Arrêter une vidéo dans un SWF imbriqué

Un site bien conçu conduit généralement à isoler les rubriques dans des SWF séparés et à ne les importer que s'ils ont été invoqués par l'utilisateur. Dans ce contexte, si vous débutez en ActionScript 3, vous avez peut-être éprouvé des difficultés à vider un chargeur de son contenu, d'une part, mais surtout à interrompre l'audio d'un fichier vidéo ou d'une musique, que la rubrique chargée pouvait contenir. Lorsque vous supprimez de la liste d'affichage (`removeChild`) un chargeur, vous ne supprimez pas son contenu comme nous le faisions en AS2 – avec `unloadMovie()` –, vous le "désaffichez". Pour contourner cette difficulté, quatre solutions sont désormais disponibles :

- Vous pouvez interrompre le contenu d'un chargeur depuis le SWF qui a appelé le contenu avec une série d'instructions compatibles Flash 9.
- Vous pouvez procéder à la même manipulation avec une simple et unique instruction compatible Flash 10.
- Vous pouvez décharger et interrompre le SWF appelé depuis le SWF lui-même, avec des instructions compatibles Flash 9.
- Vous pouvez décharger et interrompre le SWF appelé depuis le SWF lui-même, avec quelques instructions compatibles Flash 10.

Dans cette section, nous abordons les quatre solutions pour interrompre les contenus. Mais nous détaillons également quelques notions chères aux anciens codeurs AS2, le ciblage des contenus imbriqués et l'accès aux contenus de la scène parente ou racine.

Dans cet exemple, nous disposons d'un premier SWF qui en appelle un autre, lequel joue une vidéo. Nous avons placé, à l'intérieur de chaque document, un bouton Fermer qui permet, chacun dans son contexte respectif et individuellement, de supprimer le contenu chargé, et cela avec les deux syntaxes Flash 9 et Flash 10.

Exemples > ch8_videoInteractive_9.fla et Exemples > ch8_videoInteractive_9b.fla

Dans le document "ch8_videoInteractive_9.fla", sur la scène, un MovieClip nommé `cible_mc` sert de conteneur pour le SWF chargé. Au-dessus, un contour blanc représente l'espace qu'occupe le contenu de ce SWF une fois celui-ci importé (voir Figure 8.19).

Dans la fenêtre de scénario, les symboles `cible_mc` et le bouton Fermer sont répartis distinctement vers les calques (voir Figure 8.20).

Dans le second document, nommé "ch8_videoInteractive_9b.fla", la scène affiche seulement un composant et un bouton de fermeture, localisé (voir Figure 8.21).

Dans la fenêtre de scénario du document chargé, le composant et le bouton fermer sont également répartis vers les calques (voir Figure 8.22).

La structure des deux documents permet de décharger et stopper les contenus aussi bien depuis le document racine que depuis le document appelé.

Figure 8.19
Aperçu du SWF de départ.

Figure 8.20
Fenêtre de scénario.

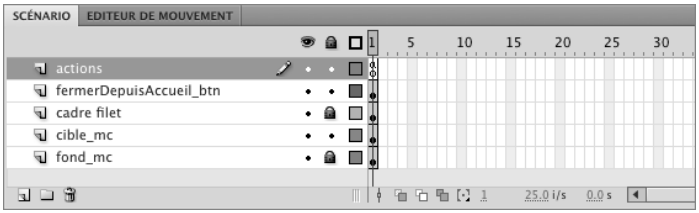


Figure 8.21
Aperçu du document SWF chargé.

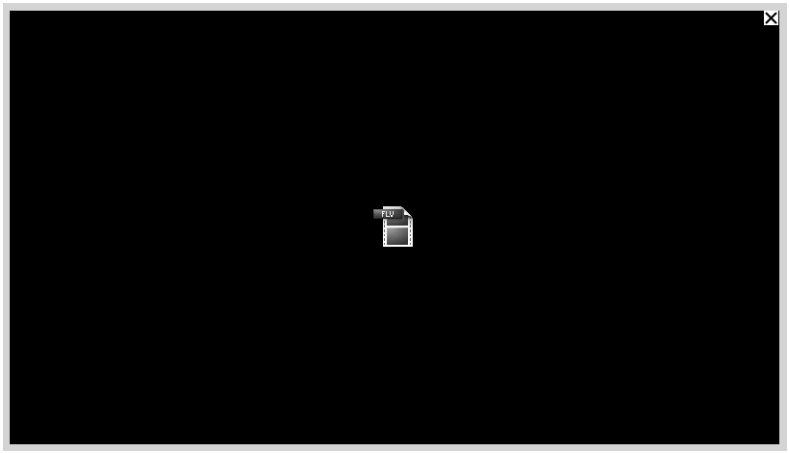
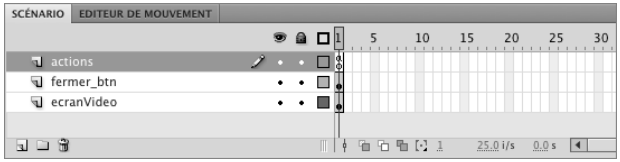


Figure 8.22
Fenêtre de scénario du document chargé.



Arrêt et fermeture depuis le document racine

Dans le calque `actions` du document racine ("ch8_videoInteractive_9.fla"), celui qui appelle l'autre, nous lisons le code suivant :

```
//---------------------- chargement

var chemin:URLRequest = new URLRequest("ch8_VideoInteractive_9b.swf");
var chargeur:Loader = new Loader();
chargeur.load(chemin);

chargeur.contentLoaderInfo.addEventListener(Event.COMPLETE, afficher);

function afficher(Evt:Event){
   cible_mc.addChild(chargeur);
}

//---------------------- fermer depuis l'accueil

fermerDepuisAccueil_btn.addEventListener(MouseEvent.CLICK,fermerSWF);
function fermerSWF (evt:MouseEvent) {
   /*
   //Méthode avant Flash 10
   MovieClip(chargeur.content).ecranVideo.stop();
   chargeur.unload();
   */

   chargeur.unloadAndStop();
}
```

Dans ce code, la première étape consiste à charger le second document SWF qui exécute la vidéo. Pour cela, nous définissions un nouveau chargeur, lequel exécute l'affichage du contenu une fois celui-ci chargé (voir Chapitre 5 pour le détail de ces instructions) :

```
//---------------------- chargement

var chemin:URLRequest = new URLRequest("ch8_VideoInteractive_9b.swf");
var chargeur:Loader = new Loader();
chargeur.load(chemin);

chargeur.contentLoaderInfo.addEventListener(Event.COMPLETE, afficher);

function afficher(Evt:Event){
   cible_mc.addChild(chargeur);
}
```

Plus loin, nous ajoutons un écouteur sur le bouton nommé `fermer_btn`. Dans la fonction appelée par l'écouteur, nous distinguons deux types d'instructions. La première,

laissée en commentaire pour ne pas doubler celle qui est restée active, est compatible Flash 9 :

```
MovieClip(chargeur.content).ecranVideo.stop();
chargeur.unload();
```

Si nous nous contentions de purger le chargeur, le contenu serait bien ôté de l'interface, mais le flux ne serait pas stoppé, un peu comme si le robinet continuait à déverser de l'eau alors que l'on ait retiré le sceau. Le son et l'image continuent de se lire. Cette commande consiste donc, dans un premier temps, à interrompre tout ce qui est exécuté. En l'occurrence, la vidéo. Puis, sur la deuxième ligne, à purger le chargeur.

Pour cibler plus spécifiquement le contenu chargé dans un SWF, nous devons d'abord faire référence au chargeur que nous avons utilisé pour importer le contenu. Plus précisément, nous devons faire référence à ce qu'il contient, et non à l'enveloppe qu'il représente. Nous spécifions donc, en plus de reprendre son nom (chargeur), que nous ciblons son contenu (`chargeur.content`).

Ensuite, la référence au contenu du chargeur est passée en paramètre d'une méthode de transtypage en MovieClip. Et la deuxième ligne, une fois la vidéo arrêtée, purge le chargeur de son contenu.

Pourquoi transtyper un chargeur en MovieClip ?

Il faut comprendre que chaque document AS3 possède désormais une structure où la scène principale n'est plus un simple MovieClip (comme ce fut le cas en AS1 ou en AS2), mais bien un objet de type `Sprite`, qui contient la scène principale MovieClip. Une couche intermédiaire a donc été ajoutée à la structure d'un document Flash. Si cette modification structurelle a permis d'apporter une plus grande stabilité et de déployer de nombreuses fonctionnalités dans Flash, elle ne permet plus de cibler directement un contenu chargé dans un SWF imbriqué. Chaque document est réellement indépendant. Le `Sprite` ne possède pas les mêmes propriétés de ciblage que MovieClip. Pour transgresser cette contrainte, il faut donc transtyper le `Sprite` qui contient la scène MovieClip du document importé, en MovieClip. Ainsi, toute la chaine d'imbrication préserve une structure intégralement composée de MovieClip imbriqués, et autorise de nouveau le ciblage. Pour en savoir plus sur la structure native d'un document Flash en AS3 et sur le principe de la liste d'affichage, consultez la documentation Adobe à la page suivante : **http://livedocs.adobe.com/flash/9.0_fr/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts_bak&file=00000142.html.**

Cette première méthode, compatible Flash 9, peut vite devenir astreignante si les contenus chargés deviennent multiples. C'est la raison pour laquelle, à la demande des utilisateurs, une nouvelle instruction est apparue avec Flash 10 : `unLoadAndStop()`.

```
chargeur.unloadAndStop();
```

Concernant le ciblage du contenu chargé, le principe reste donc le même. Mais une seule instruction suffit désormais pour interrompre tous les contenus et purger le SWF. Voici la liste des actions désormais associées à cette nouvelle méthode :

- Les flux audio et vidéo sont arrêtés.

- Les écouteurs d'événements sont supprimés de la scène.
- Les écouteurs d'événements `enterFrame`, `frameConstructed`, `exitFrame`, `activate` et `deactivate` sont supprimés.
- Les timers (horloges) sont arrêtées.
- Les occurrences `Camera` et `Microphone` sont retirées.
- Les clips sont stoppés.

Arrêt et fermeture depuis le contenu chargé

Une autre technique, pour fermer un contenu, consiste à placer les actions directement dans le SWF chargé.

Dans le calque `actions` du document appelé ("ch8_videoInteractive_9b.fla"), contenant la vidéo, nous lisons ce code-ci :

```
 //---------------------- fermer depuis le SWF chargée

fermer_btn.addEventListener(MouseEvent.CLICK,fermerSoiMeme);
function fermerSoiMeme (evt:MouseEvent) {
   /*
   //Méthode avant Flash 10
   ecranVideo.stop();
   MovieClip(root.parent.root).chargeur.unload();
   */

   MovieClip(root.parent.root).chargeur.unloadAndStop() ;

//MovieClip(this.stage.getChildAt(0)).chargeur.unloadAndStop();
}
```

Dans ce code, un écouteur est attaché à un bouton situé sur la scène principale, localement. Cet écouteur appelle une fonction qui propose deux types d'instructions : celles compatibles Flash 9 et Flash 10.

Les deux premières instructions sont en commentaire, pour de nouveau éviter de doubler l'instruction active :

```
ecranVideo.stop();
MovieClip(root.parent.root).chargeur.unload();
```

La première ligne interrompt d'abord le flux vidéo contenu dans la scène courante, comme nous le ferions simplement à travers une action classique.

La deuxième reprend le principe de ciblage du contenu dans le chargeur, évoqué précédemment. Mais ce chargeur ayant été créé dans la scène du document parent, nous le ciblons en y faisant référence par transtypage.

En paramètre de la méthode `MovieClip()`, nous indiquons simplement le chemin qui nous permet d'accéder au contenu ciblé, avec `root.parent.root`.

Ciblage dans un document Flash AS3

Avec la structure `Sprite` d'un document AS3, `root`, qui autrefois désignait la racine globale de l'ensemble des documents Flash imbriqués (d'équivalent `_root` en AS1 et en AS2), ne cible désormais plus que la scène courante.

Il est en revanche toujours possible de remonter d'un niveau en utilisant la propriété `parent`, de deux niveaux en utilisant `parent.parent`, et ainsi de suite. Mais pour ce faire, vous devez préciser de quel objet vous partez, en ajoutant à la position de départ et celle d'arrivée un `root`. Vous obtenez alors `root.parent.parent.root`, pour cibler la scène courante d'un conteneur de conteneur parent.

En ActionScript 3, la racine globale, qui correspondait avant à `_root`, est désormais désignée par la propriété stage de l'objet conteneur que constitue le document racine. Pour l'invoquer, nous écrivons à présent `this.stage.getChildAt(0)`.

Voir aussi la note suivante pour un exemple de ciblage avec `this.stage.getChildAt(0)`.

Exemples de ciblage de contenus dans des SWF imbriqués (AS2 et AS3)

En ActionScript 2, pour cibler un contenu dans un autre SWF, nous procédions ainsi :

- Cibler un symbole intitulé `clip_mc` dans un SWF chargé, contenu dans `cible_mc` : `cible_mc.clip_mc.rotation=90;`
- Cibler un symbole intitulé clip_mc dans un SWF parent, contenu dans cible_mc : _parent.cible_mc.clip_mc.rotation=90;
- Ou bien, si le SWF parent est le conteneur principal : _root.cible_mc.clip_mc.rotation=90;
- Pour cibler un contenu intitulé clip_mc et placé deux conteneurs en amont, nous inscrivions : _parent._parent.cible_mc.clip_mc.rotation=90;

En ActionScript 3, pour cibler un contenu dans un autre SWF, nous procédons désormais ainsi :

- Cibler un symbole intitulé `clip_mc` dans un SWF chargé, contenu dans `cible_mc`. Le nom du conteneur n'apparaît plus. Nous invoquons directement le chargeur :

```
MovieClip(nomDuChargeur.content).clip_mc.rotation=90;
```

- Cibler un symbole intitulé `clip_mc` dans un SWF parent, contenu dans `cible_mc` :

```
MovieClip(root.parent.root).cible_mc.clip_mc.rotation=90;
```

- Ou bien, si le SWF parent est le conteneur principal (équivalent en AS3 de `_root`) :

```
MovieClip(this.stage.getChildAt(0)).cible_mc.clip_mc.rotation=90;
```

- Pour cibler un contenu intitulé `clip_mc` et placé deux conteneurs en amont, nous inscrivions :

```
MovieClip(root.parent.parent.root).cible_mc.clip_mc.rotation=90;
```

À défaut d'utiliser la méthode `getChildAt()`, vous pouvez également cibler un objet en utilisant `getChildByName()`. Mais vous devez alors connaître le nom de l'objet pour le passer en parameter de cette méthode. Ce qui donne, par exemple :

```
getChildByName("clip_mc");
```

Dans notre exemple, la fonction propose aussi une instruction compatible Flash 10 :

```
MovieClip(root.parent.root).chargeur.unloadAndStop();
//MovieClip(this.stage.getChildAt(0)).chargeur.unloadAndStop();
```

Comme énoncé plus haut, nous ciblons ici le chargeur situé dans le SWF parent. Puis, nous reprenons, comme pour le lien placé dans le SWF parent, la méthode `unloadAndStop()`.

Une variante de la première ligne est ajoutée en commentaire. Elle propose une alternative à la syntaxe `root.parent.root`, pour cibler directement la scène du conteneur principale (avec `this.stage.getChildAt(0)`), que nous aurions autrefois appelée avec `_root`.

En publiant les deux documents, le premier charge le second et exécute directement la vidéo. Lorsque vous cliquez sur le lien situé en haut et à droite de la vidéo, comme sur le lien situé sous la vidéo à droite, celle-ci est d'abord arrêtée, puis disparaît (voir Figure 8.23).

À retenir

- Il est possible de cibler des objets dans des documents imbriqués et, inversement, de se référer à des objets placés dans un document racine, en ActionScript 3. Pour cela, nous transtypons les chargeurs avec la méthode `MovieClip()` et ciblons leur contenu avec la propriété content.
- Pour supprimer un document imbriqué qui joue une vidéo, il faut d'abord interrompre le flux, puis purger le chargeur qui a appelé ce contenu.
- Une nouvelle méthode, en ActionScript 3, permet d'interrompre les contenus chargés et de purger le chargeur, simultanément : `unLoadAndStop()`.

Figure 8.23
Aperçu du document.

Synthèse

Dans ce chapitre, vous avez appris à contrôler l'interactivité sur la vidéo en créant vous-même vos propres commandes pour, entre autres, lire, stopper, accélérer et rembobiner la vidéo et modifier le volume sonore avec progression. Vous avez appris à réaliser un système de sous-titrage en utilisant un affichage de texte personnalisé, à créer différents dispositifs de chapitrage. Vous avez appris à réaliser des présentations vidéo fluides y compris quand elles sont lues en arrière et à vitesse variable. Vous avez appris à gérer l'intégration de contenus vidéos dans des documents imbriqués et à contourner la complexité de l'ActionScript 3 pour arrêter les contenus audio et vidéo chargés. D'une manière plus générale, vous avez aussi appris à maîtriser le ciblage de contenus dans une structure de site complexe. Vous êtes à présent en mesure de réaliser des sites richmédia impliquant l'organisation de contenus vidéos et une couche importante d'interactivité.

9 La 3D native

Introduction

La gestion de la 3D est possible nativement dans Flash depuis la version CS4. Mais elle reste toutefois limitée à l'animation d'objets 2D dans un espace 3D. Dès lors que l'utilisateur dispose d'un lecteur Flash récent (Flash 10), il peut l'exécuter. Les propriétés 3D des objets du scénario, couplées à des interpolations programmées en ActionScript (avec la classe `TweenMax`), ouvrent de nouvelles "perspectives" dans la conception de sites Internet.

Dans ce chapitre, nous allons voir comment programmer des animations en 3D, dans Flash, de manière simple et accessible. Pour cela, nous utilisons en premier exemple un livre avec des pages qui tournent – un flipbook –, puis, nous analyserons des systèmes de présentation de contenus avec différents types de galeries 3D, impliquant entre autres des images et de la vidéo.

À l'issue de ce chapitre, vous saurez créer des interfaces 3D conviviales, en toute simplicité.

Les propriétés 3D disponibles nativement pour l'animation dans Flash sont les suivantes :

- `x` positionne l'objet sur l'axe des abscisses.
- `y` positionne l'objet sur l'axe des ordonnées.
- `z` positionne l'objet sur l'axe Z.
- `rotationX` effectue une rotation de l'objet sur l'axe des abscisses.
- `rotationY` effectue une rotation de l'objet sur l'axe des ordonnées.
- `rotationZ` effectue une rotation de l'objet sur l'axe Z.

Ces propriétés permettent de déplacer et pivoter les objets dans l'espace de la même manière que nous le faisons avec d'autres propriétés, y compris à travers des animations programmées ou non.

Info

Dans Flash, l'axe X représente la ligne horizontale qui part de la gauche vers la droite. L'axe Y, la ligne qui descend verticalement. L'axe Z, enfin, désigne la ligne frontale qui part du premier plan (l'écran) et se dirige vers l'infini en arrière-plan (voir Figure 9.1).

La rotation dans l'espace 3D se détermine par rapport aux axes X, Y et Z. Une rotation en X fait pivoter un objet sur l'axe X tout en le maintenant dans sa direction à la manière du balai d'une moissonneuse (voir Figure 9.2). Une rotation en Y fait pivoter un objet autour de l'axe des ordonnées (axe vertical) à la manière des lames d'un mixeur. Une rotation en Z applique un mouvement circulaire sur l'axe frontal des profondeurs, à la manière des ailes d'un moulin à vent que nous observerions de face.

Figure 9.1
Définition des axes X, Y et Z d'un espace 3D.

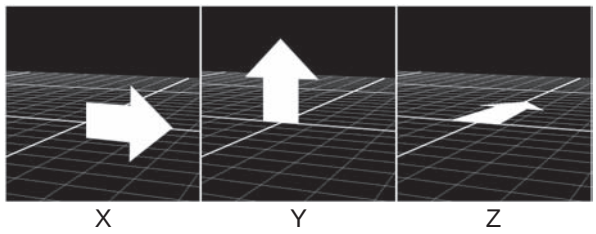
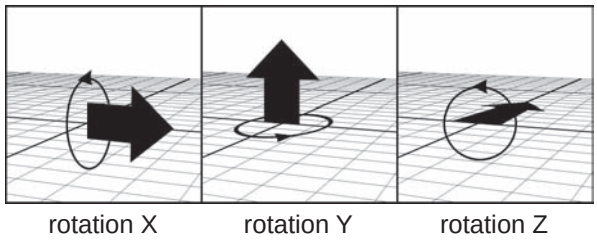


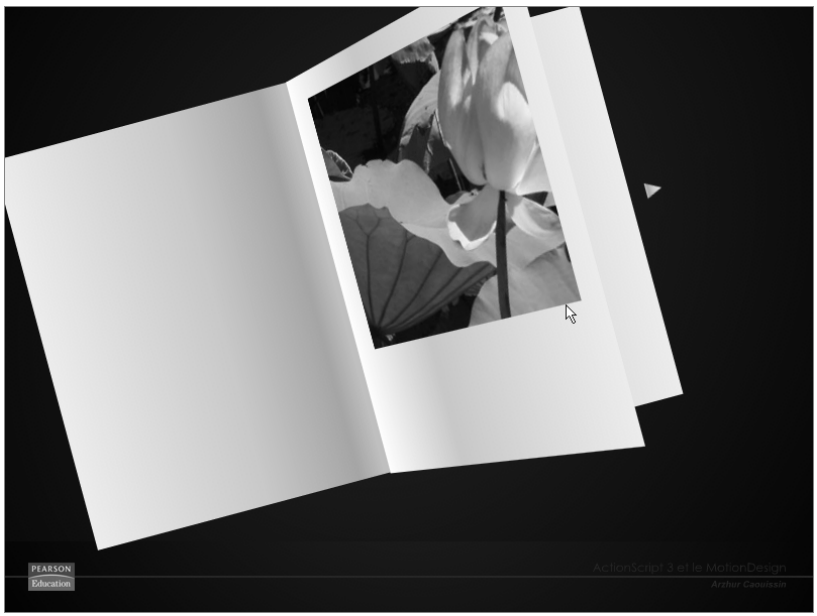
Figure 9.2
Définition des rotations X, Y et Z dans un espace 3D.



Animer un livre 3D en ActionScript

Dans cette section, nous allons aborder la conception d'un livre avec des pages qui tournent. Pour cela, nous utilisons la classe `TweenMax`, compatible avec les propriétés 3D de Flash ainsi que des propriétés de rotation dans l'espace (voir Figure 9.3).

Figure 9.3
Aperçu après publication.



En publiant le document, une animation fait entrer le livre au centre de l'écran. En cliquant sur la flèche située à droite du livre ou sur le livre lui-même, la page tourne sur un axe 3D et laisse apparaître la page suivante, ceci pour toutes les pages. Une fois le livre terminé, une animation 3D le retire de l'écran en le faisant pivoter dans l'espace 3D, sur lui-même.

Exemples > ch9_3DNative_1.fla

Dans le document "ch9-3DNative-1.fla", sur la scène principale, au-dessus du calque `fond_mc`, apparaît un symbole de type MovieClip nommé `livre_mc` (voir Figures 9.4 et 9.5). Le livre est placé hors champ.

Figure 9.4
Aperçu de la scène principale.

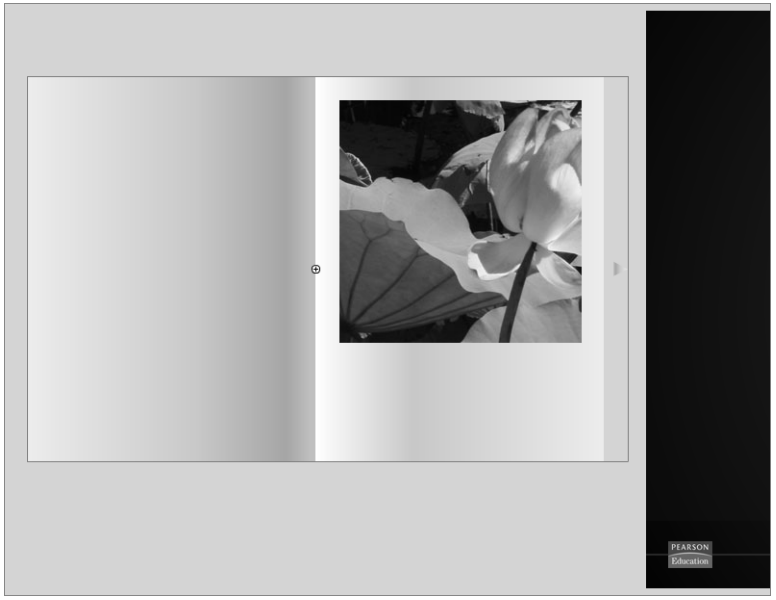
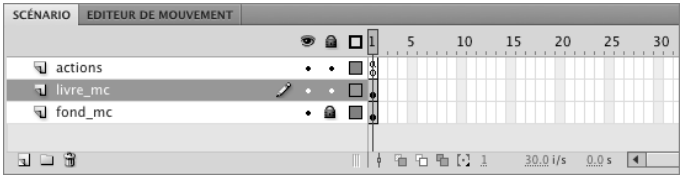


Figure 9.5
Aperçu du scénario de la scène principale.

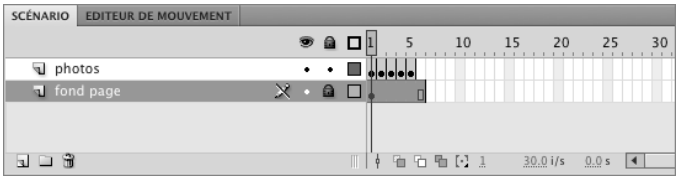


Le MovieClip `livre_mc` contient quatre symboles. Le premier représente la page de gauche (livre ouvert), c'est-à-dire, la page de couverture. Un autre clip représente la page de droite et comporte l'ensemble des écrans de toutes les pages du livre. En effet, une seule page est animée et elle laisse donc derrière elle un vide à chaque action. Une page Arrière sert de fond pendant les transitions. Enfin, un bouton en forme de flèche active le mécanisme des pages qui tournent (voir Figure 9.6).

Figure 9.6
Aperçu du scénario du symbole `pageDroite_mc`.

Le symbole `pageDroite`, ayant pour nom d'occurrence `pageDroite_mc`, contient une série d'écrans qui constituent les différentes pages du livre. Chaque écran est distribué dans le scénario sous la forme d'images-clés, placées les unes à la suite des autres. La dernière image est vide. Elle représente le verso de la page tournée (voir Figure 9.7).

Figure 9.7
Aperçu du scénario du symbole `pageDroite_mc`.



Sur la scène principale, au sommet de tous les calques, figure le calque `Actions` (voir Figure 9.5). Dans la fenêtre Actions, nous pouvons lire le code suivant :

```
import gs.TweenMax;
import gs.easing.*;
import gs.events.*;

var tweenPage:TweenMax;
var nombreDePages:Number=livre_mc.pageDroite_mc.totalFrames-1;
var i:Number=1;

livre_mc.pageDroite_mc.stop();

//-------------------------- animation d'introduction
var tween3D:TweenMax=TweenMax.to(livre_mc,4,{x:330, rotationZ:-15, delay:0,
➥ ease:Strong.easeInOut});
TweenMax.from(livre_mc.pageGauche_mc,2,{rotationY:-180, delay:2,
➥ ease:Strong.easeInOut});
TweenMax.from(livre_mc.flecheDroite_btn,2,{alpha:0,delay:4,
➥ ease:Strong.easeInOut});

//-------------------------- cliquer pour tourner les pages
livre_mc.addEventListener(MouseEvent.MOUSE_DOWN,pageSuivante);

function pageSuivante (evt:MouseEvent) {
   i++;
   // page de garde
   if (i<=1) {
      livre_mc.swapChildren(livre_mc.pageGauche_mc,livre_mc.pageDroite_mc);
   }
   // page de droite
   if (i<=nombreDePages) {

tween3D=TweenMax.to(livre_mc.pageDroite_mc,1,{rotationY:180,ease:Strong.easeOut});
```

```
      tween3D.addEventListener(TweenEvent.COMPLETE, tween3DFini);
      } else {
         //animation de fermeture
         livre_mc.pageArriere_mc.visible=false;

   TweenMax.to(livre_mc.pageDroite_mc,1,{rotationY:180,ease:Strong.easeOut});
         TweenMax.to(livre_mc,4,{rotationZ:360, x:1000, z:-500, alpha:0,
         ➥ rotationY:360, delay:1, ease:Strong.easeInOut});
         livre_mc.flecheDroite_btn.visible=false;
         tweenPage.addEventListener(TweenEvent.COMPLETE, tweenPageFini);
      }
   }

   function tween3DFini (evt:TweenEvent) {
      livre_mc.pageDroite_mc.gotoAndStop(i);
      TweenMax.from(livre_mc.pageDroite_mc,1,{alpha:0,ease:Strong.easeOut});
      livre_mc.pageDroite_mc.rotationY=0;
   }

   function tweenPageFini (evt:TweenEvent) {
      removeEventListener(Event.ENTER_FRAME,masquerVersoPage);
   }

   //-------------------------- masquer le verso des pages qui tournent
   addEventListener(Event.ENTER_FRAME,masquerVersoPage);
   function masquerVersoPage (evt:Event) {
      if (livre_mc.pageDroite_mc.rotationY>90) {
         livre_mc.pageDroite_mc.gotoAndStop(6);
      }
   }
```

Ce code est structuré en quatre parties : l'initialisation, l'animation d'introduction, les actions des pages qui tournent et la gestion du verso des pages tournées.

La première partie du code importe les classes utilisées pour les transitions `TweenMax` :

```
import gs.TweenMax;
import gs.easing.*;
import gs.events.*;
```

À la suite, nous définissons quelques variables. La première identifie une interpolation `TweenMax` en vue de lui faire succéder d'autres actions :

```
var tweenPage:TweenMax;
```

La deuxième compte le nombre de pages du livre :

```
var nombreDePages:Number=livre_mc.pageDroite_mc.totalFrames-1;
```

La méthode `totalFrames`, abordée au Chapitre 8, permet ici de compter le nombre total d'images dans le scénario du symbole `pageDroite_mc`. Nous lui retranchons cependant 1 afin de ne pas comptabiliser la page vide utilisée pour le verso.

Nous définissons ensuite une variable `i`, utilisée pour l'incrémentation des pages, à chaque fois qu'une page est tournée. La valeur démarre à 1, car la première image du scénario de `pageDroite_mc` démarre aussi à 1 et, bien que masquée, est déjà active.

```
var i:Number=1;
```

Enfin, nous stoppons le défilement automatique de la tête de lecture dans le symbole `pageDroite_mc`. À défaut, nous les verrions toutes défiler jusqu'à la première interaction. Ceci équivaut à placer un `stop` directement en tête de la première image de ce même clip :

```
livre_mc.pageDroite_mc.stop();
```

Les premières instructions consistent à réaliser une animation d'introduction pour placer le livre en position de lecture, prêt à être ouvert. Nous déployons pour cela, dans cette deuxième partie, quelques transitions de type `TweenMax` :

```
//------------------------- animation d'introduction
var tween3D:TweenMax=TweenMax.to(livre_mc,4,{x:330, rotationZ:-15, delay:0,
➥ ease:Strong.easeInOut});
TweenMax.from(livre_mc.pageGauche_mc,2,{rotationY:-180, delay:2,
➥ ease:Strong.easeInOut});
TweenMax.from(livre_mc.flecheDroite_btn,2,{alpha:0,delay:4,
➥ ease:Strong.easeInOut});
```

Dans ces interpolations, nous plaçons, en paramètre, des propriétés 3D (`rotationZ`, `rotationY`). Nous utilisons les propriétés 3D de la même manière que les autres propriétés (`x`, `alpha`, `scaleX`, etc.).

La première occurrence d'animation est typée. C'est pour nous permettre de disposer d'un nom d'objet, par la suite, sur lequel nous pourrons placer un écouteur et enchaîner avec d'autres actions (voir Chapitre 2).

Dans notre animation, le livre vient se positionner dans le champ en s'inclinant légèrement à –15° (`rotationZ:15`). Cette transition dure quatre secondes et démarre à l'instant 0.

Puis, dès la deuxième seconde, dans la deuxième interpolation, la page de gauche du livre pivote sur son axe Y (`rotationY:-180`) et révèle la page de droite placée à l'arrière-plan. Cette transition dure deux secondes et se termine en même temps que la première. Les effets des deux animations se superposent dans le temps. Enfin, une transition fait apparaître la flèche droite et signale l'interactivité. Les enchaînements de ces transitions ont été gérés avec la propriété `delay`, sans recours aux écouteurs.

L'accumulation de filtres sur des objets animés en 3D, avec des textures dégradées et des animations en alpha, sollicite beaucoup les ressources graphiques. Si des sautillements apparaissent, pensez à réduire ces effets.

Dans la troisième partie, un écouteur est attaché au livre. Il appelle la fonction `pageSuivante` :

```
//------------------------- cliquer pour tourner les pages
livre_mc.addEventListener(MouseEvent.MOUSE_DOWN,pageSuivante);

function pageSuivante (evt:MouseEvent) {
// instructions des pages qui tournent.
}
```

Nous ciblons directement le livre, et non pas seulement la flèche contenue dans le livre, afin de permettre à l'utilisateur d'activer le changement de page même en cliquant sur le livre. Puisque la flèche est contenue dans le symbole du livre, le script fonctionnera également en

cliquant sur la flèche. En proposant une action plus étendue, nous renforçons l'ergonomie du projet.

À l'intérieur du bloc d'instruction de la fonction, nous commençons par incrémenter `i` :

```
i++;
```

Puis, nous vérifions sa valeur. Si cette valeur est inférieure au nombre de pages à visiter, alors, nous intervertissons, dans la liste d'affichage, les index de position des symboles `pageGauche` et `pageDroite` :

```
// page de garde
if (i<=1) {
   livre_mc.swapChildren(livre_mc.pageGauche_mc,livre_mc.pageDroite_mc);
}
```

La méthode `swapChildern()` appelle en paramètres les deux objets, séparés par une virgule, à substituer dans la liste d'affichage (`livre_mc.pageGauche_mc`, `livre_mc.pageDroite_mc`). Le premier objet prend ainsi l'index du second.

Nous procédons à l'interversion des objets car l'animation 3D se matérialise sur le niveau de l'objet uniquement, et non sur l'ensemble de la scène. La scène n'est pas réellement projetée en 3D. Seuls les objets le sont individuellement. Des propriétés générales existent pour la scène, mais les transformations des objets se font localement, à chaque niveau. Si bien que lorsque nous faisons pivoter un objet dans l'espace, si celui-ci est placé sous un symbole, l'objet restera toujours en arrière-plan. Pour contourner ce problème, nous replaçons au premier plan (`pageGauche`), celui placé à l'arrière-plan (`pageDroite`).

Une fois la valeur de `i` vérifiée et l'index de la page de droite placé au premier plan, nous déployons le mécanisme de rotation :

```
// page de droite
  if (i<=nombreDePages) {

tween3D=TweenMax.to(livre_mc.pageDroite_mc,1,{rotationY:180,ease:Strong.easeOut});
  tween3D.addEventListener(TweenEvent.COMPLETE, tween3DFini);
  }
```

Nous programmons la rotation de la page à l'aide de la propriété `rotationY` et d'une interpolation de type `TweenMax`. La page tourne donc bien sur sa verticale. Enfin, pour nous assurer que l'objet pivote sur sa tranche et non en son centre, nous avons placé le centre du symbole à gauche, dès sa création (voir Figure 9.8).

Un écouteur est attaché au nom d'objet de l'interpolation `TweenMax` pour nous permettre de lancer une autre action, une fois l'animation terminée. Cette fonction, qui est exécutée quand la page est tournée, déplace la tête de lecture dans le symbole `pageDroite` et replace la page tournée à sa position initiale :

```
function tween3DFini (evt:TweenEvent) {
   livre_mc.pageDroite_mc.gotoAndStop(i);
   TweenMax.from(livre_mc.pageDroite_mc,1,{alpha:0,ease:Strong.easeOut});
   livre_mc.pageDroite_mc.rotationY=0;
}
```

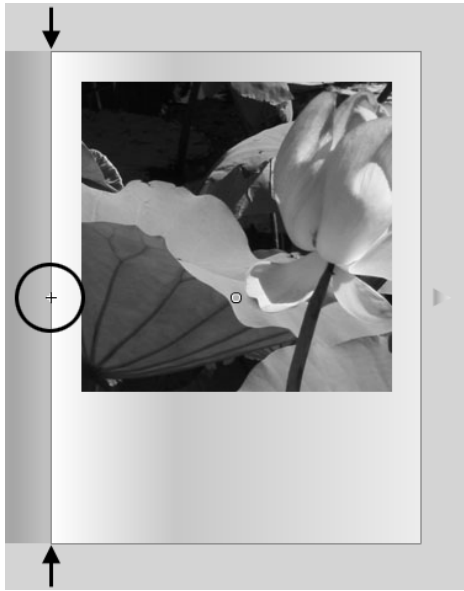


Figure 9.8
La position du centre détermine l'axe de rotation.

Nous avons également ajouté une interpolation qui affiche la page avec un fondu de type alpha, allant de 0 à 1 (avec `from`) :

```
TweenMax.from(livre_mc.pageDroite_mc,1,{alpha:0,ease:Strong.easeOut});
```

Nous devons comprendre que le fait de tourner la page va nécessairement produire un vide à l'emplacement où cette page figurait avant de pivoter. Pour combler ce trou, nous plaçons, à l'arrière-plan, le symbole `pageArriere_mc`. Afin que la nouvelle page contenant l'image n'apparaisse pas brutalement, nous utilisons un fondu en alpha.

Il est possible d'afficher plus tôt le contenu des pages en multipliant les symboles `pageDroite` dans le livre, d'une part, et en multipliant d'autant les instructions `gotoAndStop` dans le programme.

Si la condition pour réaliser la rotation de la page n'est pas vérifiée (`if(i<=nombreDePages)`), ou si le nombre de pages visitées a atteint la limite autorisée, alors `else`, une autre condition, est exécutée :

```
else {
      //animation de fermeture
      livre_mc.pageArriere_mc.visible=false;

TweenMax.to(livre_mc.pageDroite_mc,1,{rotationY:180,ease:Strong.easeOut});
      TweenMax.to(livre_mc,4,{rotationZ:360, x:1000, z:-500, alpha:0,
      ➥ rotationY:360, delay:1, ease:Strong.easeInOut});
      livre_mc.flecheDroite_btn.visible=false;
      tweenPage.addEventListener(TweenEvent.COMPLETE, tweenPageFini);
   }
```

Lorsque les images de la page de droite sont toutes visitées, la page arrière est d'abord masquée (`visible=false`) car il ne reste plus de page à tourner. La scène devient visible à l'arrière-plan du livre.

Une fois le livre refermé, deux transitions sont exécutées. La première fait pivoter la dernière page. La seconde fait s'envoler le livre, mais avec un délai d'une seconde. Ceci permet au livre de se refermer avant de le voir s'envoler.

Les mouvements du livre sont composés d'une rotation sur les axes Z (`rotationZ:360`) et Y (`rotationY:360`), puis d'un déplacement en X et Z. La valeur utilisée pour le déplacement du livre, en Z, est négative (`z:-500`). Cela signifie que l'objet quitte la scène tout en se rapprochant de l'écran.

Puis, nous masquons le bouton flèche pour éviter que l'utilisateur ne relance l'action. Nous signalons aussi la fin du programme.

Enfin, pour d'optimiser les ressources de l'utilisateur, nous invoquons une fonction qui interrompt, à la fin de la transition, le gestionnaire `ENTER_FRAME`, désormais inutile (`tweenPageFini`).

Dans les quatrième et dernière parties, pour terminer, nous avons ajouté une fonction qui masque le verso des pages tournées :

```
//------------------------- masquer le verso des pages qui tournent
addEventListener(Event.ENTER_FRAME,masquerVersoPage);
function masquerVersoPage (evt:Event) {
   if (livre_mc.pageDroite_mc.rotationY>90) {
      livre_mc.pageDroite_mc.gotoAndStop(6);
   }
}
```

Nous spécifions que, dès que la page tournée atteint une rotation Y de 90°, c'est-à-dire dès qu'elle apparaît sur la tranche et que l'on ne perçoit plus son contenu, au recto ni au verso, nous déplaçons, avec un gestionnaire de type `ENTER_FRAME`, la tête de lecture de l'objet `page` à l'image 6, qui représente une page vide. Sans cette fonction, nous pourrions voir des deux côtés du symbole la même image, mais inversée. En affichant ponctuellement, juste le temps de la rotation, l'image 6, nous donnons l'illusion de pages indépendantes les unes des autres.

Pour compléter, l'action appelée par le gestionnaire `ENTER-FRAME` fait référence à une nouvelle image du symbole `pageDroite` afin de compenser le fait que le moteur 3D affiche les contenus sur les deux faces simultanément. Lorsque la page bascule à 90°, il nous suffit donc de modifier l'image affichée dans ce clip pour donner l'illusion que le verso se distingue du recto. Ainsi, pour créer un livre recto-verso, vous pouvez ajouter dans le symbole `pageDroite_mc` autant de pages verso que voulues, puis les appeler depuis le gestionnaire `ENTER_FRAME` en fonction de `i`, à travers la même méthode `gotoAndStop()`. Vous obtenez :

```
livre_mc.pageDroite_mc.gotoAndStop(i+nombreDePages);
```

Pensez dans ce cas à adapter aussi le calcul de la variable `nombreDePages`, en fonction du nombre de pages verso. Par exemple :

```
Var nombreDePages:int = (livre_mc.pageDroite_mc.totalFrames-1) / 2
```

Réaliser un livre interactif avec InDesign. Des systèmes très réalistes de livres interactifs préprogrammés (composants `pageFlip`) existent sur le Web et sont facilement accessibles. Dans la suite Adobe, par exemple, InDesign CS4 intègre cette solution. Vous pouvez exporter tout type de mise en pages en livre Flash interactif au format SWF. Il ne reste plus, ensuite, qu'à lier le document à votre site avec un chargeur programmé en ActionScript, comme nous l'avons vu précédemment dans le chapitre sur les galeries d'images. Pour exporter une mise en pages sous la forme d'un livre réaliste au format SWF, depuis InDesign, faites Fichier > Exporter. Dans les options d'exportation, choisissez SWF. Pour aller plus loin dans la gestion de documents mis en pages, reportez-vous au Chapitre 15, section "Gérer le PDF".

Lisser les images pour la 3D. L'affichage 3D peut générer un crénelage sur les images lorsqu'elles sont projetées dans un espace 3D ou simplement inclinées. Pour lisser les images, depuis la fenêtre de Bibliothèque, sur chaque image, faites clic-droit > Propriétés. Puis, dans la fenêtre de dialogue, activez l'option Autoriser le lissage. Vous pouvez procéder à la même action en programmation. Reportez-vous à la section "Lissage des images bitmap", au Chapitre 11, pour connaître les solutions de lissage en programmation.

Le langage

3D avec la classe `Tween`

Les propriétés 3D natives de Flash peuvent être gérées avec la classe `Tween`. Pour un objet `clip_mc` animé, par exemple, avec les propriétés `z` et `rotationX`, vous obtenez ceci :

```
import fl.transitions.Tween;
import fl.transitions.easing.*;
var monTween:Tween;
monTween = new Tween(clip_mc, "z", Elastic.easeInOut, 0, 500, 3, true);
var monTweenRx:Tween;
monTweenRx = new Tween(clip_mc, "rotationX", Elastic.easeInOut, 0, 500, 3, true);
```

3D avec la classe `Caurina`

Les propriétés 3D natives de Flash peuvent être gérées avec la classe `Caurina` (disponible sur **http://code.google.com/p/tweener/downloads/list**). Pour un objet `clip_mc` animé avec, par exemple, les propriétés `z` et `rotationX`, vous obtenez ceci :

```
import caurina.transitions.Tweener;
Tweener.addTween(clip_mc, {x:300, y:300, z:330, rotationX:90, rotationY:0,
➥ rotationZ:0, time:5, transition:"easeinoutexpo"});
```

3D avec la classe `TweenMax`

Les propriétés 3D natives de Flash peuvent être gérées avec la classe TweenMax. Pour un objet `clip_mc` animé avec, par exemple, les propriétés `z` et `rotationX`, vous obtenez ceci :

```
import gs.TweenMax;
import gs.easing.*;
TweenMax.to(clip_mc, 2, {rotationX:90,rotationY:90,z:500,delay:1,
➥ ease:Bounce.easeOut});
```

À retenir

- Flash autorise la gestion de la 3D y compris à travers les interpolations des classes `TweenMax`, `Caurina` et `Tween`. Il suffit d'y introduire les propriétés 3D `x`, `y`, `z`, `rotationX`, `rotationY` et `rotationZ`.
- Pour animer un objet en 3D, celui-ci doit être un MovieClip.
- La gestion de la 3D dans Flash est localisée sur chaque objet. Il faut donc modifier l'ordre d'apparition des objets dans la liste d'affichage lorsque leurs mouvements se croisent.
- La position du centre de chaque symbole animé en 3D est déterminant pour la définition des mouvements dans l'espace.

Carrousel d'images 3D

Un carrousel d'images 3D est une suite d'images que l'on peut faire défiler dans un espace 3D. Les images qui passent au centre sont frontales. Celles qui restent en retrait de la zone centrale se repositionnent dans l'espace 3D par une transition animée. Les carrousels d'images peuvent être linéaires, circulaires ou épouser d'autres formes. L'exemple le plus illustre de ce type de présentation est le système de navigation intitulé *CoverFlow* et développé par Apple. On le retrouve principalement sur le système Mac OS X, dans l'iPhone, iTunes et de nombreux services associés à la marque Apple (voir Figure 9.9).

Figure 9.9
Apple CoverFlow.

Dans cette section, nous allons voir comment faire défiler des images sur une bande horizontale. Mais nous présentons surtout comment définir un repositionnement 3D automatique, en fonction de la position de chaque image sur l'axe des abscisses X.

Pour cet exemple, nous proposons deux documents codés différemment, selon que vous préférez un codage simple à partir d'objets placés manuellement dans le scénario ou un codage plus adapté à un interfaçage dynamique (pour lier la présentation à un fichier XML, par exemple).

Version simplifiée

Exemples > ch9_3DNative_2.fla

En publiant le premier document, nommé "ch9_3DNative_2.fla", deux flèches donnent accès au défilement d'une suite d'images. À chaque clic, les images se repositionnement dynamiquement dans l'espace 3D en exerçant une rotation sur l'axe Y (voir Figure 9.10).

Figure 9.10
Aperçu du document publié.

Dans la scène principale du document, une série de MovieClip est répartie vers les calques (voir Figure 9.11).

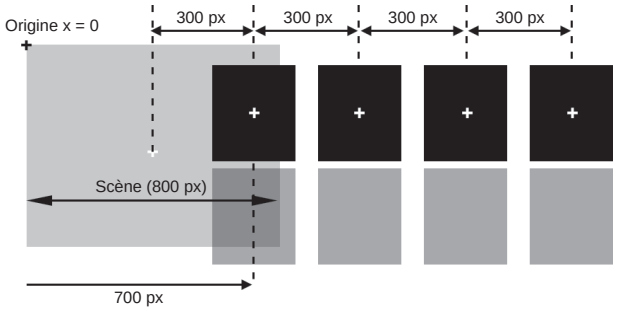
Chaque symbole comporte une image doublée verticalement pour simuler un reflet. Le premier d'entre eux est situé précisément à 700 pixels à droite. Dans le code, nous utilisons un pas d'incrémentation de 300 pixels pour définir le repositionnement en X de ces images. La scène, elle, mesure 800 pixels et son centre est donc situé à 400 pixels (voir Figure 9.12).

En plaçant la première image à une valeur multiple du pas d'incrémentation (400 pixels + 300 pixels), nous permettons à l'ensemble des images d'apparaître plus tard, au centre de la scène, frontalement.

Les images sont par ailleurs collées les unes aux autres, et pourraient même se chevaucher, car la rotation Y que nous ajoutons sur chaque image en programmation, réduit leur espace nécessaire en largeur. De ce fait, une marge due à l'inclinaison apparaîtra lors de la publication.

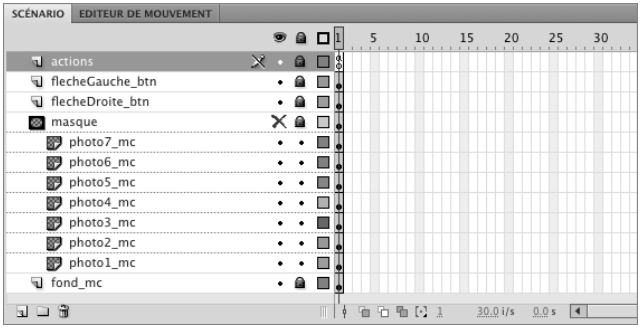
Figure 9.11
Aperçu de la scène principale.

Figure 9.12
Calcul du positionnement des images.



Les flèches gauche et droite situées de part et d'autre du document risquent d'être cachées par le défilement des images. Pour contourner ce problème, dans le scénario, un masque qui affecte toutes les images a été appliqué et réduit la partie visible de la zone de défilement (voir Figure 9.13).

Figure 9.13
Aperçu du scénario de la scène principale.



À l'intérieur de chaque MovieClip d'image, le visuel est en fait contenu dans un second symbole. Ce dernier est dupliqué et placé symétriquement en miroir bas de façon à obtenir un effet de reflet. Sur ce deuxième symbole, nous avons appliqué un alpha à 40 %.

Tous les symboles de la scène possèdent chacun leur propre nom d'occurrence. Le code placé dans la fenêtre du calque `Actions` affiche ceci :

```
//------------------ initialisation
import gs.TweenMax;
import gs.easing.*;
import gs.events.*;

//------------------ actions

// flèche droite
flecheDroite_btn.addEventListener(MouseEvent.CLICK,defilementDroite);
function defilementDroite (evt:MouseEvent) {
   TweenMax.to(photo1_mc, 2, {x:photo1_mc.x-300, delay:0, ease:Back.easeInOut});
   TweenMax.to(photo2_mc, 2, {x:photo2_mc.x-300, delay:0, ease:Back.easeInOut});
   TweenMax.to(photo3_mc, 2, {x:photo3_mc.x-300, delay:0, ease:Back.easeInOut});
   TweenMax.to(photo4_mc, 2, {x:photo4_mc.x-300, delay:0, ease:Back.easeInOut});
   TweenMax.to(photo5_mc, 2, {x:photo5_mc.x-300, delay:0, ease:Back.easeInOut});
   TweenMax.to(photo6_mc, 2, {x:photo6_mc.x-300, delay:0, ease:Back.easeInOut});
   TweenMax.to(photo7_mc, 2, {x:photo7_mc.x-300, delay:0, ease:Back.easeInOut});
}
// flèche gauche
flecheGauche_btn.addEventListener(MouseEvent.CLICK,defilementGauche);
function defilementGauche (evt:MouseEvent) {
   TweenMax.to(photo1_mc, 2, {x:photo1_mc.x+300,delay:0, ease:Back.easeInOut});
   TweenMax.to(photo2_mc, 2, {x:photo2_mc.x+300,delay:0, ease:Back.easeInOut});
   TweenMax.to(photo3_mc, 2, {x:photo3_mc.x+300,delay:0, ease:Back.easeInOut});
   TweenMax.to(photo4_mc, 2, {x:photo4_mc.x+300,delay:0, ease:Back.easeInOut});
   TweenMax.to(photo5_mc, 2, {x:photo5_mc.x+300,delay:0, ease:Back.easeInOut});
   TweenMax.to(photo6_mc, 2, {x:photo6_mc.x+300,delay:0, ease:Back.easeInOut});
   TweenMax.to(photo7_mc, 2, {x:photo7_mc.x+300,delay:0, ease:Back.easeInOut});
}

// rotationY
var largeurScene:Number=stage.stageWidth;
var moitieScene:Number=largeurScene/2;

addEventListener(Event.ENTER_FRAME,position3DAuto);
function position3DAuto (evt:Event) {
   if (photo1_mc.x<largeurScene+photo1_mc.width && photo1_mc.x>-(photo1_mc.width)) {
   photo1_mc.rotationY=Math.ceil((photo1_mc.x-moitieScene)/4);
   }
   if (photo2_mc.x<largeurScene+photo2_mc.width && photo2_mc.x>-(photo2_mc.width)) {
   photo2_mc.rotationY=Math.ceil((photo2_mc.x-moitieScene)/4);
   }
```

```
    if (photo3_mc.x<largeurScene+photo3_mc.width && photo3_mc.x>-(photo3_mc.width)) {
    photo3_mc.rotationY=Math.ceil((photo3_mc.x-moitieScene)/4);
    }
    if (photo4_mc.x<largeurScene+photo4_mc.width && photo4_mc.x>-(photo4_mc.width)) {
    photo4_mc.rotationY=Math.ceil((photo4_mc.x-moitieScene)/4);
    }
    if (photo5_mc.x<largeurScene+photo5_mc.width && photo5_mc.x>-(photo5_mc.width)) {
    photo5_mc.rotationY=Math.ceil((photo5_mc.x-moitieScene)/4);
    }
    if (photo6_mc.x<largeurScene+photo6_mc.width && photo6_mc.x>-(photo6_mc.width)) {
    photo6_mc.rotationY=Math.ceil((photo6_mc.x-moitieScene)/4);
    }
    if (photo7_mc.x<largeurScene+photo7_mc.width && photo7_mc.x>-(photo7_mc.width)) {
    photo7_mc.rotationY=Math.ceil((photo7_mc.x-moitieScene)/4);
    }
}
```

Ce code, conçu pour une gestion simple des contenus à partir d'objets placés directement dans le scénario, est structuré en trois parties. La première partie appelle les classes utilisées pour les transitions `TweenMax`. La deuxième partie gère le défilement horizontal des vignettes. La troisième partie assure la rotation dans l'espace 3D selon la position courante de chaque image dans la scène.

Le déplacement des images en X, engagé avec les flèches, utilise des interpolations de type `TweenMax` :

```
// flèche droite
flecheDroite_btn.addEventListener(MouseEvent.CLICK,defilementDroite);
function defilementDroite (evt:MouseEvent) {
   TweenMax.to(photo1_mc, 2, {x:photo1_mc.x-300, delay:0, ease:Back.easeInOut});
}
```

Le principe est décliné pour chaque image de la scène et sur les deux flèches. Nous inversons simplement les valeurs de positionnement de +300 à –300.

Ensuite, nous organisons le système de rotation automatique avec un gestionnaire de type `ENTER_FRAME`, afin que l'événement soit recalculé perpétuellement :

```
// rotationY
var largeurScene:Number=stage.stageWidth;
var moitieScene:Number=largeurScene/2;

addEventListener(Event.ENTER_FRAME,position3DAuto);
function position3DAuto (evt:Event) {
   if (photo1_mc.x<largeurScene+photo1_mc.width && photo1_mc.x>-(photo1_mc.width)) {
   photo1_mc.rotationY=Math.ceil((photo1_mc.x-moitieScene)/4);
   }
}
```

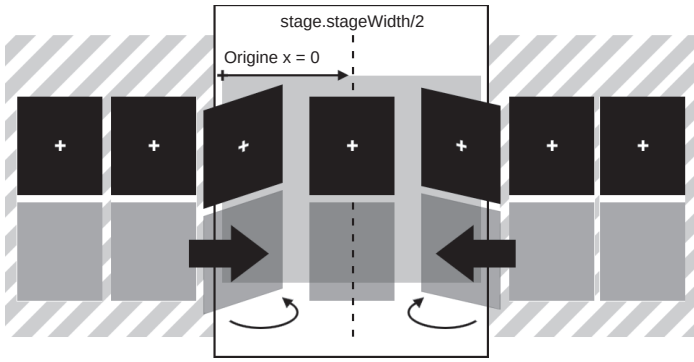
D'abord, nous initialisons deux valeurs. La première enregistre la largeur de la scène (`stage.stageWidth`). La seconde enregistre la position du centre de la scène, en divisant la première par deux (`largeurScene/2`). Ces valeurs sont utilisées dans la fonction

`position3DAuto`. Dans le bloc d'instructions de la fonction, pour chaque image, une condition est vérifiée :

```
if (photo1_mc.x<largeurScene+photo1_mc.width && photo1_mc.x>-(photo1_mc.width)) {
   photo1_mc.rotationY=Math.ceil((photo1_mc.x-moitieScene)/4);
}
```

Si la position courante de l'image est inférieure à la largeur de la scène et sa largeur cumulée – donc si l'image entre dans le champ par la droite–, alors le programme calcule sa rotation en Y. Simultanément, une seconde condition est vérifiée. Elle se distingue de la première par les deux esperluettes (`&&`) : si, la position courante de l'image est également supérieure à sa largeur négative, c'est-à-dire, dès que l'image entre dans le champ par la gauche, alors, la même instruction que pour la première condition est exécutée (voir Figure 9.14).

Figure 9.14
Mécanisme de rotation en Y.



L'instruction désignée est la suivante :

```
photo1_mc.rotationY=Math.ceil((photo1_mc.x-moitieScene)/4);
```

Nous définissons la rotation Y de l'objet en fonction de sa position courante en X. Plus précisément, nous reprenons la position en X de l'objet moins la demie largeur de la scène, de sorte que la rotation soit neutralisée au centre de la scène, et non à l'extrémité gauche où `x` vaut 0. Pour définir un angle de rotation entier, nous arrondissons la valeur grâce à l'utilisation de la méthode `Math.ceil()`.

Ce principe est appliqué à chacune des images présentes dans la scène.

Version dynamique

Afin de vous permettre d'exploiter plus facilement le carrousel avec des contenus externalisés, nous avons optimisé la gestion de ce programme en utilisant une structure composée à partir d'une boucle `for` et une autre avec une structure `for each… in`.

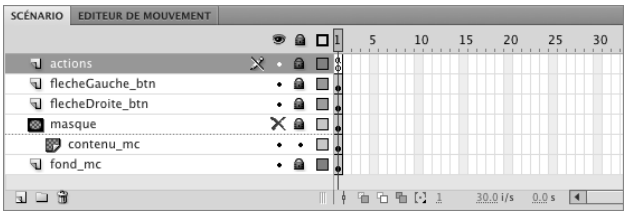
Exemples > ch9_3DNative_2b.fla
Exemples > ch9_3DNative_2c.fla

Dans le premier document, nous retrouvons quasiment la même structure que dans le fichier initial, à la différence que l'ensemble des vignettes est désormais rassemblé dans un conteneur

de type MovieClip que vous pourrez substituer éventuellement par un conteneur généré dynamiquement.

Cette modification structurelle, en ajoutant un conteneur, permet d'appliquer dynamiquement le même comportement à l'ensemble des éléments qu'il contient. Dans le scénario de la scène principale, le symbole `contenu_mc` rassemble tous les éléments (voir Figure 9.15).

Figure 9.15
Aperçu du scénario de la scène principale.



Dans la fenêtre Actions, nous pouvons lire le code suivant :

```
//------------------- initialisation
import gs.TweenMax;
import gs.easing.*;
import gs.events.*;

var positionPhoto1:Number=(contenu_mc.getChildAt(0)).x;
var pasIncrementation:Number=300;

//------------------- actions

// flèche droite
flecheDroite_btn.addEventListener(MouseEvent.CLICK,defilementDroite);
function defilementDroite (evt:MouseEvent) {
   for (var i:Number=0;i<contenu_mc.numChildren;i++) {
      TweenMax.to(
         (contenu_mc.getChildAt(i)), 2,
         {
            x:(contenu_mc.getChildAt(i)).x-pasIncrementation,
            delay:0,
            ease:Back.easeInOut
         }
         );
   }
}

// flèche gauche
flecheGauche_btn.addEventListener(MouseEvent.CLICK,defilementGauche);
function defilementGauche (evt:MouseEvent) {
   for (var j:Number=0;j<contenu_mc.numChildren;j++) {
      TweenMax.to(
         (contenu_mc.getChildAt(j)), 2,
         {
            x:(contenu_mc.getChildAt(j)).x+pasIncrementation,
            delay:0,
            ease:Back.easeInOut
         }
```

```
            );
        }
    }

    // rotationY
    var largeurScene:Number=stage.stageWidth;
    var moitieScene:Number=largeurScene/2;

    contenu_mc.addEventListener(Event.ENTER_FRAME,position3DAuto);
    function position3DAuto (evt:Event) {
       for (var k:Number=0;k<contenu_mc.numChildren;k++) {
          if
    ((contenu_mc.getChildAt(k)).x<largeurScene+(contenu_mc.getChildAt(k)).width-
    positionPhoto1 && (contenu_mc.getChildAt(k)).x>-largeurScene) {

    (contenu_mc.getChildAt(k)).rotationY=Math.ceil(((contenu_mc.getChildAt(k)).
    x-moitieScene-positionPhoto1)/4);
          }
       }
    }

    // ciblage dynamique de contenu
    contenu_mc.addEventListener(MouseEvent.CLICK,actionPhotos);
    function actionPhotos (evt:MouseEvent) {
       trace(evt.target.name);
       TweenMax.to(evt.target,2, {rotationZ:evt.target.rotationZ+360, delay:0,
    ease:Back.easeInOut});
    }
```

Dans ce programme, nous initialisons d'abord deux valeurs que nous utiliserons dans le script. La première définit la position actuelle du premier symbole image dans `contenu_mc`. La méthode `getChildAt(0)` permet de sélectionner l'élément qui apparaît en premier dans la liste d'affichage de ce conteneur. Vous pouvez en vérifier le nom en ajoutant la propriété `name` à une action `trace` :

```
trace(contenu_mc.getChildAt(0).name); //photo1_mc
```

Dans les actions, le code est aéré sur plusieurs lignes afin de mieux distinguer chacune des propriétés utilisées dans l'interpolation `TweenMax`. Cela n'a pas d'incidence sur l'exécution du programme.

Nous remplaçons ici la répétition des instructions par une boucle `for` qui ne conserve qu'une seule ligne d'instruction. Il existe autant d'itérations que d'images. C'est la boucle qui, en fonction de ses paramètres, va multiplier et adapter les lignes de commande pour chaque contenu :

```
for (var i:Number=0;i<contenu_mc.numChildren;i++) {
   TweenMax.to(
     (contenu_mc.getChildAt(i)), 2, {x:(contenu_mc.getChildAt(i)).
     x-pasIncrementation, delay:0, ease:Back.easeInOut });
}
```

La valeur qui permet d'adapter l'instruction est véhiculée par `i`. Nous indiquons, au travers du constructeur `for`, d'initialiser `i` à 0. Puis, tant que cette valeur reste inférieure au nombre d'enfants (symboles ou objets) disponibles dans `contenu_mc`, nous incrémentons cette valeur. Ainsi, la boucle répète l'ensemble des instructions qu'elle contient, entre ses accolades,

autant de fois qu'il y aura d'autres images. À chaque itération, la valeur de `i` est augmentée (`i++`) si bien que les instructions qui utilisent cette valeur comme paramètre peuvent maintenant atteindre tour à tour chaque objet.

Dans l'interpolation `TweenMax`, nous ciblons chaque objet individuellement à l'aide de la méthode `getChildAt()` (avec `(contenu_mc.getChildAt(i))`), déjà invoquée pour la variable, plus haut. Cependant, il n'est pas possible d'appliquer directement la propriété `x` à la méthode `getChildAt()`. Des parenthèses sont donc ajoutées ici, pour contenir la première instruction qui cible d'abord l'objet, avant, seulement, de pouvoir modifier une des propriétés de cet objet. Cette méthode est déclinée pour l'ensemble des actions du programme.

En fin de code, une dernière action est ajoutée au conteneur principal. Elle permet de cibler individuellement chacun des éléments de ce conteneur en usant de la propriété `target` et de lui appliquer une rotation sur l'axe Z :

```
// ciblage dynamique de contenu
contenu_mc.addEventListener(MouseEvent.CLICK,actionPhotos);
function actionPhotos (evt:MouseEvent) {
   trace(evt.target.name);
   TweenMax.to(evt.target,2, {rotationZ:evt.target.rotationZ+360, delay:0,
   ➥ ease:Back.easeInOut});
}
```

Pour en savoir plus sur le développement dynamique d'interfaces, reportez-vous au manuel d'Anne Tasso (Apprendre à programmer en ActionScript 3, éd. Eyrolles).

Une déclinaison de ce document est proposée, avec la structure `for each… in`, dans le fichier "ch9_3DNative_2c.fla". Dans ce document, nous avons remplacé la boucle for avec une boucle de type `for each… in`. Ce type de boucle possède la particularité de désigner l'ensemble des objets d'un conteneur simultanément. Ce qui évite d'avoir à les cibler individuellement. C'est donc encore plus optimisé. Cela donne :

```
// flèche droite
flecheDroite_btn.addEventListener(MouseEvent.CLICK,defilementDroite);
function defilementDroite (evt:MouseEvent) {
   for each (var mc:MovieClip in contenu_mc){
      TweenMax.to(mc, 2,{x:mc.x-pasIncrementation, delay:0, ease:Back.easeInOut });
   }
}

// flèche gauche
flecheGauche_btn.addEventListener(MouseEvent.CLICK,defilementGauche);
function defilementGauche (evt:MouseEvent) {
   for each (var mc:MovieClip in contenu_mc){
      TweenMax.to(mc, 2,{x:mc.x+pasIncrementation, delay:0,   ease:Back.easeInOut   });
   }
}

// rotationY
var largeurScene:Number=stage.stageWidth;
var moitieScene:Number=largeurScene/2;
```

```
contenu_mc.addEventListener(Event.ENTER_FRAME,position3DAuto);
function position3DAuto (evt:Event) {
   for each (var mc:MovieClip in contenu_mc){
      if (mc.x<largeurScene+mc.width-positionPhoto1 && mc.x>-largeurScene) {
         mc.rotationY=Math.ceil((mc.x-moitieScene-positionPhoto1)/4);
      }
   }
}
```

À retenir

- Pour centrer dans la scène les effets ou les objets, nous intégrons, dans le calcul des valeurs, la position du centre de la scène avec `stage.stageWidth/2`.
- La boucle `for each… in` permet de cibler directement l'ensemble des objets d'un conteneur.

Mur d'images 3D

Un mur d'images est une mosaïque sur laquelle on clique pour afficher une image en détail. Lorsque le pointeur survole l'écran, le mur d'images oscille proportionnellement dans un sens ou dans l'autre, verticalement et horizontalement.

Dans cet exemple, nous utilisons des symboles placés sur la scène dont nous contrôlons l'inclinaison en fonction de la position du pointeur à l'écran. En balayant la scène, le mur d'image bouge. Nous ajoutons une interactivité sur les images à l'aide de la propriété `target`, pour, par exemple, zoomer individuellement sur chaque image et matérialiser un effet de "rollOver" avec un filtre dynamique (voir Figures 9.16 à 9.18).

Figure 9.16
Aperçu du document à la publication, pointeur sur le côté.

Figure 9.17
Aperçu du document à la publication, image survolée.

Figure 9.18
Aperçu du document à la publication, image cliquée et zoomée.

Exemples > ch9_3DNative_3.fla

Le document "ch9_3DNative_3.fla" affiche une mosaïque nommée `contenu_mc`, à l'intérieur de laquelle sont répartis différents MovieClip. Chacun de ces MovieClip contient sa propre image et possède un nom d'occurrence (voir Figure 9.19).

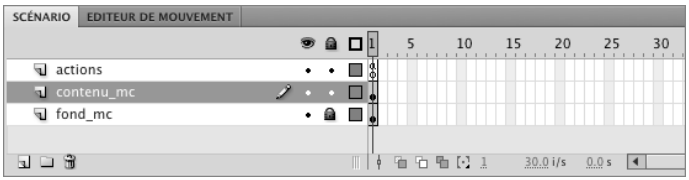
Figure 9.19
Aperçu de la scène principale.

Chaque symbole est déjà réduit à 50 % de ses dimensions originales. Cela permet d'activer une fonction de zoom sans perdre sur la netteté de l'image. Le zoom leur affectera seulement leur échelle normale.

Pour améliorer également le rendu des images, dans un contexte en perpétuel mouvement, nous avons activé, depuis la fenêtre Bibliothèque, dans les propriétés de chaque image (clic-droit > Propriétés), l'option Autoriser le lissage.

Le scénario ne laisse apparaître que le symbole `contenu_mc` (voir Figure 9.20).

Figure 9.20
Aperçu du scénario de la scène principale.



Dans la fenêtre Actions, nous pouvons lire le code suivant :

```
//------------------- initialisation
import flash.filters.*;
var haloIn:GlowFilter=new GlowFilter(0xffffff, 1, 2, 2, 3, 255, false, false);
var haloOut:GlowFilter=new GlowFilter(0xffffff, 1, 0, 0, 3, 255, false, false);

import gs.TweenMax;
import gs.easing.*;
import gs.events.*;

var demieScene:Number=stage.stageWidth/2;
var hauteurScene:Number=stage.stageHeight/2;
var Tween3D:TweenMax;
```

```
var nomImageActive:String;

var positionInitX:Array=new Array();
var positionInitY:Array=new Array();
for (var i:Number=0; i<contenu_mc.numChildren; i++) {
   positionInitX.push(contenu_mc.getChildAt(i).x)
   positionInitY.push(contenu_mc.getChildAt(i).y)
}

//------------------ actions

// mouvement 3D
contenu_mc.addEventListener(Event.ENTER_FRAME, murImages);
function murImages (evt:Event) {
   contenu_mc.rotationY=(mouseX-demieScene)*0.1;
   contenu_mc.rotationX=(mouseY-hauteurScene)*-0.1;
}

// interactivité zoom
contenu_mc.addEventListener(MouseEvent.CLICK,zoom);
function zoom (evt:MouseEvent) {
   Tween3D=TweenMax.to(contenu_mc.photo1_mc, 0.3, {z:0, x:positionInitX[0],
   ➥y:positionInitY[0], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
   TweenMax.to(contenu_mc.photo2_mc, 0.3, {z:0, x:positionInitX[1],
   ➥ y:positionInitY[1], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
   TweenMax.to(contenu_mc.photo3_mc, 0.3, {z:0, x:positionInitX[2],
   ➥ y:positionInitY[2], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
   TweenMax.to(contenu_mc.photo4_mc, 0.3, {z:0, x:positionInitX[3],
   ➥ y:positionInitY[3], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
   TweenMax.to(contenu_mc.photo5_mc, 0.3, {z:0, x:positionInitX[4],
   ➥ y:positionInitY[4], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
   TweenMax.to(contenu_mc.photo6_mc, 0.3, {z:0, x:positionInitX[5],
   ➥ y:positionInitY[5], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
   TweenMax.to(contenu_mc.photo7_mc, 0.3, {z:0, x:positionInitX[6],
   ➥ y:positionInitY[6], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
   TweenMax.to(contenu_mc.photo8_mc, 0.3, {z:0, x:positionInitX[7],
   ➥ y:positionInitY[7], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
   TweenMax.to(contenu_mc.photo9_mc, 0.3, {z:0, x:positionInitX[8],
   ➥ y:positionInitY[8], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
   //
   Tween3D.addEventListener(TweenEvent.COMPLETE,restaureFini);
   function restaureFini (yo:TweenEvent) {
      contenu_mc.addChild(MovieClip(evt.target));
   }
   if (nomImageActive!=evt.target.name) {
      nomImageActive=evt.target.name
      //
      TweenMax.to(evt.target,2, {z:-100, x:0, y:0, scaleX:1.2, scaleY:1.2,
      ➥ delay:0.3, ease:Elastic.easeOut});
   } else {
      nomImageActive="";
   }
}

// rollOvers
contenu_mc.addEventListener(MouseEvent.MOUSE_OVER,over);
```

```
function over (evt:MouseEvent) {
   evt.target.filters=[haloIn];
}
contenu_mc.addEventListener(MouseEvent.MOUSE_OUT,out);
function out (evt:MouseEvent) {
   evt.target.filters=[haloOut];
}
```

Le code est composé de quatre parties. L'initialisation, les actions de positionnement du mur dans l'espace 3D, la gestion de l'interactivité (zoom) sur les images et la gestion de l'effet rollOver.

Avant de détailler chaque étape, nous devons comprendre le processus employé. En publiant le document, d'abord, l'ensemble nommé `conteneur_mc` oscille en fonction de la position du pointeur dans la scène. Puis, au passage de la souris sur les symboles qu'il contient, ceux-ci réagissent avec un filtre dynamique. En cliquant sur un symbole, celui-ci s'agrandit. En cliquant à nouveau dessus, il revient à sa position normale. Mais, si nous cliquons directement sur un autre symbole, simultanément, la nouvelle image s'agrandit, tandis que la précédente revient en position initiale.

Dans la première partie, nous commençons par importer les classes requises, dont `filters`, qui permettent, plus bas, de gérer l'effet rollOver (voir Chapitre 2). Nous commençons par instancier les deux effets, l'effet survolé et l'effet inactif. Lorsque le pointeur survolera l'image, le premier sera activé. Le second sera appelé en sortant de l'image :

```
//------------------ initialisation
import flash.filters.*;
var haloIn:GlowFilter=new GlowFilter(0xffffff, 1, 2, 2, 3, 255, false, false);
var haloOut:GlowFilter=new GlowFilter(0xffffff, 1, 0, 0, 3, 255, false, false);
```

Nous importons ensuite les classes des transitions `TweenMax` :

```
import gs.TweenMax;
import gs.easing.*;
import gs.events.*;
```

Puis, nous définissons quelques valeurs :

```
var demieScene:Number=stage.stageWidth/2;
var hauteurScene:Number=stage.stageHeight/2;
var Tween3D:TweenMax;
var nomImageActive:String;
```

Les deux premières valeurs servent à définir les rotations par rapport au centre de la scène, sur le même principe que celui énoncé dans la section précédente. Ensuite, nous typons une interpolation `TweenMax` afin de pouvoir enchaîner des actions à l'issue d'une interpolation.

Enfin, nous créons une variable chaîne de caractères afin de capturer, à chaque clic, le nom de l'image cliquée (`nomImageActive`), de manière à différencier son comportement, selon qu'elle ait déjà été zoomée ou non.

Nous définissons ensuite deux tableaux qui enregistrent respectivement les positions de départ des images en X et Y. Nous utiliserons, plus loin dans le code, ces valeurs afin de restaurer les images à leurs positions initiales :

```
var positionInitX:Array=new Array();
var positionInitY:Array=new Array();
```

```
for (var i:Number=0; i<contenu_mc.numChildren; i++) {
   positionInitX.push(contenu_mc.getChildAt(i).x)
   positionInitY.push(contenu_mc.getChildAt(i).y)
}
```

La création de chaque tableau est lancée avec la classe `Array`. Une boucle `for` ajoute, grâce à la méthode `push()`, la position courante, en X et en Y, de chaque objet en fonction de son ordre d'affichage `(contenu_mc.getChildAt(i))`. La boucle exécute autant d'itération qu'il y a d'enfant dans le symbole `contenu_mc (contenu_mc.numChildren)`.

Dans les actions, nous commençons par définir le mouvement global du conteneur d'images :

```
//------------------ actions

// mouvement 3D
contenu_mc.addEventListener(Event.ENTER_FRAME, murImages);
function murImages (evt:Event) {
   contenu_mc.rotationY=(mouseX-demieScene)*0.1;
   contenu_mc.rotationX=(mouseY-hauteurScene)*-0.1;
}
```

Un écouteur est attaché au contenu et fait pivoter, en continu, l'objet sur l'axe Y (`rotationY`) et sur l'axe X (`rotationX`). La rotation sur l'axe Y est déterminée en fonction de la position du pointeur sur X. Celle de l'axe X, en fonction de la position du pointeur sur Y. Mais, comme vu précédemment, nous retranchons la demi-largeur et hauteur de la scène, afin de neutraliser les rotations lorsque le pointeur survole le milieu de l'écran (quand la position vaut 0, la `rotationX` et `rotationY` vaut 0).

D'autre part, la position du pointeur définie en X et en Y se mesure en centaines de pixels. La rotation, elle, se mesure en dizaines de degrés. Nous réduisons donc la valeur obtenue en la multipliant par `0.1` (équivaut à diviser par `10`), ceci afin d'éviter un mouvement trop prononcé.

Nous ajustons également la polarité du mouvement, avec un signe moins, selon que le sens que l'on veut affecter à l'inclinaison. Le signe moins inverse l'inclinaison par rapport à la position du pointeur. L'absence de signe conserve une inclinaison qui accompagne le mouvement du pointeur.

À la suite, nous créons l'interactivité relative à la fonction `zoom`, appliquée à chaque image :

```
// interactivité zoom
contenu_mc.addEventListener(MouseEvent.CLICK,zoom);
function zoom (evt:MouseEvent) {
   Tween3D=TweenMax.to(contenu_mc.photo1_mc, 0.3, {z:0, x:positionInitX[0],
   ➥ y:positionInitY[0], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
   TweenMax.to(contenu_mc.photo2_mc, 0.3, {z:0, x:positionInitX[1],
   ➥ y:positionInitY[1], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
   TweenMax.to(contenu_mc.photo3_mc, 0.3, {z:0, x:positionInitX[2],
   ➥ y:positionInitY[2], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
   TweenMax.to(contenu_mc.photo4_mc, 0.3, {z:0, x:positionInitX[3],
   ➥ y:positionInitY[3], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
   TweenMax.to(contenu_mc.photo5_mc, 0.3, {z:0, x:positionInitX[4],
   ➥ y:positionInitY[4], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
```

```
        TweenMax.to(contenu_mc.photo6_mc, 0.3, {z:0, x:positionInitX[5],
        ➥ y:positionInitY[5], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
        TweenMax.to(contenu_mc.photo7_mc, 0.3, {z:0, x:positionInitX[6],
        ➥ y:positionInitY[6], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
        TweenMax.to(contenu_mc.photo8_mc, 0.3, {z:0, x:positionInitX[7],
        ➥ y:positionInitY[7], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
        TweenMax.to(contenu_mc.photo9_mc, 0.3, {z:0, x:positionInitX[8],
        ➥ y:positionInitY[8], scaleX:0.5, scaleY:0.5, delay:0 ,ease:Strong.easeInOut});
        //
        Tween3D.addEventListener(TweenEvent.COMPLETE,restaureFini);
        function restaureFini (yo:TweenEvent) {
           contenu_mc.addChild(MovieClip(evt.target));
        }
        // autres actions
    }
```

Dans un premier temps, nous appliquons un écouteur qui détecte tout clic capturé par l'objet `contenu_mc`. Le principe de propagation événementielle utilisé par AS3 permet aux objets contenus par un conteneur d'intercepter un événement appliqué à celui-ci. Nous utilisons pour cela la propriété target et nous désignons un écouteur global pour l'ensemble des symboles qu'il contient.

En cliquant sur l'un des symboles, une interpolation `TweenMax` restaure d'abord la position initiale de ceux qui ont éventuellement déjà été zoomés. Cette interpolation est très courte (`0.3` secondes) afin de ne pas créer une attente trop longue lorsqu'aucune image n'a encore été activée (au premier clic). Les propriétés animées sont l'index z, qui définit la position de l'image en profondeur. De valeur 0, elle désigne une position normale.

Nous restaurons aussi la position courante du symbole en X et Y, ainsi que son échelle `scaleX` et `scaleY`, à 50 % (`0.5`).

L'interpolation créée pour le premier symbole est déclinée pour l'ensemble des éléments à animer, soit 9 fois. Pour chacun d'entre eux, nous modifions le nom de l'objet à manipuler et sa position en X et Y. Les valeurs utilisées sont celles stockées initialement dans les tableaux. Pour lire ces valeurs, nous reprenons le nom de chaque tableau en ajoutant deux crochets, avec, en paramètre, la valeur correspondant au symbole ciblé (`positionInitX[0]` et `positionInitY[0]`).

La première d'entre elles est identifiée avec un nom de variable `tween3D`. Cette instanciation nous permet d'ajouter une fonction une fois l'interpolation de restauration achevée.

La fonction `restaureFini`, qui est appelée, est placée dans la fonction `zoom`, car elle fait directement référence à l'objet cliqué avec target par le biais de la variable evt, utilisée en paramètre de la fonction zoom. Si la fonction était placée en dehors de `zoom`, l'objet ne serait plus identifié. Si nous avions employé le même nom de variable en paramètre de la fonction restaureFini, nous aurions aussi généré une erreur, car le lecteur n'aurait pas su à quelle classe il devait se référer, `MouseEvent` ou bien `TweenEvent`, pour recevoir les informations. C'est la raison pour laquelle cette fonction emploie également un terme différencié du premier et sans valeur spécifique (`yo`).

La fonction `restaureFini` commence donc par redistribuer au sommet de la liste d'affichage, l'objet cliqué. Cette action replace virtuellement l'objet sur le calque du dessus, dans

contenu_mc, en faisant retomber la pile d'objets restant, un niveau en dessous, jusqu'à l'emplacement alors resté vide. Ainsi, l'image zoomée apparaîtra au premier plan indépendamment des autres, et non en chevauchant les autres images.

La fonction se poursuit avec une structure conditionnelle :

```
Tween3D.addEventListener(TweenEvent.COMPLETE,restaureFini);
   function restaureFini (yo:TweenEvent) {
      contenu_mc.addChild(MovieClip(evt.target));
   }
   if (nomImageActive!=evt.target.name) {
      nomImageActive=evt.target.name
      //
      TweenMax.to(evt.target, 2, {z:-100, x:0, y:0, scaleX:1.2, scaleY:1.2,
      ➥ delay:0.3, ease:Elastic.easeOut});
   } else {
      nomImageActive="";
   }
}
```

Si le nom du symbole cliqué (evt.target.name) est différent (!=) du nom enregistré dans la variable (evt.target.name) ou si l'image n'a pas déjà été cliquée, alors, une action est exécutée.

Cette action commence par mémoriser le nom de l'objet cliqué afin de pouvoir l'identifier comme actif lors du prochain clic.

La deuxième instruction est une interpolation. Elle cible l'objet cliqué (evt.target) et lui affecte un index de profondeur –100. L'objet est donc rapproché de l'écran. La position en X et Y est recentrée afin que l'image agrandie ne sorte pas de l'écran lorsqu'un symbole situé dans les coins est activé. Puis, en plus du changement d'index z, l'image est agrandie à 120 %. Comme nous avons activé le lissage des pixels, cette valeur reste tolérable et la déformation demeure encore imperceptible.

Si la condition else n'est pas vérifiée, nous purgeons le nom enregistré. Cela permet que lorsque l'utilisateur clique directement deux fois sur la même image, de l'ouvrir à nouveau, sans devoir cliquer au préalable sur une autre image. Tant que le nom de l'image n'est pas modifié, l'image déjà cliquée ne pourra pas en effet être rejouée, sauf si nous modifions ce nom en le substituant, par exemple, avec un texte vide ("").

Le programme s'achève avec la gestion des rollOver :

```
// rollOvers
contenu_mc.addEventListener(MouseEvent.MOUSE_OVER,over);
function over (evt:MouseEvent) {
   evt.target.filters=[haloIn];
}
contenu_mc.addEventListener(MouseEvent.MOUSE_OUT,out);
function out (evt:MouseEvent) {
   evt.target.filters=[haloOut];
}
```

En passant avec le pointeur sur les éléments du conteneur, le filtre dynamique `GlowFilter (filters=[haloIn])` affiche un halo blanc autour de l'objet survolé (`evt.target`). De la même manière, en sortant le pointeur de l'objet survolé, le filtre est initialisé avec le second objet `GlowFilter (filters=[haloOut])`.

À retenir

- Pour regrouper le stockage de valeurs, nous pouvons utiliser un tableau grâce à la méthode `Array`. Puis les redistribuer en l'invoquant par son nom et en paramètre duquel nous spécifions l'objet stocké à lire.
- Afin de placer un symbole au premier plan, dans un espace 3D, nous devons le repositionner au sommet de la liste d'affichage, avec, par exemple, la méthode `addChild()`.
- Lorsque plusieurs fonctions sont imbriquées, il est souhaitable de distinguer les identifiants passés en paramètre, afin de distinguer les classes distribuées par les différents écouteurs.

Galerie vidéo 3D circulaire

Une galerie en 3D avec de la vidéo peut paraître similaire, d'un point de vue structurel, à une galerie 3D d'images. Mais, pour obtenir une image propre et un document optimisé, vous devez tenir compte de certaines distinctions :

- L'image jouée en vidéo doit être dimensionnée en fonction de l'affichage final.
- Lors des transitions, pour préserver la qualité de l'image, c'est le composant vidéo qui doit être redimensionné et non son conteneur. C'est donc le conteneur qui évolue dans l'espace 3D et non la vidéo.
- Les vidéos doivent de préférence être arrêtées si aucune interaction de l'utilisateur n'est avérée. Ceci afin d'optimiser les ressources machines de l'utilisateur déjà sollicitées par la gestion de l'affichage 3D.

Dans cet exemple, nous présentons une galerie d'écrans vidéo disposés de manière circulaire. La galerie oscille selon la position du pointeur sur le même mécanisme que l'exemple précédent. En survolant les écrans, un effet de rollOver se produit, et, simultanément, la vidéo survolée démarre. En cliquant sur la vidéo, celle-ci est projetée frontalement avec un zoom spatial (échelle + index z) et l'oscillation est arrêtée. Lorsque l'utilisateur clique à nouveau sur la vidéo, celle-ci revient à sa position initiale (voir Figures 9.21 et 9.22).

Dans cette section, nous proposons deux approches pour la construction de cette interface. La première, accessible facilement, met en forme le dispositif à partir d'instructions simples et répétées. Une seconde approche exploite à l'inverse une structure tabulaire pour véhiculer l'ensemble des propriétés des objets, afin de simplifier la gestion du code. Vous trouverez donc, nécessairement, la formule qui vous convient le mieux pour réaliser vos propres systèmes d'affichage en 3D.