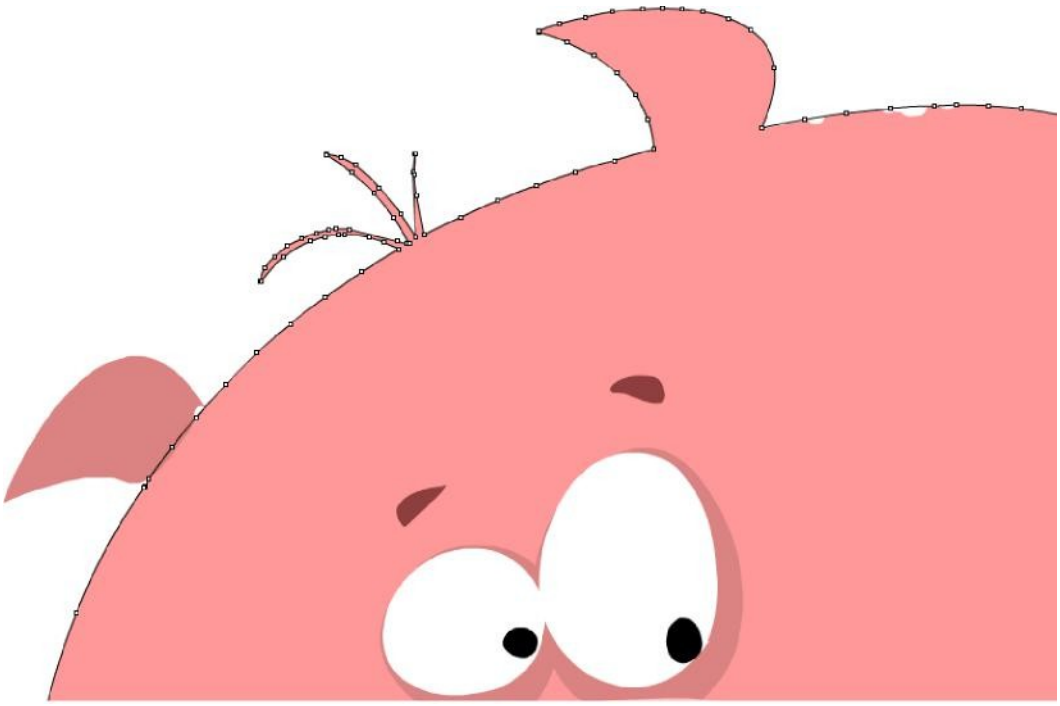

Figure 1.5

Image vectorielles non optimisée.

En utilisant l'outil Lissage, vous pouvez enlever les points de contrôles inutiles très facilement.

Ci-dessous, une copie d'écran d'Adobe Flash Pro :

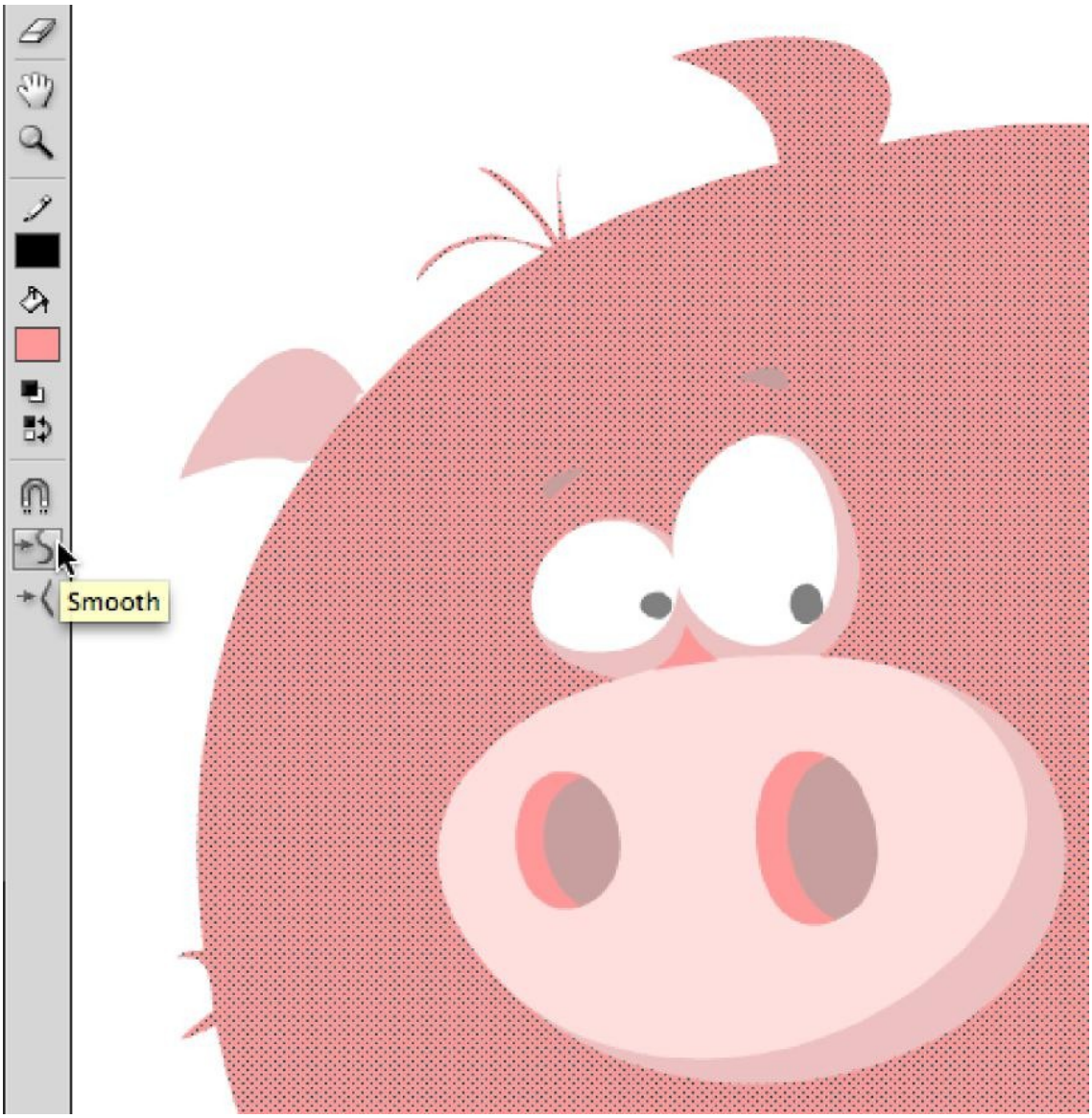

Figure 1.6

Outil Lissage dans Adobe Flash.

Rappelez-vous que cet outil est cumulatif, en simplifiant les chemins, le résultat final de l'image peut être altéré.

Quand il est utilisé intelligemment, il y a peu de chance qu'un œil humain perçoive de différence.

Un outil équivalent s'appelant Simplify, est disponible dans Adobe Illustrator, le nombre total de points et les chemins peuvent être lu à partir du panneau d'information du documents.

Ci-dessous, une copie d'écran qui illustre ce panneau et l'outil :

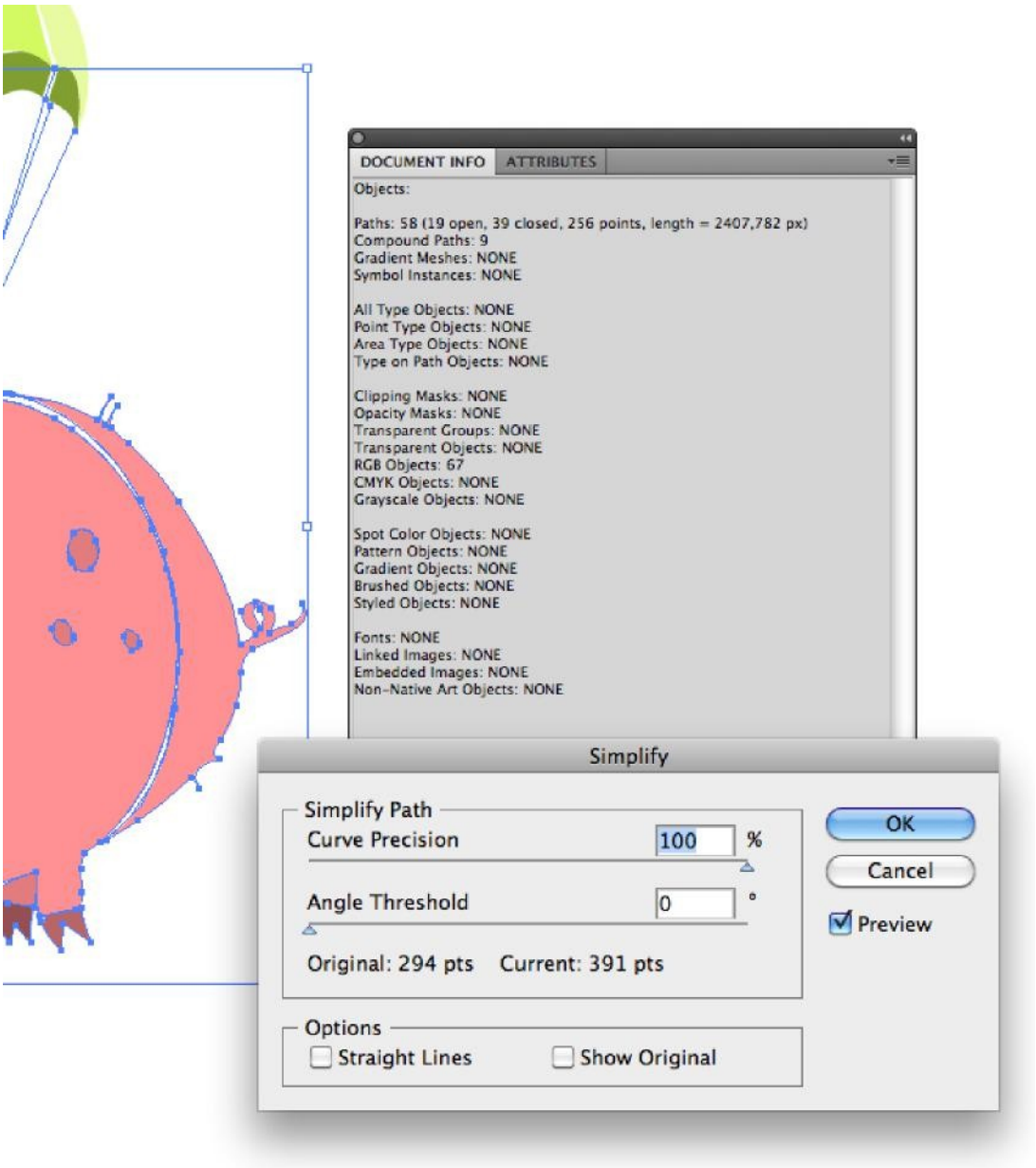

Figure 1.7

Outil Simplify dans Adobe Illustrator.

Une fois optimisé, les points de contrôles inutiles sont supprimés, cela réduit la taille du swf créé, et peut agmenté significativement les performances du rendu.

L'image suivante montre les chemins une fois optimisés :

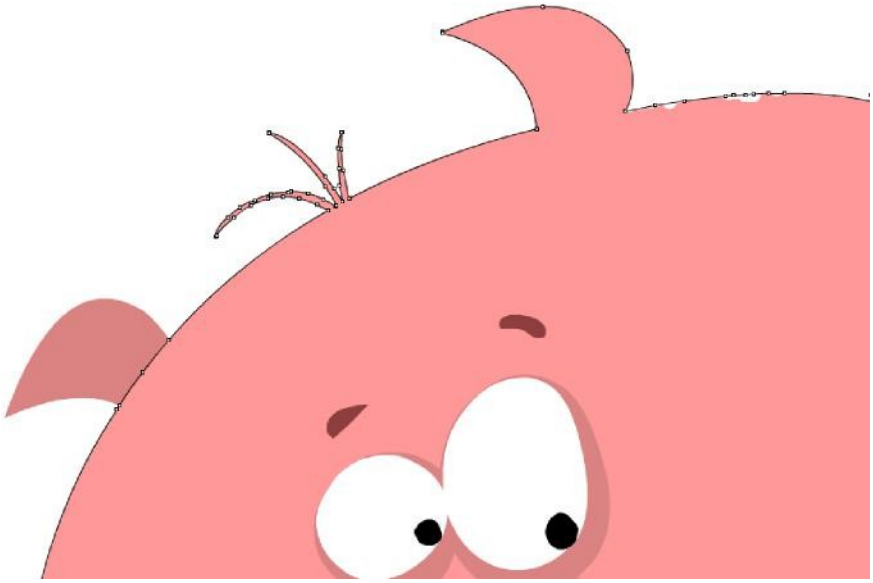

Figure 1.8

Vecteurs Optimisé.

Remarquez le nombre très limité de points de contrôle maintenant.

Le résultat est que l'image finale est plus légère avec un rendu plus rapide.

L'image suivante montre la différence de taille entre les 2 (non optimisé et optimisé) version du même graphique :

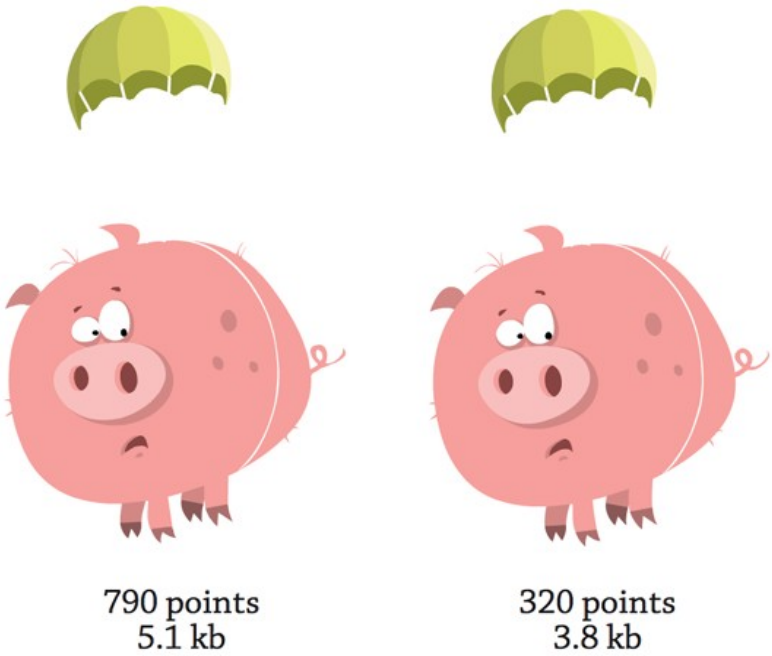

Figure 1.9
Différences un graphisme non optimisé et optimisé.

L'outil Simplify est aussi disponible dans Flash Pro, avec le menu Shape :

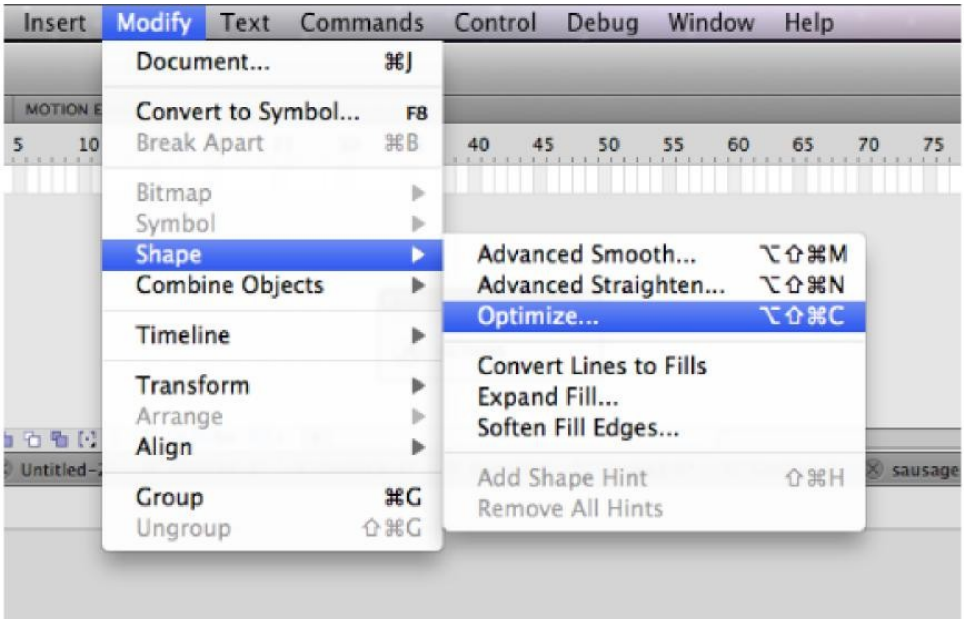

Figure 1.10

Outil Optimize dans Adobe Flash Pro.

Dans Adobe Illustrator, les courbes peuvent être simplifiées et optimisées avec le panneau optimisation comme illustré ci-dessous :

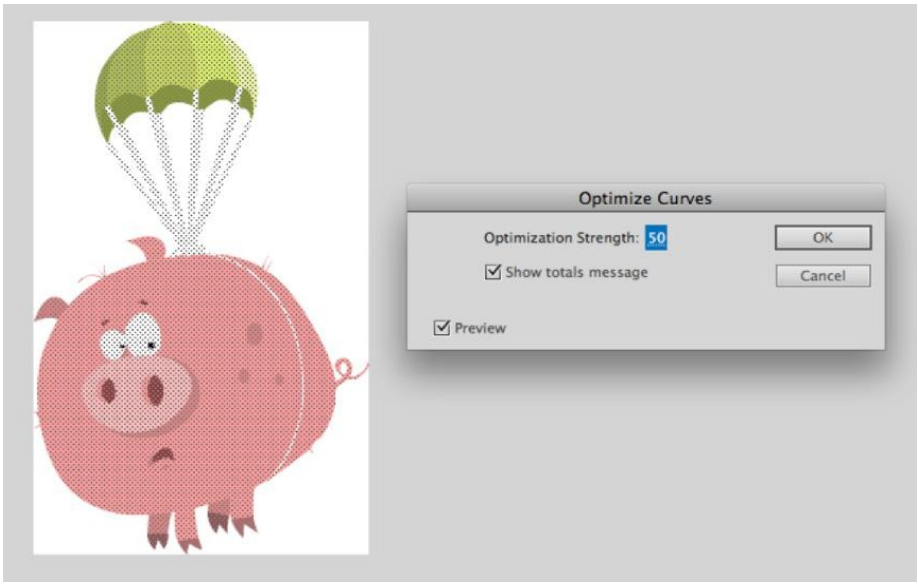

Figure 1.11

Optimisation des courbes avec la prévisualisation activée.

Avec la case à cocher prévisualisation, il est possible de voir le résultat final, une fois optimisé.

Gardez à l'esprit que ces optimisations peuvent être réalisées dans Adobe Illustrator ou Adobe Flash, en fonction de cadre de travail.

Bitmaps contre les Vecteurs

> *Dans certains cas, les bitmaps peuvent être plus petite que les vecteurs avec un rendu beaucoup plus rapide.*

Généralement, les éléments devenant imposant en taille sont réalisés vectoriellement plutôt qu'avec des bitmaps.

C'est un des nombreux avantages des vecteurs. C'est généralement vraie pour le contenu vectoriel simple, mais souvent beaucoup de points de contrôle sont utilisés, un version bitmap du contenu peut être plus léger.

L'image ci-dessous illustre un tracé bitmap, pour convertir une image bitmap en vecteur, les paramètres utilisés pour le rendu sont des couleurs simplifiées :

Figure 1.12

Image Bitmap.

Chaque couleur est un vecteur différent, avec un impact très important sur la taille du fichier mais plus important sur le rendu des performances.

La même image en bitmap est plus petite avec un rendu beaucoup plus rapide dans le Player Flash.

Rappelez-vous toujours que le rendu des bitmaps est plus rapide que les images vectorielles.

La raison est simple : pour les bitmaps, il n'y a pas de calculs de rendu, seulement des pixels qui sont décompressés et dessiné sur l'écran.

Bitmap mise en cache

Utilisez le cache des bitmaps quand le texte se déplace ou pour le contenu vectoriel complexe à l'écran.

Une autre optimisation importante peut être effectuée en utilisant la fonctionnalité de cache des bitmap, apparue avec le Flash Player 8.

Cette fonctionnalité met en cache un objet vectoriel, le rendu est fait à partir d'un bitmap interne.

Le résultat est un gain de performance pour le rendu, mais il est nécessaire d'utiliser beaucoup de mémoire.

Rappelez-vous d'utiliser toujours la fonctionnalité de cache des bitmaps pour les images vectorielles complexes, comme des dégradés ou du texte, seulement quand c'est nécessaire. Nous reviendrons sur cela plus tard dans ce document.

Dans Adobe Flash, cette fonctionnalité s'appelle « Cache as bitmap » et est activable via une case à cocher :

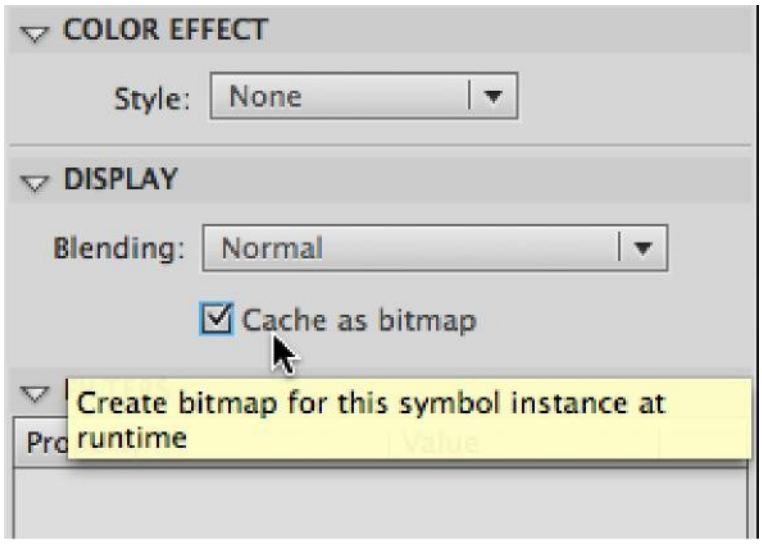

Figure 1.13

Fonctionnalité de cache d'un bitmap.

Note : cette case à cocher n'apparait pas avec un **TextField**. En utilisant un **TextField** dans un **MovieClip**, il est possible d'obtenir le même résultat de Cache as bitmap avec le container **MovieClip**.

Rappelez-vous que cette fonctionnalité peut être activée ou désactivée dans toutes les versions ActionScript, avec le paramètre **cacheAsBitmap** sur un **TextField**, ou un **Button** ou un **MovieClip**.

```
myTextField.cacheAsBitmap = true; myButton.cacheAsBitmap = true;
myMovieClip.cacheAsBitmap = true;

En ActionScript 3, tous les DisplayObject possècent cette propriété :

myDisplayObject.cacheAsBitmap = true;
```

Pour ce résultat, vous utilisez un peu plus de mémoire mais avec des performances de rendu améliorées et qui demande peu de ressources CPU, et qui consomme moins les batteries des mobiles et ordinateurs portables.

Évidemment, il n'y pas d'intérêt à activer cette option de cache sur un bitmap ou un contenant **MovieClip** un bitmap.

Le cache de bitmap doit être activé sur le contenu vectoriel uniquement.

Attention, avec les filtres dynamiques sur les bitmap, cette fonctionnalité doit être utilisée avec précaution, sinon vous obtiendrez l'effet inverse.

La mauvaise utilisation de cacheAsBitmap

N'utilisez jamais le cache as bitmap sur du contenu qui doit être translaté en x et y.

Rappelez-vous, activer le cache as bitmap sur un objet animé (translaté en x et y) qui contient des graphismes vectorielles complexes (comme du texte ou des dégradés) améliore les performances.

Modifier les propriétés x et y d'un objet n'implique une recréation de celui-ci.

Attention, toute rotation, scale ou modification de l'alpha provoque, dans le Player Flash, une recréation de la version caché du bitmap, et par conséquent, une diminution des performances.

Suivant le même principe, si le cache as bitmap est activé sur un Display Object comme un movieclip, qui contient ses propres éléments imbriqués, sur chaque frame, le Player Flash met à jour le cache des bitmap et les recalcule sur l'écran, ce qui demande plus de ressources CPU et peut diminuer la vitesse du rendu.

Note : La fonctionnalité de cache d'un bitmap est intéressante seulement quand le cache est généré une seule fois, sans avoir besoin de le mettre à jour.

Un bitmap en cache peut utiliser plus de mémoire qu'une instance d'un movieclip.

Par exemple, si le movieclip sur le Stage est de 250 x 250 pixels, quand il est en cache, il utilise 240 KB (250*250*4/1000), au lieu d'1 KB non caché.

Une autre solution pour améliorer le rendu et utiliser moins de CPU est de supprimer la transparence des **MovieClip** en utilisant la propriété **opaqueBackground**, avec laquelle il est possible d'ajouter une couleur opaque de fond au **MovieClip**.

En enlevant la transparence, le contenu du Player Flash est affiché plus vite en consommant moins de ressources CPU et en augmentant le framerate global.

Note : si l'a propriété **opaqueBackground** d'un Display Object est mise à une couleur spécifique, le Player Flash considère le Display Object comme opaque.

Lorque la propriété **cacheAsBitmap** est utilisé, le Player Flash crée un bitmap de 32 bits non transparent dans la mémoire.

Le canal alpha est mis à 0xFF, ce qui améliore les performances, parce qu'il n'y a pas besoin de transparence pour dessiner la bitmap à l'écran.

Supprimer l'alpha donne un rendu plus rapide.

Si la profondeur de l'écran courant est limité à 16 bits, alors une bitmap est stocké en mémoire sur 16 bits.

Utiliser la propriété **opaqueBackground**, n'implique pas forcément d'activer le cache as bitmap.

Pour sauvegarder en mémoire, utilisez la propriété **cacheAsBitmap** et activez là sur chaque Display Object, et également pour le container.

L'activation du cache as bitmap sur le container crée un bitmap plus lourd en mémoire, en créant un bitmap transparent avec les dimensions de 211 x 279 pixels.

Cette image utilise environ 229 KB en mémoire :

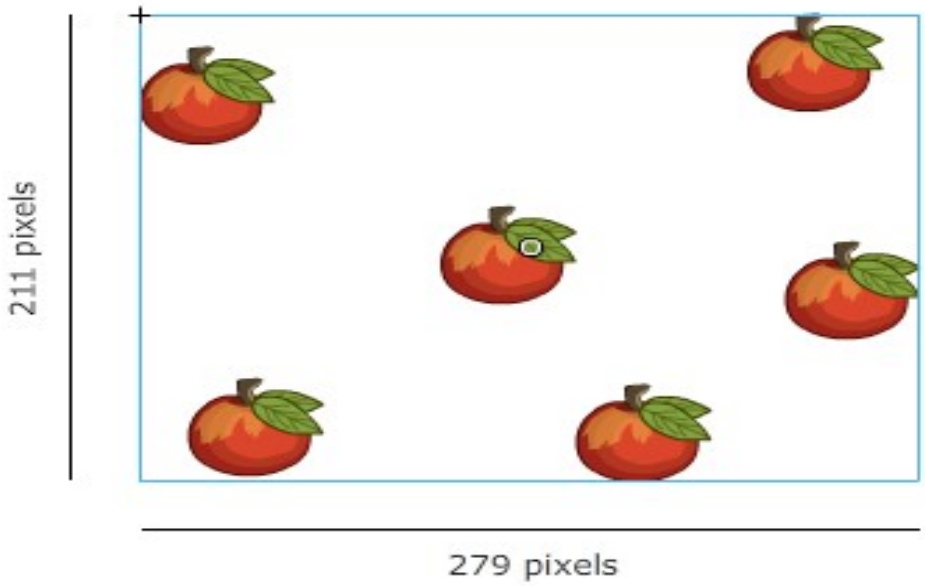

Figure 1.14

Mauvaise utilisation, activer le cache as bitmap sur le container.

En plus, en activant le cache du container, vous avez le risque de la mise à jour des bitmaps (à l'intérieur) en mémoire.

Si une pomme commence à se déplacer sur une frame, activer le cache des bitmaps sur chaque instance permet d'obtenir un cache de 7 KB en mémoire, qui utilise au total seulement 42 KB en mémoire :

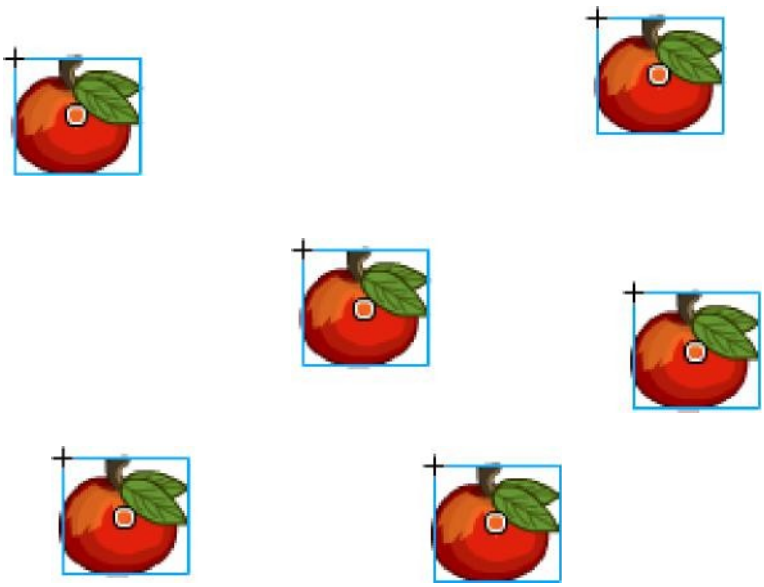

Figure 1.15

Activer le cache des bitmap sur chaque instance.

Il y a d'autre cas ou l'utilisation du cache bitmap peut dégradé les performances.

Découvrons cela tout de suite.

Gardez toujours à l'esprit que le cache bitmap peut augmenter les performances si le contenu en cache ne subit pas de rotation ou ne se déplace pas à chaque frame.

Attention, pour chaque transformation, comme une translation sur les axes x et y, le rendu n'est pas amélioré.

Dans ces cas, le Player Flash met à jour le cache bitmap pour chaque transformation sur un Display Object.

Mettre à jour le cache bitmap nécessite beaucoup de ressource CPU, diminue les performances et augmente l'utilisation de la batterie.

L'image ci-dessous illustre une situation où le cache bitmap est utilisé efficacement :

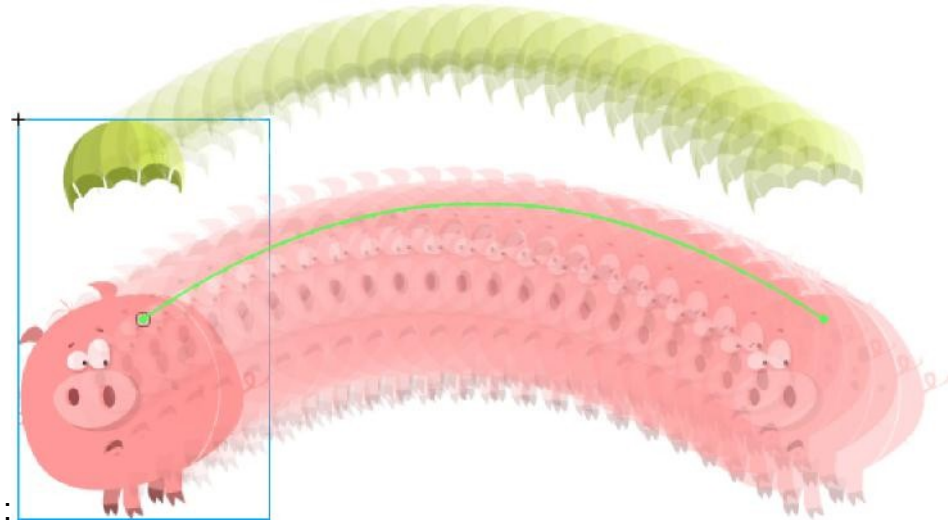

Figure 1.16

Le MovieClip se déplace sur les axes x et y.

Le **MovieClip** se déplace sur les axes x et y sans application d'autre transformation.

L'image suivante illustre l'utilisation du cache bitmap, ce qui force le Player Flash à redessiner l'objet graphique parce que l'objet est modifié tout le temps :

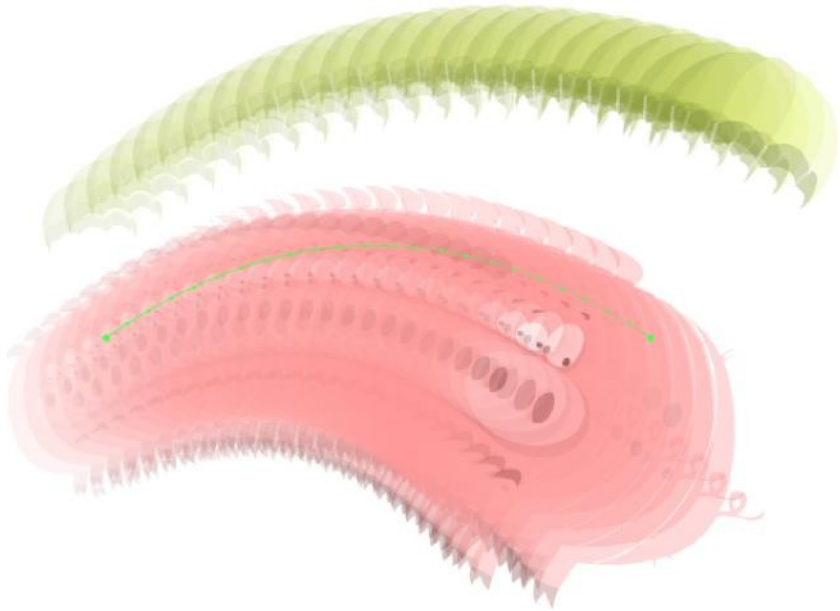

Figure 1.17
MovieClip se déplace sur x et y pendant qu'il est scalé.

La mise en cache bitmap est également utilisé, en fond de tâche, lors de l'utilisation des filtres disponibles dynamiquement ou à partir de l'IDE d'Adobe Flash Pro.

Nous allons découvrir, dans ce guide, certaines des meilleures pratiques concernant les filtres.

Utiliser les filtres correctement

Essayez de limiter l'utilisation des filtres; ou utilisez les en basse qualité ou mieux, utilisez des vidéos.

Les filtres sont accessibles dans le panneau filtre d'Adobe Flash comme l'image ci-dessous :

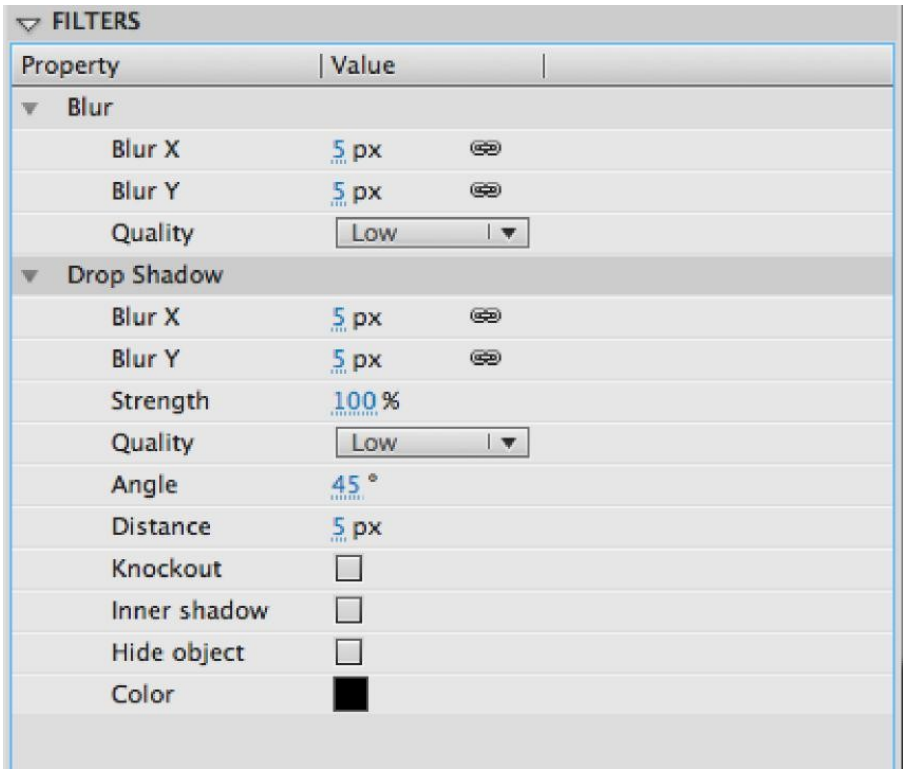

Figure 1.18
Panneau des filtres dans Adobe Flash.

Nous pensons que les filtres dans Flash sont identiques aux filtres de Photoshop, qu'ils n'utilisent pas beaucoup de ressources CPU.

Pour démontrer les conséquences, regardons l'image suivante :

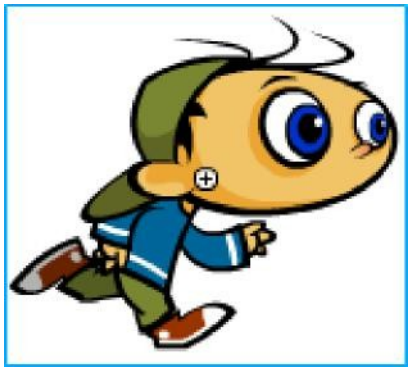

Figure 1.19
MovieClip animé sans filtre appliqué.

Un **MovieClip** animé est placé sur le stage, on peut être tenté d'ajouter un filtre d'ombre portée, comme la figure suivante le montre :

Figure 1.20
MovieClip animé avec des filtres.

Les filtres dynamiques dans Flash, s'appuient sur des bitmaps, utilisent automatiquement la propriété **cacheAsBitmap** activée.

Lorsqu'un filtre est appliqué sur un **MovieClip** animé, 2 bitmaps sont créées et sont mises à jour à chaque que le **MovieClip** se déplace, ce qui nécessite plus de ressources CPU, avec une diminution des performances, et une utilisation élevée de la batterie.

En scalant un **MovieClip** souvent, nous ajoutons un calcul supplémentaire, ce qui nous donne de mauvaises performances et une utilisation élevée du CPU.

Regardons tout cela à travers un exemple :

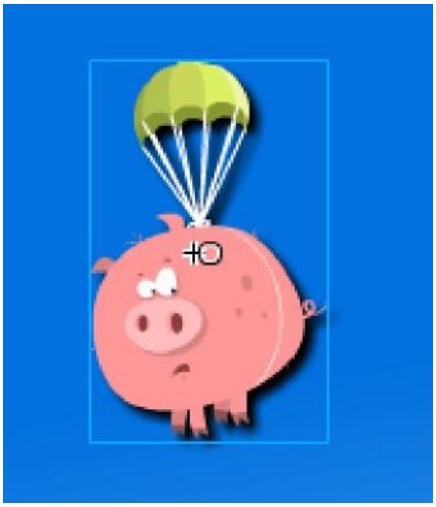

Figure 1.21
MovieClip avec le filtre DropShadow appliqué.

Si les filtres sont absolument nécessaires, rappelez-vous de les utiliser en basse qualité et ne modifiez pas les propriétés des filtres tout le temps pour éviter des calculs à chaque frame.

Le tableau ci-dessous vous montre lorsqu'il faut appliquer ou non des filtres sans risque de dégrader les performances.

Type de contenu	Utilisation des Filtres
Static	Oui
Animé	Oui
Propriétés des filtres animées	Non
Transformation (rotation, scale)	Non
Niché	Non

L'image ci-dessous montre un exemple, un **MovieClip** est déplacé sur la timeline avec un filtre de flou animé avec ses propriétés qui sont modifiées tout le temps :

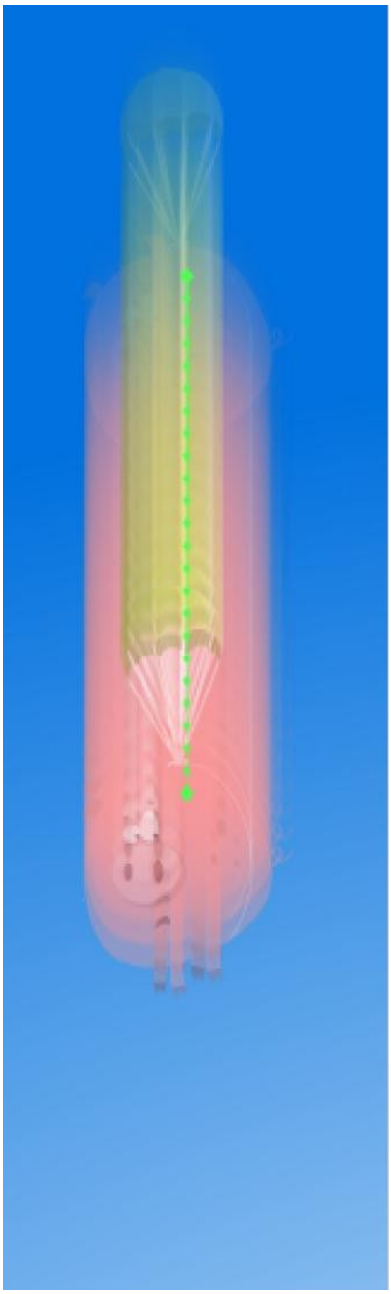

Figure 1.22
Le filtre Blur animé.

Maintenant, découvrons d'autres optimisations classiques relatives à la transparence.

L'abus de l'alpha blending

Essayez de limiter l'utilisation des effets de transparence.

Les designers et développeurs ont tendance à utiliser des effets d'alpha, qui aident à obtenir un affichage sympa, mais l'alpha est extrêmement coûteux en terme de performances CPU.

Évitez d'utiliser les effets nécessitant l'alpha en utilisant la propriété alpha, comme effets de fondu.

Rappelez-vous que l'alpha nécessite énormément de ressources CPU pour le Player Flash et peut nuire aux performances sur les machines lentes.

Supprimez la transparence des DisplayObject

La suppression de la transparence des DisplayObject améliore les performances de rendu.

Pour éviter au Flash Player les rendus de transparence, nous pouvons supprimer la transparence des fond des display objects en utilisant la propriété (de toutes les versions ActionScript) **opaqueBackground**.

Cette optimisation est très intéressante pour le rendu de texte pour lesquels il y a des dessins complexes ou coûteux.

Le code ci-dessous montre comment supprime la transparence en renseignant une couleur opaque pour le fond :

```
myTextField.opaqueBackground = 0xFFFFFF;
```

En combinant **cacheAsBitmap** et **opaqueBackground**, nous pouvons optimiser le rendu encore plus, le bitmap généré en cache est opaque et le rendu est plus rapide :

```
myTextField.opaqueBackground = 0xFFFFFF;

myTextField.cacheAsBitmap = true;
```

Pour plus d'information à propos de cela, rendez-vous sur « Rendu des objets textes » dans le paragraphe « Optimisation des performances pour la plateforme Adobe Flash » sur le guide disponible ici : http://www.adobe.com/go/optimize

Offstage content

> *N'ajoutez pas de contenu non visible sur le stage, ajoutez seulement les objets utilisés dans la display list.*

Dans la mesure du possible, n'ajoutez pas d'objets graphiques non visible sur le stage.

L'image ci-dessous montre une astuce des designers et développeurs pour réutiliser des éléments pendant l'exécution d'une animation :

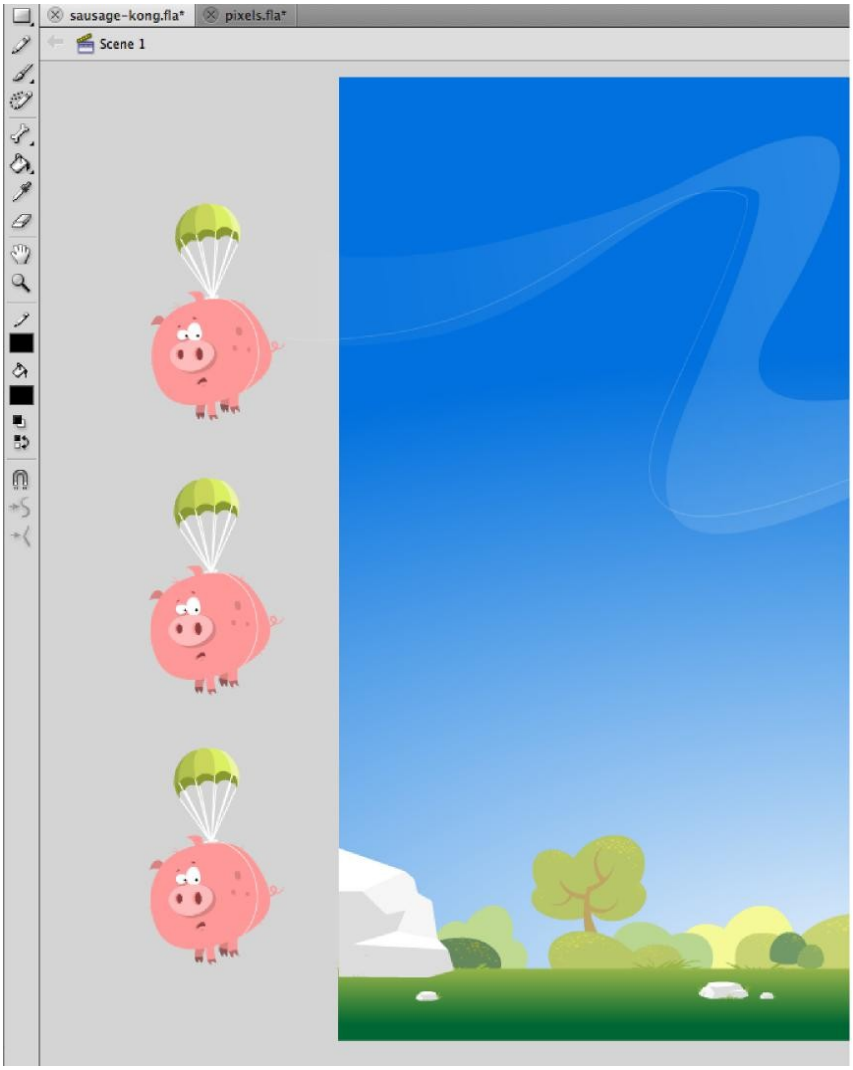

Figure 1.23
Du contenu sur le stage et non visible pour l'utilisateur.

Si les éléments non visible sur l'écran et qui ne sont pas affiché, il existe comme même dans la display list.

Le Flash Player effectue toujours des tests en interne pour s'assurer qu'ils sont toujours sur le stage et non visible...

Évitez, le plus souvent possible, de placer des objets non visibles sur le stage, supprimez les de la scène, tout simplement.

Optimiser le rendu avec l'option « redraw regions »

Utilisez toujours l'option « redrax regions » est pratique pour visualiser ce qui est redessiner et qui consomme du CPU.

L'option « redraw regions » est très intéressante pour tous les projets Flash sur lesquels vous travaillez.

L'option "redessiner les régions" est presque obligatoire pour chaque projet Flash sur lequel vous travaillez.

Cela vous permet de trouver facilement ce qui est en cours de rendu et ce que le Flash Player affiche réellement comme rendu.

Cela vous permet de détecter des zones inutiles redessiné et de les corriger ce pour optimiser le rendu, un outil très pratique pour le rendu.

L'option « redraw regions » peut être activée sans code, avec un click droit sur le SWF exécuté, en cochant « show redraw regions » dans le menu contextuel :

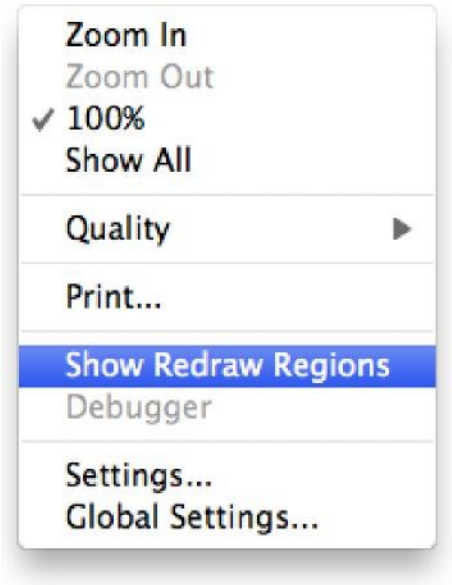

Figure 1.24
L'option « Show Redraw Regions » est disponible dans la version debug du Flash Player.

Vous pouvez aussi activer cette option par la programmation, en utilisant la méthode ActionScript 3 **flash.profiler.showRedrawRegions** :

```
// Activation de Show Redraw Regions
// La couleur bleu est utilisée pour afficher les zones redessinée
flash.profiler.showRedrawRegions ( true, 0x0000FF );
```

Ou simplement en ActionScript 1/2:

```
// Activation de Show Redraw Regions
// La couleur bleu est utilisée pour afficher les zones redessinée
showRedrawRegions ( true, 0x0000FF );
```

Rappelez-vous que cette option est seulement disponible dans la version debug du Flash Player.

Ci-dessous une copie d'écran montre un **MovieClip** animé avec l'option « redraw regions » activée :

Figure 1.25
L'option « Show Redraw Regions » activée.

En visualisant un autre objet, au dessus, nous pouvons croire que le **MovieClip** est stoppé et qu'il n'utilise pas de ressources CPU.

En utilisant l'option « redraw regions », nous voyons clairement que ce **MovieClip** utilise toujours du calcul pour le rendu :

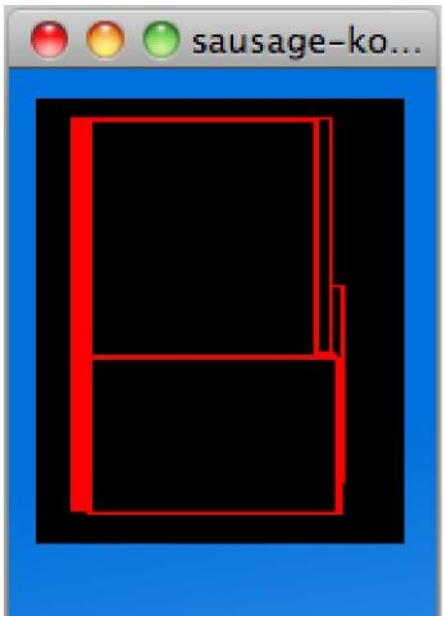

Figure 1.26

L'option « Show Redraw Regions » permet de voir le contenu caché en cours de rendu.

L'option « redraw regions » devrait toujours être utilisé pour comprendre le coût de rendu de chaque élément graphique dans votre animation.

TRAVAILLER AVEC LA VIDEO

Encodez une vidéo en utilisant des images lentes et évitez la transparence lorsque c'est possible.

Les publicités contiennent souvent des vidéos et c'est important que vos publicités puissent aussi être visualisées sur les appareils mobiles. Pour améliorer les performances, rappelez-vous ces points :

- Seulement les vidéos H.264 sont accélérées matériellement sur les mobiles et le bureau dans le Flash Player 10.1.
- La fonction de décodage matériel n'affecte pas la lecture de Sorensen Spark ou On2 VP6-encodé.

Assurez-vous de toujours utiliser le codec H.264 quand c'est possible et n'utilisez pas un framerate élevé dans les vidéos.

De plus, évitez la transparence dans les vidéos.

Quelques motion designer sont tentés d'utiliser la transparence dans les vidéos embarquées dans des MovieClip.

Dans tous les cas, la transparence est toujours plus consommatrice en rendu que le contenu opaque.

Remplacez les effets coûteux par de la vidéo

Remplacez les effets consommateurs de CPU par de la vidéo.

Lorsque les effets ont de besoin de beaucoup de transparence ou de filtres, utilisez plutôt la vidéo.

Dans une publicité, la taille est importante, mais dans certaines situations, utilisez la vidéo pour les rendus complexes est une meilleure solution.

Aussi la vidéo, la vidéo peut être créée avec un autre outil comme After Effects ou plus simplement avec Adobe Flash CS5, en utilisant la fonctionnalité « Exporter le film ».

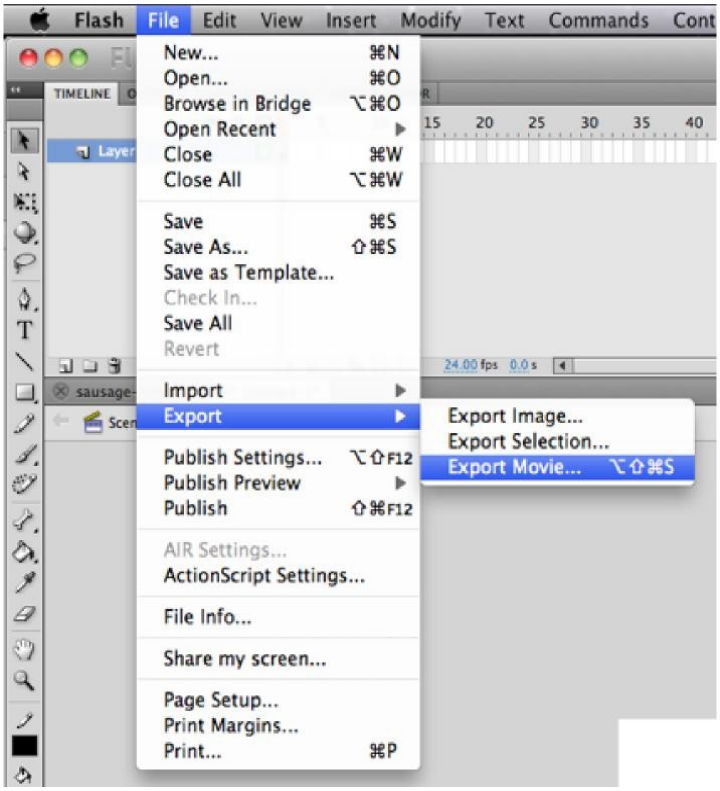

Figure 1.27
The « Export Movie » feature.

La vidéo peut être embarquée dans le swf ou chargée dynamiquement, mais ce choix rend le travail du designer plus compliqué et la gestion des fichiers distants n'est pas recommandé pour les publicités, où un fichier seul est plus facile à intégrer dans une page.

Regardons cela dans un exemple simple, l'image suivante montre des étoiles qui brillent la nuit :

Figure 1.28
Des étoiles animées.

Chaque étoile est composée d'un petit rond avec un dégradé de couleur qui est animé avec une oscillation transparente entre 0 et 100%.

L'image suivante explique cette idée :

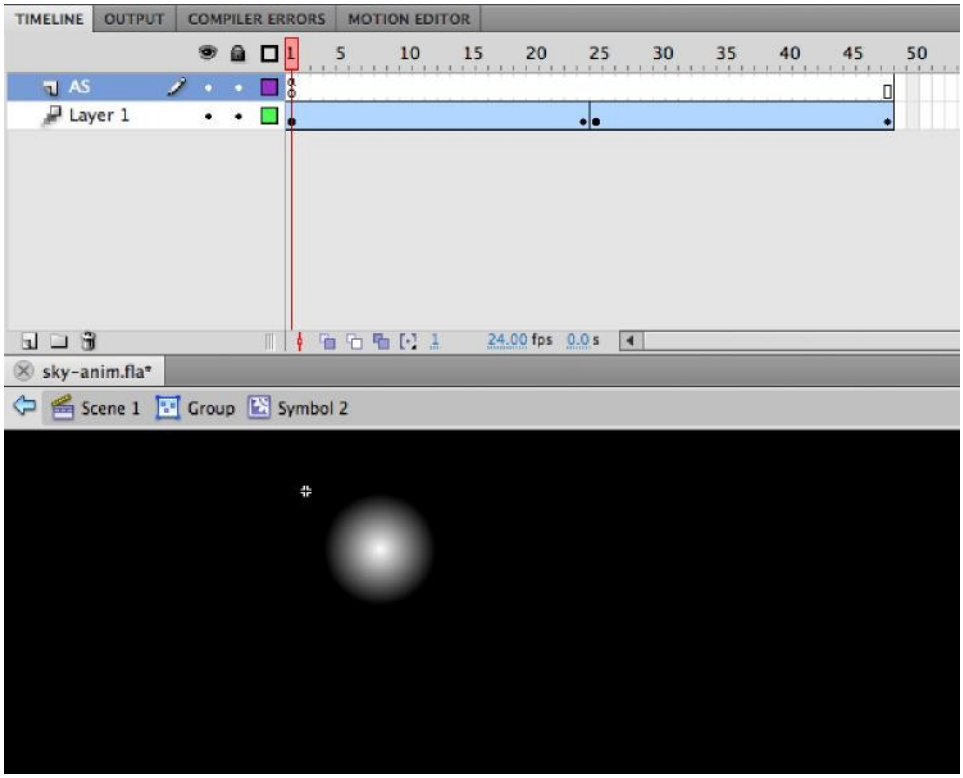

Figure 1.29
L'animation des étoiles avec un dégradé et de l'alpha.

Sur l'image 1.28, il y a environ 400 ou 500 étoiles qui sont animées, ce qui demande beaucoup de calculs pour le Player Flash.

Si vous utilisez un fond de ce style, cela nécessite beaucoup de ressources CPU.

Une meilleure implémentation est d'utiliser une vidéo pour cette animation, exportée à partir de Flash CS5 et placé sur le stage comme un fond.

Vous économisez ainsi des ressources CPU et créé une meilleure expérience de flou.

En utilisant le codec H.264, il y a de grandes chances que l'accélération du GPU soient utilisées par toutes les plate-formes (Windows, Mac et les appareils mobiles).

Note : rappelez-vous de ne pas utiliser la vidéo pour tout, mais seulement lorsque cela est approprié. Utilisez des douzaines de vidéos dans des objets animés avec des filtres sera pire que l'animation graphique sur la timeline.

OPTIMISATIONS ACTIONSCRIPT

Les publicités peuvent contenir aussi du code, les recommandations suivantes présentent quelques concepts essentiels que tout développeur de publicités doit prendre en compte pour les rendre plus performantes, plus rapides, nécessitant moins de ressources processeur et consomment moins les batteries.

Cela permet d'obtenir des publicités avec de meilleures performances, plus rapides, utilisant moins de ressources CPU et consommant moins de batteries.

Désactiver les DisplayObjects correctement

Toujours désactiver les DisplayObjects correctement.

En ActionScript 1 et 2, un **MovieClip** est automatiquement désactivé lorsqu'il est supprimé avec la méthode **removeMovieClip**.

En ActionScript 3, le freeze (blocage) des objets est un sujet vraiment important que tous les développeurs doit apprendre d'abord en découvrant ActionScript 3.

Pour optimiser le code ActionScript 3, supprimez toujours vos objets.

La suppression est très importante pour tous les objets,mais encore plus pour les display objects.

Toujours faire en sorte que les display objects ne soient pas dans la display list et qu'ils attendent le passage du garbage collector, ils peuvent utiliser du CPU.

Par exemple, ils peuvent encore utiliser **Event.ENTER_FRAME**. C'est un point critique de supprimer les objets proprement avec les évènements **Event.REMOVED** et **Event.ADDED_TO_STAGE**.

L'exemple ci-dessous montre un MovieClip en lecture sur le stage avec une interaction avec le clavier.

```
// Affiche ou supprimer le garçon

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);

}
```

A chaque fois qu'un élément est supprimé de la display list, le **MovieClip** envoie l'évènement **Event.ENTER_FRAME**.

Le MovieClip continue sa lecture, mais sans le rendu sur l'affichage.

Pour détecter cette situation correctement, écouter les évènements **Event.ADDED_TO_STAGE** et **Event.REMOVE_FROM_STAGE** et supprimez les écouteurs pour éviter une utilisation importante du CPU par l'exécution du code.

```
// Ecoute des évènements 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

}

function deactivate(e:Event):void

{

// en supprimant les écouteurs Freeze everything by removing event listeners

}
```

Lorsqu'un **DisplayObject** est ajouté ou supprimé du stage, nous enregistrons ou retiront tous les évènements requis ou utilisé.

Note : si un display object est supprimé de la display list, en définissant sa référence à null après sa suppression, cela ne garantit pas sa suppression de la mémoire.

Si le garbage collector ne s'exécute pas, l'objet continue de consommer de la mémoire et du CPU, tout le temps que l'objet n'est pas affiché.

Pour vous assurer que l'objet consomme le minimum de CPU que possible, vérifiez sa suppression complète de la display list.

Depuis le Player Flash 10, si le tête de lecture rencontre une frame vide, le display object est automatiquement supprimé même si vous ne l'avez pas spécifié par le code.

Le concept de suppression est aussi important pour le chargement de contenu distant avec la classe **Loader**.

Lorsque vous utilisez la classe Loader avec le Player Flash 9 en ActionScript 3, il est nécessaire de supprimé manuellement le contenu en écoutant l'évènement **Event.UNLOAD**, envoyé par l'objet **LoaderInfo.**

Chaque objet doit être manuellement supprimé, ce qui n'est pas une tâche facile.

Le Player Flash 10 a ajouté une importante nouvelle méthode sur la classe **Loader** qui s'appelle **unLoadAndStop.**

Cette méthode permet de supprimer un fichier swf, automatiquement, pour chaque objet chargé dans un fichier swf, et oblige le garbage collector à s'exécuter.

Dans le code ci-dessous, un fichier swf est chargé et supprimé en utilisant la méthode **unload**, ce qui nécessite pas plus de ressources qu'une suppression manuelle.

```
var loader:Loader = new Loader();
loader.load ( new URLRequest ( "content.swf" ) ); addChild ( loader );
stage.addEventListener ( MouseEvent.CLICK, unloadSWF );
function unloadSWF ( e:MouseEvent ):void
{
// Suppression du SWF file sans une désactivation automatique de l'objet
// Toutes les désactivations doivent être faites manuellement
loader.unload();
}
```

Une meilleure pratique est d'utiliser la méthode **unLoadAndStop**, ce qui supprime les objets et oblige l'exécution du garbage collector :

```
var loader:Loader = new Loader();
loader.load ( new URLRequest ( "content.swf" ) ); addChild ( loader );
stage.addEventListener ( MouseEvent.CLICK, unloadSWF );
function unloadSWF ( e:MouseEvent ):void
{
// Suppression du SWF file avec une désactivation automatique de l'objet
// Toutes les désactivations sont faites automatiquement
loader.unloadAndStop();
}
```

Les actions suivantes sont effectuées lorsque la méthode **unLoadAndStop** est appelée :

- Les sons sont stoppés.
- Les écouteurs enregistrés sur le fichier swf sont supprimés.
- Les objets Timer sont stoppés.
- Les appareils matériels (comme la caméra et le micro) sont désactivés.
- Tous les **MovieClip** sont stoppés.
- L'envoie des évènements **Event.ENTER_FRAME**, **Event.FRAME_CONSTRUCTED**, **Event.EXIT_FRAME**, **Event.ACTIVATE** et **Event.DEACTIVATE** sont arrêtés.

Depuis le Player Flash 9, lorsqu'une imagé clé blanche survient, le contenu graphique est supprimé du stage (display list) comme si l'API **removeChild** avait été appelée :

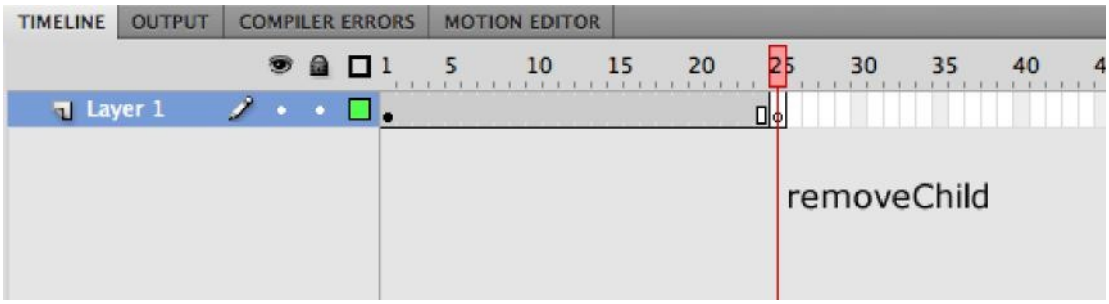

Figure 1.30
Comportement ActionScript 3 de la timeline (depuis le Player Flash 10).

Une amélioration importante a été apportée dans le Player Flash 10, lorsque les objets graphiques envoyaient l'évènement **Event.ENTER_FRAME** sont maintenant automatiquement désactivés,

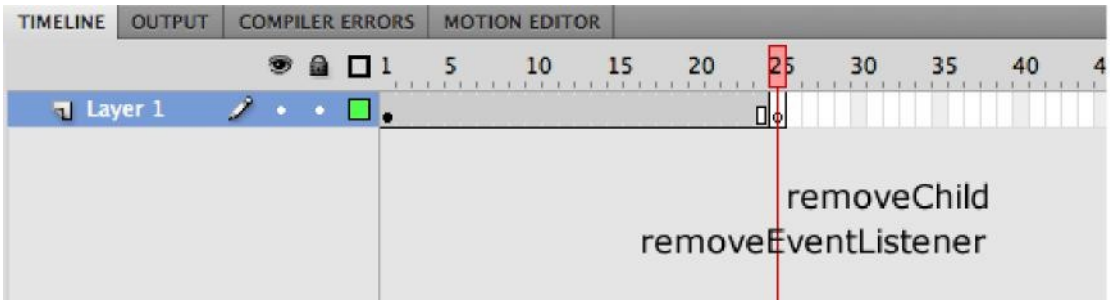

Figure 1.31
Comportement ActionScript 3 de la timeline (depuis le Player Flash 10).

Pour plus d'informations à propos de cela, rendez-vous sur le chapitre « freezing and unfreezing » dans le guide « optimiser les performances pour la plateforme Adobe Flash », disponible ici : http://www.adobe.com/go/optimize

Note : cette fonctionnalité est seulement disponible pour l'AS3.

En AS1 / AS2, une image clé blanche désactive totalement les objets graphiques qui sont supprimés du stage comme MovieClip.removeMovieClip.

Lorsque vous travaillez avec des publicités, il peut arriver qu'il soit nécessaire de synchroniser les animations, le Timer est utilisé pour cela.

Nous allons découvrir les pièges classiques à éviter lors de l'utilisation des Timers.

Travailler avec les Timers

> *N'utilisez jamais des Timers avec des intervalles court (moins d'une durée de 1000ms)dans vos SWF.*

Les Timers sont souvent utilisés par les designers et les développeurs pour le fonctionnement des tweens. Si l'intervalle utilisé est trop court, cela peut vraiment vider une batterie et consommer beaucoup de ressources CPU.

Note : choisissez entre le Timers ou l'évènement **ENTER_FRAME**, en fonction du contenu animé. Les Timers sont préférables que l'évènement **Event.ENTER**_FRAME pour le contenu non animé, qui s'exécute sur une période longue.

Quelque soit la version ActionScript utilisée, il y a 2 solutions pour appeler une fonction à un intervalle donné.

La première solution est d'utilisé l'évènement **Event.ENTER_FRAME** envoyé par les objets interactifs en ActionScript 3 ou l'évènement **enterFrame** en AS1 / AS2.

La deuxième solution est d'utiliser un objet Timer en ActionScript 3 et le setInterval en AS1 / AS2.

Les développeurs ActionScript utilisent souvent l'évènement **enterFrame**.

Avec comme résultat, l'intervalle avec lequel la fonction est appelée est en fonction du framerate.

Le framerate est accessible avec la propriété frameRate de l'objet stage en ActionScript 3. Attention, dans certains cas, l'utilisation du **Timer** est un choix plus judicieux que l'évènement entreFrame.

Par exemple, si vous n'utilisez pas d'animation, mais que vous voulez appelez votre code à des intervalles spécifiques, utilisez un **Timer** ou un **setInterval** peut être un choix judicieux.

Un Timer se comporte d'une façon similaire à un événement enterFrame, mais un événement peut être envoyé sans être lié à la vitesse de défilement (le fps).

Ce comportement peut offrir une optimisation significative.

Prenez, par exemple, un player vidéo. Dans ce cas, vous n'avez pas besoin d'un framerate élevé parce que seulement les contrôles de l'application se déplacent.

Note : Le framerate n'agit pas sur la vidéo, parce que les vidéos ne sont pas sur la timeline. La vidéo est chargée dynamiquement avec un téléchargement progressif ou en streaming.

Dans cet exemple, le framerate est mis à la valeur 10fps.

Le Timer met à jour les contrôles à la fréquence de 1 mise à jour chaque seconde.

Une mise à jour plus fréquente est possible avec la méthode **updateAfterEvent**, qui est disponible avec l'objet **TimerEvent.**

Cette méthode oblige l'écran à se mettre à jour à chaque fois que le Timer envoie un événement si nécessaire :

```
// Utilisez un framerate faible pour votre application stage.frameRate = 10;
// Choisissez une mise à jour par seconde
var updateInterval:int = 1000;
var myTimer:Timer = new Timer( updateInterval, 0 ); myTimer.start();
myTimer.addEventListener ( TimerEvent.TIMER, updateControls );
function updateControls ( e:TimerEvent ):void
{
// Mise à jour des contrôles ici
// Oblige les contrôles à se mettre à jour sur l'écran
e.updateAfterEvent();
}
```

Appelez la méthode **updateAfterEvent** ne modifie pas le framerate.

Cela oblige juste le Player Flash à mettre à jour le contenu qui a changé sur l'écran. La timeline continue de s'exécuter à 10 fps.

Rappelez-vous que les **Timers** et **Event.ENTER_FRAME** ne sont pas parfaitemetn précis pour les appareils à faibles performances, ou si la fonction qui écoute l'évènement contient du code qui demande beaucoup de ressources CPU.

Comme le framerate d'un fichier swf, la mise à jour du framerate du Timer peut varier dans certains cas :

Note : économisez le nombre d'objets Timer et d'écouteur enterFrame dans votre application.

A chaque frame, le runtime envoie un événement **Event.ENTER_FRAME** ou **enterFrame** pour chaque display object dans la display list.

Aussi, vous pouvez enregistrer des écouteurs pour l'évènement **Event.ENTER_FRAME** avec plusieurs display objects,

Centralisez tout le code qui s'exécute fréquemment.

De même, si vous utilisez des objets Timer, il est préférable

De même, si vous utilisez des objets Timer, il y a une surcharge CPU associée à la création et à l'envoi d'événements provenant d'objets Timer multiples.

Si vous devez déclencher les différentes opérations à différents intervalles, voici quelques solutions proposées:

- Utilisez un nombre minimum d'objets Timer et regroupez les opération en fonction de leur fréquence.
- Par exemple, utilisez un Timer pour les opérations fréquentes, réglez l'appel toutes les 100ms. Utilisez un autre Timer pour les opérations moins fréquentes ou en tâches de fond, réglez l'appel toutes les 2000ms.
- Utilisez un objet Timer unique et avec des appels à plusieurs fonction avec le paramètre delay.

Par exemple, imaginons que nous avons plusieurs opérations qui doivent être appelées toutes les 1000ms, et d'autres toutes les 2000ms.

Dans ce cas, utilisez un objet Timer unique, avec un délai de 1000ms.

Dans la fonction appelée par le Timer, ajouter une condition pour l'appel une fois sur deux pour les opérations toutes les 2000ms.

L'exemple ci-dessous vous montre cette technique :

```
var timer:Timer = new Timer(1000);
timer.addEventListener(TimerEvent.Timer, timerHandler); timer.start();
var offCycle:Boolean = true;
function timerHandler(event:TimerEvent):void {
// exécution toutes les 1000 ms
if (!offCycle)
{
// exécution toutes les 2000 ms
}
offCycle = !offCycle;
}
```

Note : pour un événement enterFrame ou une fonction appelée par un Timer, réduisez le nombre de modification de l'apparence des display objects, car cela oblige l'écran à se redessiner.

Pour chaque frame, la phase de rendu redessine sur une partie du stage qui a été modifié pendant cette frame.

Si la zone est grande à redessiner, ou si elle est petite, mais avec une grande quantité d'objets ou d'affichages complexes, le runtime nécessite plus de temps pour le rendu.

Pour tester la quantité de redessin nécessaire, utilisez l'option «voir les zones redessinées» dans le Flash Player ou AIR, avec la version Debug.

Les interactions avec la souris

Lorsque c'est possible, désactivez les interactions avec la souris.

Lorsque vous utilisez des objets interactifs, comme un **MovieClip** ou un objet **Sprite**, le Player Flash exécute du code en natif pour détecter les mouvements de la souris.

La détection des interactions avec la souris peut être consommateur de CPU quand il y a beaucoup d'objets interactifs sur l'écran surtout si ils se chevauchent !

Une solution pour éviter ces calculs est de désactiver les interactions avec la souris sur les objets n'en ayant pas l'utilité.

Le code suivant vous montre l'utilisation des propriétés **mouseEnabled** et **mouseChildren** en AS3 :

```
// Désactivation des interactions avec la sourisavec cet InteractiveObject
myInteractiveObject.mouseEnabled = false;
const MAX_NUM:int = 10;
// Création d'un container pour les  InteractiveObjects
var container:Sprite = new Sprite();
for ( var i:int = 0; i< MAX_NUM; i++ )
{
// Ajout des InteractiveObject dans le container
container.addChild( new Sprite() );
}
// Désactivation des interactions avec la souris sur tous les enfants
container.mouseChildren = false;
```

En AS1 / AS2, il n’est pas possible de désactiver les interactions de la souris.

A chaque fois que c’est possible, désactivez les interactions de la souris, cela permet de consommer moins de ressources CPU, et de réduire l’utilisation de la batterie.

CONCLUSION

Ce sont des optimisations très simple qui peuvent être appliquées à vos publicités 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.

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 publicités, sont liées à de simples recommandations graphiques.

C'EST MAINTENANT !

Rendez-vous sur cette page pour me dire ce que ce guide vous a apporté :

http://www.actionscript-facile.com/livret-formation-actionscript-optimiser-publicites-flash/article1212573.html

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

Profitez-en pour poster vos exemples de publicités flash optimisées !

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

Je tiens à remercier Thibault Imbert pour la réalisation de ce guide, très utile, sur l'optimisation des publicités avec Flash.

Merci pour votre lecture,

Matthieu DELOISON.

PRISE DE NOTES

L'Optimisation des Publicités Flash

Ce livret vous a été remis par le site **ActionScript-Facile.com**.

Vous pouvez librement l'imprimer, le prêter, le distribuer ou le faire suivre à un développeur qui appréciera les conseils et stratégies de programmation actionscript présentées ici.

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 :

http://www.actionscript-facile.com/

Livret de Formation

Ecrit par Thibault Imbert : **http://www.bytearray.org/?p=1586**

Traduction par Matthieu Deloison : **http://www.actionscript-facile.com/**

Rendez-vous sur **http://www.actionscript-facile.com/** pour d'autres cours en vidéos gratuits.