Figure 9.21
Aperçu du document à la publication, vidéo survolée.

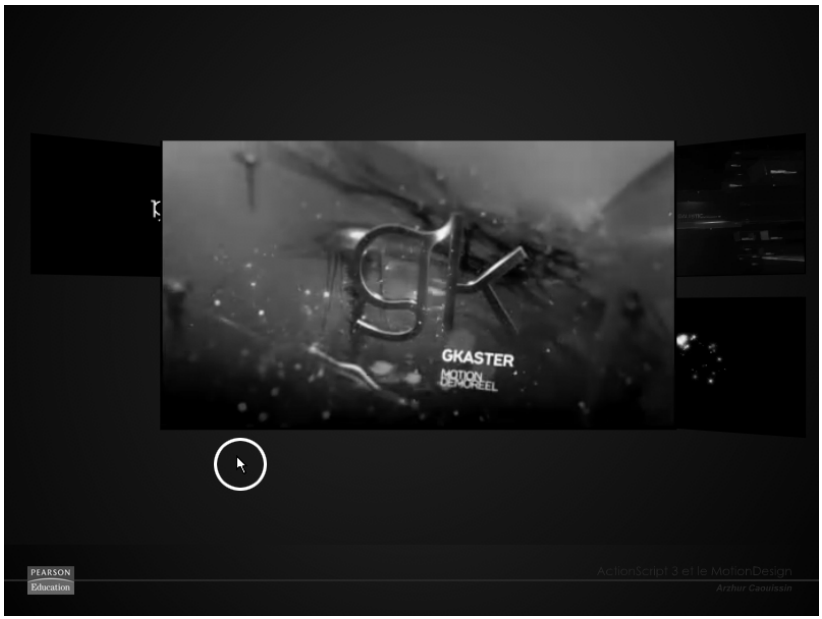


Figure 9.22
Aperçu du document à la publication, vidéo projetée frontalement.

La galerie simple

Dans cette version, nous initialisons les objets à mettre en forme à partir d'instructions répétées. Puis, nous ajoutons les comportements requis pour les animations et l'interactivité.

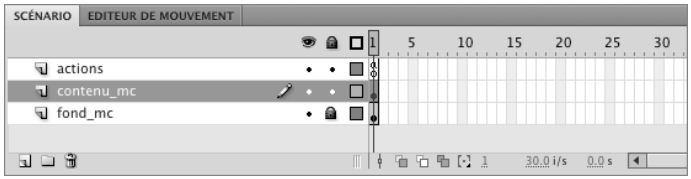
Exemples > ch9_3DNative_4.fla

Dans le document "ch9_3DNative_4.fla", sur la scène principale, le symbole `contenu_mc` affiche six occurrences du même MovieClip, mais de noms différents. Ce MovieClip contient un composant `FLVPlayBack` nommé `ecranVideo` (voir Figure 9.23).

Figure 9.23
Aperçu de la scène principale.

Dans la fenêtre de scénario, au-dessus du calque `fond_mc`, on distingue le symbole `contenu_mc` (voir Figure 9.24).

Figure 9.24
Aperçu du scénario.



La fenêtre Actions affiche 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, 0, 2, 2, 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;

//------------------ configuration des écrans vidéo
contenu_mc.v1_mc.ecranVideo.source="video3D/3d-gKaster-C19.f4v";
contenu_mc.v2_mc.ecranVideo.source="video3D/3d-gKaster-amusement.f4v";
contenu_mc.v3_mc.ecranVideo.source="video3D/3d-gKaster-balistic.f4v";
contenu_mc.v4_mc.ecranVideo.source="video3D/3d-gKaster-demoreel.f4v";
contenu_mc.v5_mc.ecranVideo.source="video3D/3d-galaxieFull.f4v";
contenu_mc.v6_mc.ecranVideo.source="video3D/3d-particules.f4v";
//
contenu_mc.v1_mc.rotationY=-30;
contenu_mc.v2_mc.rotationY=0;
contenu_mc.v3_mc.rotationY=30;
contenu_mc.v4_mc.rotationY=-30;
contenu_mc.v5_mc.rotationY=0;
contenu_mc.v6_mc.rotationY=30;
//
contenu_mc.v1_mc.z=-25;
contenu_mc.v2_mc.z=40;
contenu_mc.v3_mc.z=-25;
contenu_mc.v4_mc.z=-25;
contenu_mc.v5_mc.z=40;
contenu_mc.v6_mc.z=-25;
//
for (var i:Number=0; i<contenu_mc.numChildren; i++) {
   MovieClip(contenu_mc.getChildAt(i)).ecranVideo.stop();
   MovieClip(contenu_mc.getChildAt(i)).ecranVideo.scaleMode="exactFit";
   MovieClip(contenu_mc.getChildAt(i)).ecranVideo.autoPlay=false;
}

//------------------ actions
// mur d'image 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 vidéo
contenu_mc.addEventListener(MouseEvent.CLICK,zoomVideo);
function zoomVideo (evt:MouseEvent) {
   //
contenu_mc.removeEventListener(Event.ENTER_FRAME,murImages);
   TweenMax.to(contenu_mc, 3, {rotationX:0, rotationY:0, delay:0,
   ➥ ease:Elastic.easeInOut});
   //
```

```
    Tween3D=TweenMax.to(contenu_mc.v1_mc, 0.3, {z:-25, x:-245, y:-72,
    ➥ rotationY:-30, delay:0, ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.v2_mc, 0.3, {z:40, x:0, y:-72, delay:0, rotationY:0,
    ➥ ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.v3_mc, 0.3, {z:-25, x:245, y:-72, delay:0,
    ➥ rotationY:30, ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.v4_mc, 0.3, {z:-25, x:-245, y:72, delay:0,
    ➥ rotationY:-30, ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.v5_mc, 0.3, {z:40, x:0, y:72, delay:0, rotationY:0,
    ➥ ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.v6_mc, 0.3, {z:-25, x:245, y:72, delay:0,
    ➥ rotationY:30, ease:Strong.easeInOut});
    //
    TweenMax.to(contenu_mc.v1_mc.ecranVideo, 2, {width:220, height:123, x:-110,
    ➥ y:-62, delay:0.3, ease:Elastic.easeOut});
    TweenMax.to(contenu_mc.v2_mc.ecranVideo, 2, {width:220, height:123, x:-110,
    ➥ y:-62, delay:0.3, ease:Elastic.easeOut});
    TweenMax.to(contenu_mc.v3_mc.ecranVideo, 2, {width:220, height:123, x:-110,
    ➥ y:-62, delay:0.3, ease:Elastic.easeOut});
    TweenMax.to(contenu_mc.v4_mc.ecranVideo, 2, {width:220, height:123, x:-110,
    ➥ y:-62, delay:0.3, ease:Elastic.easeOut});
    TweenMax.to(contenu_mc.v5_mc.ecranVideo, 2, {width:220, height:123, x:-110,
    ➥ y:-62, delay:0.3, ease:Elastic.easeOut});
    TweenMax.to(contenu_mc.v6_mc.ecranVideo, 2, {width:220, height:123, x:-110,
    ➥ y:-62, delay:0.3, ease:Elastic.easeOut});
    //
    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.ecranVideo,2, {width:440, height:246, x:-220,
       ➥ y:-123, delay:0.3, ease:Elastic.easeOut});
       TweenMax.to(evt.target, 2, {z:-100, x:0, y:0, rotationY:0, delay:0.3,
       ➥ ease:Elastic.easeOut});
    } else {
       nomImageActive="";
       contenu_mc.addEventListener(Event.ENTER_FRAME,murImages);
    }
}

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

Le programme se décompose en cinq parties. La première importe les classes requises dont les filtres et les transitions `TweenMax`. La deuxième initialise les caractéristiques de chaque composant vidéo et le positionnement de son conteneur dans l'espace 3D. La troisième

active l'animation 3D du mur d'écrans. La quatrième lance l'interactivité avec les flux vidéo. La cinquième, enfin, ajoute l'effet rollOver sur chaque vidéo et active la lecture de cette vidéo.

Dans la deuxième partie, chaque composant est initialisé selon les paramètres suivants. L'ensemble de ces réglages est décliné pour chacune des vidéos. Voici les caractéristiques pour l'une d'entre elles :

```
//------------------ configuration des écrans vidéo
contenu_mc.v1_mc.ecranVideo.source="video3D/3d-gKaster-C19.f4v";
//
contenu_mc.v1_mc.rotationY=-30;
//
contenu_mc.v1_mc.z=-25;
```

Analysons le code :

- La propriété `source` désigne l'URL du fichier vidéo à lire.
- Les propriétés `rotationY` et `rotationZ` positionnent non pas le composant mais son conteneur, le MovieClip, dans l'espace 3D.

Plus loin, nous utilisons une boucle for afin de regrouper la gestion de l'ensemble des composants vidéo distribués dans chaque MovieClip :

```
for (var i:Number=0; i<contenu_mc.numChildren; i++) {
   MovieClip(contenu_mc.getChildAt(i)).ecranVideo.stop();
   MovieClip(contenu_mc.getChildAt(i)).ecranVideo.scaleMode="exactFit";
   MovieClip(contenu_mc.getChildAt(i)).ecranVideo.autoPlay=false;
}
```

Analysons le code :

- La méthode `stop()`, une fois la source appelée, permet de préserver les ressources de l'utilisateur en empêchant les vidéos de se lire automatiquement.
- La propriété `scaleMode` indique si le composant doit être redimensionné en fonction des dimensions réelles de la vidéo. La valeur `exactFit` interdit le redimensionnement. Si la vidéo est plus grande ou de proportions différentes du composant, elle épousera quand même les dimensions du composant (voir Chapitre 8 pour le descriptif détaillé de ces paramètres).
- Enfin, la propriété `autoPlay` passée sur `false` empêche la vidéo de lire au démarrage (deux précautions valent mieux qu'une).

Plus loin, les actions affichent le contrôle du mouvement du mur d'écrans (voir section précédente). À la suite, apparaissent les contrôles d'interactivité avec les flux vidéos :

```
// interactivité vidéo
contenu_mc.addEventListener(MouseEvent.CLICK,zoomVideo);
function zoomVideo (evt:MouseEvent) {
   //
contenu_mc.removeEventListener(Event.ENTER_FRAME,murImages);
   TweenMax.to(contenu_mc, 3, {rotationX:0, rotationY:0, delay:0,
   ➥ ease:Elastic.easeInOut});
}
```

Dans un premier temps, la fonction, appelée lorsque l'utilisateur active une vidéo, neutralise l'oscillation du mur (`removeEventListener`). Ainsi, la vidéo peut être repositionnée frontalement assez simplement et permettre une lecture confortable du contenu. Cette fonction sera réactivée lorsque toutes les vidéos seront restaurées à leur position initiale, plus loin dans le code.

Une deuxième instruction indique au symbole `contenu_mc`, dont nous venons d'interrompre le mouvement, de passer progressivement de sa position actuelle à une position frontale, pour laquelle tous les paramètres modifiés sont initialisés à la valeur 0 (`rotationX:0`, `rotationY:0`).

Ensuite, nous ajoutons les transitions pour chacun des composants et leurs conteneurs respectifs :

```
//
    Tween3D=TweenMax.to(contenu_mc.v1_mc, 0.3, {z:-25, x:-245, y:-72,
    ➥ rotationY:-30, delay:0 ,ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.v2_mc, 0.3, {z:40, x:0, y:-72, delay:0, rotationY:0,
    ➥ ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.v3_mc, 0.3, {z:-25, x:245, y:-72, delay:0,
    ➥ rotationY:30, ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.v4_mc, 0.3, {z:-25, x:-245, y:72, delay:0,
    ➥ rotationY:-30, ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.v5_mc, 0.3, {z:40, x:0, y:72, delay:0, rotationY:0,
    ➥ ease:Strong.easeInOut});
    TweenMax.to(contenu_mc.v6_mc, 0.3, {z:-25, x:245, y:72, delay:0,
    ➥ rotationY:30, ease:Strong.easeInOut});
    //
    TweenMax.to(contenu_mc.v1_mc.ecranVideo, 2, {width:220, height:123, x:-110,
    ➥ y:-62, delay:0.3, ease:Elastic.easeOut});
    TweenMax.to(contenu_mc.v2_mc.ecranVideo, 2, {width:220, height:123, x:-110,
    ➥ y:-62, delay:0.3, ease:Elastic.easeOut});
    TweenMax.to(contenu_mc.v3_mc.ecranVideo, 2, {width:220, height:123, x:-110,
    ➥ y:-62, delay:0.3, ease:Elastic.easeOut});
    TweenMax.to(contenu_mc.v4_mc.ecranVideo, 2, {width:220, height:123, x:-110,
    ➥ y:-62, delay:0.3, ease:Elastic.easeOut});
    TweenMax.to(contenu_mc.v5_mc.ecranVideo, 2, {width:220, height:123, x:-110,
    ➥ y:-62, delay:0.3, ease:Elastic.easeOut});
    TweenMax.to(contenu_mc.v6_mc.ecranVideo, 2, {width:220, height:123, x:-110,
    ➥ y:-62, delay:0.3, ease:Elastic.easeOut});
    //
```

Vous relevez que nous animons distinctement le conteneur et le composant, avec différentes propriétés.

Si nous avions effectivement animé uniquement le conteneur, en lui appliquant les propriétés de redimensionnement qui nous permettent d'agrandir l'image, celle-ci aurait été considérablement détériorée, même avec un lissage. Or, la vidéo s'adapte par rapport aux dimensions du composant. Il convient donc de redimensionner le composant et non le conteneur. C'est la raison pour laquelle nous obtenons deux séries d'interpolations, affectant chacune tous les conteneurs (`contenu_mc.v1_mc`) – voir Figure 9.25 – puis tous les composants (`contenu_mc.v1_mc.ecranVideo`).

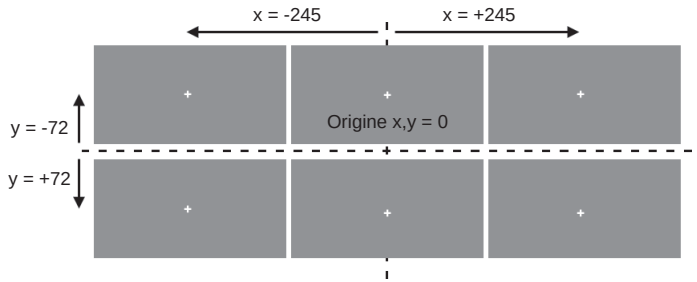


Figure 9.25
Calcul du positionnement des écrans, dans le conteneur `contenu_mc`.

Pour compenser néanmoins le décalage que peut provoquer le redimensionnement du composant indépendamment de son conteneur, nous rappelons les propriétés `x` et `y` de chaque objet pour recentrer l'ensemble à chaque modification d'échelle. Le changement de taille des éléments conduit en effet à les décaler de leur position d'origine. Il faut donc aussi animer et restaurer les propriétés `x` et `y` de chacun d'entre eux.

Pour permettre d'enchaîner ces transitions avec d'autres actions, l'une d'entre elles possède un identifiant `tween3D`.

Plus loin, nous ajoutons un écouteur à l'objet `tween3D`, comme vu à la section précédente. Nous enchaînons ici avec la condition suivante :

```
        //
        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.ecranVideo,2, {width:440, height:246, x:-220,
           ➥ y:-123, delay:0.3, ease:Elastic.easeOut});
           TweenMax.to(evt.target, 2, {z:-100, x:0, y:0, rotationY:0, delay:0.3,
           ➥ ease:Elastic.easeOut});
        } else {
           nomImageActive="";
           contenu_mc.addEventListener(Event.ENTER_FRAME,murImages);
        }
    }
```

Nous spécifions d'abord de placer le conteneur cliqué au premier plan pour éviter les chevauchements lors du redimensionnement :

```
contenu_mc.addChild(MovieClip(evt.target));
```

Nous vérifions ensuite la condition selon laquelle l'objet cliqué n'est pas déjà zoomé, selon le même principe que celui utilisé dans la section précédente.

Les transitions affectent les deux objets (`evt.target.ecranVideo et evt.target`).

Le composant vidéo activé est agrandi selon les dimensions exactes observées durant l'encodage. Nous avons relevé les valeurs 440 × 246 pixels. Nous les appliquons donc aux propriétés `width` et `height` du composant vidéo. Le flux vidéo, projeté au premier plan,

s'adaptera ainsi en pleine résolution à la nouvelle dimension du composant. Le conteneur de la vidéo activée (`evt.target`) est placé, quant à lui, frontalement. Si la condition n'est pas vérifiée, donc, si aucune autre vidéo ne prend la place centrale, nous réactivons le mur d'images (`addEventListener`).

Le programme se termine avec la gestion des `rollOver` :

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

Lorsque l'utilisateur survole un conteneur, la vidéo qu'il véhicule est jouée (`evt.target.ecranVideo.play()`). Le filtre du halo blanc est exécuté (`evt.target.filters=[haloIn]`).

Les mêmes instructions, mais neutralisées, sont exécutées lorsque le pointeur quitte la surface de l'objet (`MOUSE_OUT`).

La galerie optimisée

Dans cette version, nous optimisons la gestion des propriétés de chaque objet avant de réaliser les animations et développer l'interactivité. Nous distribuons ensuite, sous la forme de commandes compactes, chacune des instructions déjà étudiées dans la version précédente.

Exemples > ch9_3DNative_4b.fla

Le document possède une structure similaire au précédent. La fenêtre Actions affiche 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, 0, 2, 2, 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;
```

```
//------------------ configuration des écrans vidéo
var datas:Array = [
   {source:"video3D/3d-gKaster-C19.f4v",rotationY:-30,x:-245,y:-72,z:-25},
   {source:"video3D/3d-gKaster-amusement.f4v",rotationY:0,x:0,y:-72,z:40},
   {source:"video3D/3d-gKaster-balistic.f4v",rotationY:30,x:245,y:-72,z:-25},
   {source:"video3D/3d-gKaster-demoreel.f4v",rotationY:-30,x:-245,y:72,z:-25},
   {source:"video3D/3d-galaxieFull.f4v",rotationY:0,x:0,y:72,z:40},
   {source:"video3D/3d-particules.f4v",rotationY:30,x:245,y:72,z:-25}
];

for (var i:Number=0; i<contenu_mc.numChildren; i++) {
   contenu_mc["v"+i+"_mc"].ecranVideo.source=datas[i].source;
   contenu_mc["v"+i+"_mc"].rotationY=datas[i].rotationY;
   contenu_mc["v"+i+"_mc"].z=datas[i].z;
   contenu_mc["v"+i+"_mc"].ecranVideo.stop();
   contenu_mc["v"+i+"_mc"].ecranVideo.scaleMode="exactFit";
   contenu_mc["v"+i+"_mc"].ecranVideo.autoPlay=false;
}

//------------------ actions
// mur d'image 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 vidéo
contenu_mc.addEventListener(MouseEvent.CLICK,zoomVideo);
function zoomVideo (evt:MouseEvent) {
   //
   contenu_mc.removeEventListener(Event.ENTER_FRAME,murImages);
   TweenMax.to(contenu_mc, 3, {rotationX:0, rotationY:0, delay:0,
   ➥ ease:Elastic.easeInOut});
   //
   for (var i:Number=0; i<contenu_mc.numChildren; i++) {
      Tween3D=TweenMax.to(contenu_mc["v"+i+"_mc"], 0.3, {z:datas[i].z,
➥ x:datas[i].x, y:datas[i].y, rotationY:datas[i].rotationY, delay:0,
➥ ease:Strong.easeInOut});
      TweenMax.to(contenu_mc["v"+i+"_mc"].ecranVideo, 2, {width:220,
      ➥ height:123, x:-110, y:-62, delay:0.3, ease:Elastic.easeOut});
   }
   //
   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.ecranVideo, 2, {width:440, height:246, x:-220,
            ➥ y:-123, delay:0.3, ease:Elastic.easeOut});
            TweenMax.to(evt.target,2, {z:-100, x:0, y:0, rotationY:0, delay:0.3,
            ➥ ease:Elastic.easeOut});
         } else {
            nomImageActive="";
            contenu_mc.addEventListener(Event.ENTER_FRAME,murImages);
         }
      }

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

Dans la première partie du code, nous commençons par instancier l'ensemble des propriétés de chaque vidéo à travers un tableau, grâce à la méthode `Array`. Pour chaque nouvelle entrée, nous ouvrons et fermons une paire d'accolades. Chaque bloc créé enregistre, sous la forme de paramètres, chacune des propriétés et valeurs dont nous souhaitons disposer par la suite dans les fonctions d'animation :

```
var datas:Array = [
   {source:"video3D/3d-gKaster-C19.f4v",rotationY:-30,x:-245,y:-72,z:-25},
   {source:"video3D/3d-gKaster-amusement.f4v",rotationY:0,x:0,y:-72,z:40},
   {source:"video3D/3d-gKaster-balistic.f4v",rotationY:30,x:245,y:-72,z:-25},
   {source:"video3D/3d-gKaster-demoreel.f4v",rotationY:-30,x:-245,y:72,z:-25},
   {source:"video3D/3d-galaxieFull.f4v",rotationY:0,x:0,y:72,z:40},
   {source:"video3D/3d-particules.f4v",rotationY:30,x:245,y:72,z:-25}
];
```

Puis, nous utilisons une boucle `for` pour affecter, aux différents objets, les propriétés stockées dans le précédent tableau :

```
for (var i:Number=0; i<contenu_mc.numChildren; i++) {
   contenu_mc["v"+i+"_mc"].ecranVideo.source=datas[i].source;
   contenu_mc["v"+i+"_mc"].rotationY=datas[i].rotationY;
   contenu_mc["v"+i+"_mc"].z=datas[i].z;
   contenu_mc["v"+i+"_mc"].ecranVideo.stop();
   contenu_mc["v"+i+"_mc"].ecranVideo.scaleMode="exactFit";
   contenu_mc["v"+i+"_mc"].ecranVideo.autoPlay=false;
}
```

Dans ce contexte, pour cibler chaque objet individuellement, nous désignons d'abord le conteneur principal (`contenu_mc`), suivi de ses éléments descendants grâce à une syntaxe tabulée (avec des crochets []). En paramètre de cette structure, nous appelons les objets par leur nom d'occurrence en utilisant la valeur de `i` pour les distinguer les uns des autres

(`contenu_mc["v"+i+"_mc"]`). Il ne reste alors qu'à appliquer les valeurs relatives à chaque propriété initialement définie dans le tableau `datas`, pour affecter directement l'ensemble des objets. Ce qui donne, par exemple, pour le chemin de référence de chaque fichier vidéo désigné par la propriété `source` :

```
contenu_mc["v"+i+"_mc"].ecranVideo.source=datas[i].source;
```

Lorsque nous avons besoin de faire référence à chaque propriété de chacun des objets, nous invoquons également, dans les animations, le nom du tableau (`datas`) suivi du numéro d'ordre d'apparition de l'objet ciblé (avec `[i]`) et de la propriété concernée (par exemple `.z`). Ce qui donne pour la propriété `z` :

```
datas[i].z
```

Nous avons ensuite remplacé la répétition des interpolations `TweenMax` par une animation type, répétée dans une boucle `for`. Pour cibler chaque objet individuellement, nous reprenons le mécanisme abordé pour l'affectation des propriétés, dans la première partie du programme. Ce qui donne :

```
contenu_mc["v"+i+"_mc"]
```

À chaque itération des boucles `for`, ce sont tous les objets enregistrés qui sont affectés. Pour mettre à jour les données, il suffit donc de modifier les valeurs stockées dans le tableau initial (`datas`). L'ensemble de la construction est instantanément mise à jour.

À retenir

- *Afin d'optimiser les ressources d'affichage, si des vidéos sont déployées, il est souhaitable de ne pas les jouer toutes simultanément et de les organiser pour un affichage frontal.*
- *Les composants vidéos ne disposent pas des mêmes propriétés que les MovieClip. Il est possible d'animer ces composants dans des MovieClip pour disposer de plus de contrôles sur chacun des objets.*
- *Pour optimiser le stockage de valeurs et simplifier la distribution des instructions sur plusieurs objets simultanément, nous utilisons deux types de structure : les tableaux et une boucle `for`. Les tableaux permettent le stockage des propriétés. La boucle simplifie leur affectation.*
- *Afin de cibler dynamiquement un symbole d'un conteneur parent, nous pouvons le capturer en désignant d'abord le conteneur parent, et à l'aide de crochets, nous y inscrivons en paramètre le nom de l'objet enfant.*

Navigation spatiale 3D façon TimeMachine ou Aero

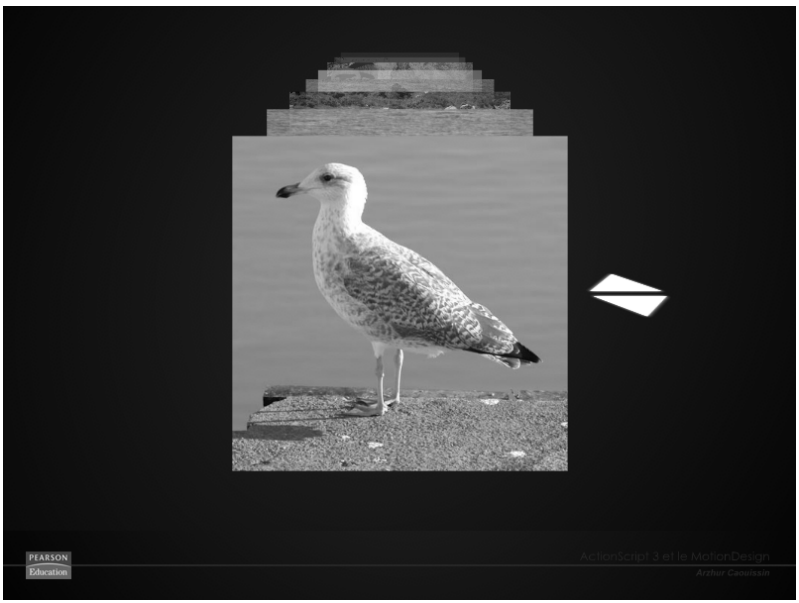
Puisque la 3D offre une profondeur Z, nous pouvons aussi distribuer les contenus de l'arrière-plan vers le premier plan comme le proposent, par exemple, les systèmes de navigation connus que sont Aero pour Windows et TimeMachine pour Apple (voir Figure 9.26).

Dans cette section, nous allons voir comment réaliser une navigation sur l'axe Z. Pour cela, nous utilisons différents MovieClip placés dans le scénario.

Figure 9.26
Aperçu du système d'archivage Apple Time Machine.

En cliquant sur une flèche, nous modifions leur position en Z avec une transition. Mais nous ajoutons au dispositif un système de boucle qui permet aussi de replacer les images déjà visitées, à l'arrière de la file. Nous ajoutons également un effet d'opacité selon la profondeur des objets. Nous déclinons enfin le principe en sens inverse avec une flèche qui oriente le mouvement dans le sens opposé (voir Figure 9.27).

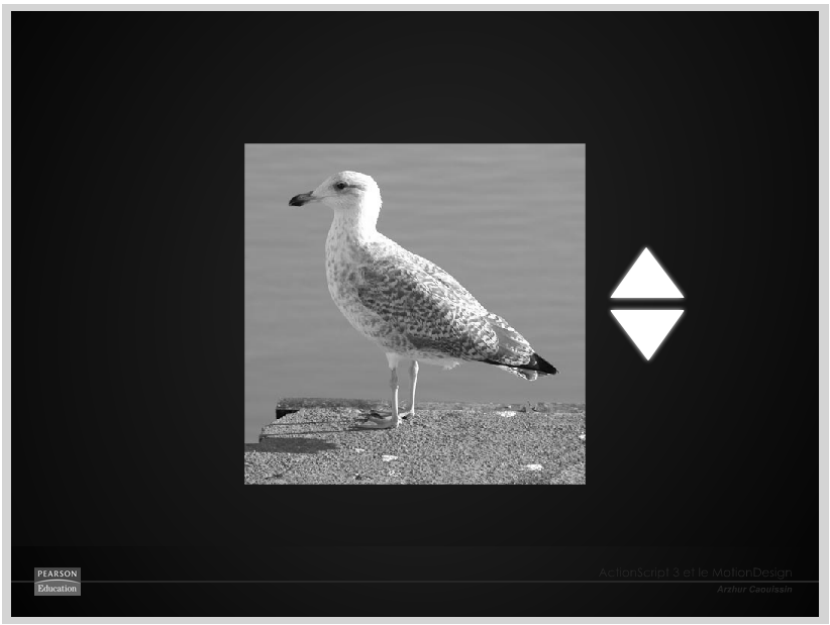
Figure 9.27
Aperçu du document publié.



Exemples > ch9_3DNative_5.fla

Sur la scène principale du document "ch9_3DNative_5.fla", nous pouvons voir un symbole `contenu_mc` qui intègre une série de neuf MovieClip. Chacun dispose d'un nom d'occurrence et est réparti vers les calques, dans l'ordre d'affichage qui correspond à l'ordre d'empilement dans l'espace. Un symbole `navigation_mc` contient également deux flèches, haut et bas, pour faire avancer et reculer les images sur l'axe Z (voir Figure 9.28).

Figure 9.28
Aperçu de la scène principale.



Dans la fenêtre de scénario, nous identifions les symboles `contenu_mc` et `navigation_mc` (voir Figure 9.29).

Figure 9.29
Aperçu du scénario.

La fenêtre d'actions affiche le code suivant :

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

```
var tween3DAvant:TweenMax;
var tween3DArriere:TweenMax;

contenu_mc.photo1_mc.z=1600;
contenu_mc.photo2_mc.z=1400;
contenu_mc.photo3_mc.z=1200;
contenu_mc.photo4_mc.z=1000;
contenu_mc.photo5_mc.z=800;
contenu_mc.photo6_mc.z=600;
contenu_mc.photo7_mc.z=400;
contenu_mc.photo8_mc.z=200;
contenu_mc.photo9_mc.z=0;

navigation_mc.rotationX=-90;

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

// point de fuite
var pointDeFuite:Point=new Point(stage.stageWidth/2,0);
transform.perspectiveProjection.projectionCenter=pointDeFuite;

// Avancer
navigation_mc.flecheHaut_btn.addEventListener(MouseEvent.CLICK,avancer);
function avancer(evt:MouseEvent) {
   for (var i:Number=0; i<contenu_mc.numChildren-1; i++) {
      TweenMax.to(contenu_mc.getChildAt(i), 0.5,
➥ {z:contenu_mc.getChildAt(i).z-200, alpha:1.1-((contenu_mc.getChildAt(i).z)/
➥ 1600), delay:0, ease:Strong.easeInOut});
   }
   if (contenu_mc.getChildAt(contenu_mc.numChildren-1).z<1) {
      tween3DAvant=TweenMax.to(contenu_mc.getChildAt(contenu_mc.numChildren-
      ➥ 1),0.5,{alpha:0,delay:0,ease:Strong.easeOut});
   }
   tween3DAvant.addEventListener(TweenEvent.COMPLETE,sortie);
}
//
function sortie(evt:TweenEvent) {
   contenu_mc.getChildAt(contenu_mc.numChildren-1).z=1600;
   contenu_mc.addChildAt(contenu_mc.getChildAt(contenu_mc.numChildren-1),0);
}

// Reculer
navigation_mc.flecheBas_btn.addEventListener(MouseEvent.CLICK,reculer);
function reculer(evt:MouseEvent) {
   for (var j:Number=0; j<contenu_mc.numChildren; j++) {
      tween3DArriere=TweenMax.to(contenu_mc.getChildAt(j), 0.5,
➥ {z:contenu_mc.getChildAt(j).z+200, alpha:1-((contenu_mc.getChildAt(j).z)
➥ /1600), delay:0, ease:Strong.easeInOut});
   }
   tween3DArriere.addEventListener(TweenEvent.COMPLETE,entree);
}
```

```
//
function entree (evt:TweenEvent) {
   contenu_mc.getChildAt(0).alpha=0;
   contenu_mc.addChildAt(contenu_mc.getChildAt(0),contenu_mc.numChildren-1);
   //
   var nombreDeBoucle:Number=1;
   var dureeBoucle:Number=100;
   var boucle:Timer=new Timer(dureeBoucle,nombreDeBoucle);
   boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
   boucle.start();
}
//
function lancerBoucle (evt:TimerEvent) {
   contenu_mc.getChildAt(contenu_mc.numChildren-1).z=0;
   TweenMax.to(contenu_mc.getChildAt(contenu_mc.numChildren-1), 0.5, {alpha:1,
   ➥ delay:0, ease:Strong.easeInOut});
}

// initialiser les alphas
for (var k:Number=0; k<contenu_mc.numChildren-1; k++) {
   contenu_mc.getChildAt(k).alpha=1-((contenu_mc.getChildAt(k).z)/1600);
}
```

Le programme est structuré en cinq parties dont : l'initialisation, la gestion du point de fuite, la navigation avec la flèche du haut, puis avec la flèche du bas, et enfin, la gestion de l'opacité au lancement de l'application. L'ordre d'exécution et les valeurs de calcul n'étant pas les mêmes pour les deux flèches, nous avons préféré isoler ici les deux fonctions.

D'abord, nous définissons les classes et quelques variables requises pour les transitions. Nous modifions ensuite l'agencement des contenus dans l'espace.

Plus loin, dans les actions, nous redéfinissons la position du point de fuite, par défaut placé au centre de la scène. Dans notre configuration, nous voulons surplomber légèrement l'animation de manière à percevoir les objets, de même dimensions, situés à l'arrière-plan des premiers éléments. Pour ce faire, nous utilisons la propriété `filedOfView` de la classe `perspectiveProjection` :

```
// point de fuite
var pointDeFuite:Point=new Point(stage.stageWidth/2,0);
transform.perspectiveProjection.projectionCenter=pointDeFuite;
```

Le point de fuite est désigné par un objet `Point` qui possède deux valeurs, `x` et `y` (vu lors de la programmation de squelettes).

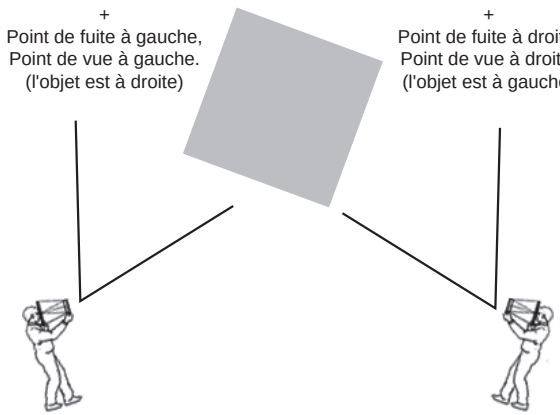
Le principe du point de fuite, en perspective, est de situer le point de départ de tous les tracés fuyants qui partent de la ligne d'horizon. Plus le point de fuite est haut, plus vous dominez la scène. Plus vous prenez, en somme, de l'altitude.

Vous pouvez aussi placer le point de fuite horizontalement. Cela détermine la direction de la perspective. Si le point de fuite est situé à gauche (de valeur `x` faible), vous vous placez alors à gauche de la scène et l'objet se trouve sur votre droite. Si le point de fuite se trouve à droite, votre regard aussi se retrouve à droite et l'objet, lui, se retrouve à gauche (voir

Figure 9.30). Vous placez le point de fuite en fonction du côté de l'objet que vous souhaitez observer.

Le point de fuite représente aussi le reflet de la position de votre œil sur la ligne d'horizon, autrement dit, du point de vue.

Figure 9.30
Placement du point de fuite.



Le langage

Propriétés 3D de la scène

La scène comporte plusieurs propriétés qui peuvent être contrôlées en ActionScript, y compris en animation. Ces propriétés sont : le champ de vision, le point de fuite et la focale.

- Le champ de vision se définit entre 0 et 180 degrés. Le code suivant applique un champ de vision de 55˚, qui est aussi la valeur appliquée par défaut.

```
var champDeVision:Number=55;
this.transform.perspectiveProjection.fieldOfView=champDeVision;
```

- Le point de fuite désigne le point de départ de tout tracé de forme perspective, à partir de la ligne d'horizon. Il représente la position de l'œil du spectateur. Le code suivant indique la valeur appliquée par défaut dans un document Flash 3D :

```
var pointDeFuite:Point = new Point( stage.stageWidth/2, stage.stageHeight/2);
transform.perspectiveProjection.projectionCenter = pointDeFuite;
```

- La focale indique le type de déformation appliqué aux fuyantes, déterminé par la distance entre l'écran et l'objet. La focale est calculée dynamiquement, comme suit :

```
var focale:uint=stage.stageWidth/ 2 * ( Math.cos(champDeVision/2) /
➥ Math.sin(champDeVision/2) );
transform.perspectiveProjection.focalLength= focale;
```

- Vous pouvez, par exemple, modifier la focale ou toute autre propriété de la scène 3D, en la manipulant à travers une instruction, comme celle-ci où les valeurs utilisées reprennent la position courante du pointeur en X :

```
addEventListener(Event.ENTER_FRAME,focaleTest);
function focaleTest(evt:Event) {
    transform.perspectiveProjection.focalLength=mouseX;
}
```

Le défilement des images de l'arrière vers le premier plan est activé par la flèche du haut. Les actions associées à ce comportement sont rassemblées dans la partie "Avancer" :

```
// Avancer
navigation_mc.flecheHaut_btn.addEventListener(MouseEvent.CLICK,avancer);
function avancer(evt:MouseEvent) {
   for (var i:Number=0; i<contenu_mc.numChildren-1; i++) {
      TweenMax.to(contenu_mc.getChildAt(i), 0.5,
{z:contenu_mc.getChildAt(i).z-200, alpha:1.1-((contenu_mc.getChildAt(i).z)
➥ /1600), delay:0, ease:Strong.easeInOut});
   }
// autres actions
}
```

Dans un premier temps, la fonction `avancer` est exécutée sur un clic de souris. Dans cette fonction, nous mettons en place une boucle for qui applique une interpolation `TweenMax` pour l'ensemble des éléments contenus dans le clip principal (avec `contenu_mc.getChildAt(i)`), comme vu précédemment. La transition réduit ici l'index z d'une valeur de 200 pixels, pour chaque objet de la liste d'affichage et les fait ainsi progresser, simultanément, vers l'écran.

Dans le même temps, un alpha permet de modifier aussi l'opacité des objets en fonction de leur index respectif :

```
alpha:1.1-((contenu_mc.getChildAt(i).z)/1600),
```

La valeur 1 600 correspond au seuil que nous avons arbitrairement fixé, pour le démarrage du défilement, en plaçant le premier objet à cette profondeur (voir les paramètres d'initialisation en début de programme).

Dans cette équation, plus l'index est proche de 1 600, plus l'alpha diminue. Inversement, plus il tend vers 0, plus l'objet se révèle.

Pour comprendre plus précisément le calcul, nous savons que l'alpha se mesure sur une échelle de 0 à 1. Or, l'index maximum que nous observons est 1 600. Il n'est pas possible d'appliquer un alpha de 1 600 en attribuant systématiquement la valeur de l'index z comme valeur d'alpha. L'équation reprend donc l'index z de chaque objet et le divise par la profondeur z maximum pour obtenir une valeur comprise entre 0 et 1.

Mais, ce faisant, nous obtenons l'inverse de ce que nous désirons. L'alpha est nul pour les images de premier plan et vaut presque 1 pour les images situées à l'arrière. Pour inverser les valeurs, nous effectuons donc une petite soustraction à partir de la valeur d'alpha maximum : 1. Enfin, pour garantir toutefois que les premières images situées en premier plan demeurent intégralement visibles et ne risquent pas d'être légèrement translucides (d'alpha légèrement inférieur à 1), nous augmentons la valeur à 1.1.

À la suite, dans la même fonction, nous plaçons quelques structures conditionnelles pour déterminer quels objets de la liste doivent être replacés au premier-plan :

```
function avancer(evt:MouseEvent) {
   for (var i:Number=0; i<contenu_mc.numChildren-1; i++) {
      TweenMax.to(contenu_mc.getChildAt(i), 0.5,
➥ {z:contenu_mc.getChildAt(i).z-200, alpha:1.1-((contenu_mc.getChildAt(i).z)/
➥ 1600), delay:0, ease:Strong.easeInOut});
```

```
    }
    if (contenu_mc.getChildAt(contenu_mc.numChildren-1).z<1) {
       tween3DAvant=TweenMax.to(contenu_mc.getChildAt(contenu_mc.numChildren-
       ➥ 1),0.5,{alpha:0,delay:0,ease:Strong.easeOut});
    }
    tween3DAvant.addEventListener(TweenEvent.COMPLETE,sortie);
}
//
function sortie(evt:TweenEvent) {
    contenu_mc.getChildAt(contenu_mc.numChildren-1).z=1600;
    contenu_mc.addChildAt(contenu_mc.getChildAt(contenu_mc.numChildren-1),0);
}
```

Une condition vérifie d'abord les objets dont l'index est inférieur à 1, c'est-à-dire que nous filtrons uniquement les objets situés au premier plan. Pour ces objets, nous ajoutons une interpolation qui le fait disparaître avec l'alpha passé à 0.

Dès que la transition (`tween3DAvant`) est terminée (`COMPLETE`), un écouteur exécute une autre fonction (sortie). Nous modifions alors l'index z de l'objet situé au sommet de la liste d'affichage, c'est-à-dire, en premier plan, et le ramenons à 1 600, donc, à une échelle qui le représente à l'arrière-plan. Mais, comme nous le savons, la 3D dans Flash est limitée à chaque objet. Il faut donc, en plus, modifier l'ordre d'empilement de l'objet dans la liste d'affichage. La deuxième instruction prend donc l'objet situé au sommet de la pile (`contenu_mc.getChildAt(contenu_mc.numChildren-1)`) pour le replacer au-dessous (0).

Les actions exécutées pour la flèche du bas sont une déclinaison de la flèche du haut, à ceci près que les valeurs `z` et `alpha` sont inversées.

Nous relevons néanmoins que, par nécessité, nous utilisons un chronomètre (Timer) pour décaler, dans le temps, les deux méthodes qui interviennent sur l'ordre d'affichage (`addChild()` et `getChild()`) afin d'éviter les conflits. En exécutant les deux instructions simultanément, selon le contexte et le poids des contenus affichés dans les objets, le repositionnement à l'index z pourrait effectivement ne pas avoir lieu, écrasé par la méthode `addChild()`, prioritaire et peut-être toujours en cours d'exécution :

```
//
function entree (evt:TweenEvent) {
    contenu_mc.getChildAt(0).alpha=0;
    contenu_mc.addChildAt(contenu_mc.getChildAt(0),contenu_mc.numChildren-1);
    //
    var nombreDeBoucle:Number=1;
    var dureeBoucle:Number=100;
    var boucle:Timer=new Timer(dureeBoucle,nombreDeBoucle);
    boucle.addEventListener(TimerEvent.TIMER,lancerBoucle);
    boucle.start();
}
//
function lancerBoucle (evt:TimerEvent) {
    contenu_mc.getChildAt(contenu_mc.numChildren-1).z=0;
   TweenMax.to(contenu_mc.getChildAt(contenu_mc.numChildren-1), 0.5, {alpha:1,
    ➥ delay:0, ease:Strong.easeInOut});
}
```

Le programme s'achève avec l'initialisation de l'alpha au démarrage de l'application. Ceci afin d'appliquer, dès le début, une transparence aux objets situés en profondeur :

```
// initialiser les alphas
for (var k:Number=0; k<contenu_mc.numChildren-1; k++) {
   contenu_mc.getChildAt(k).alpha=1-((contenu_mc.getChildAt(k).z)/1600);
}
```

Dans le fichier SWF, les deux flèches actionnent le défilement, dans un sens comme dans l'autre, en préservant toujours l'alpha et les index de chaque objet.

Info

Pour en savoir plus sur la 3D dans Flash, et notamment sur la gestion de formes matricielles (volumes composés de triangles), consultez l'adresse suivante : **http://help.adobe.com/fr_FR/ActionScript/3.0_ProgrammingAS3/WSF24A5A75-38D6-4a44-BDC6-927A2B123E90.html**. *D'autres techniques permettent aussi de gérer la 3D à partir de formes déjà modelées. Nous les abordons au chapitre suivant.*

À retenir

- *Vous pouvez contrôler la focale et le point de fuite d'une perspective 3D grâce à la propriété* `perspectiveProjection`.
- *L'ordre d'affichage des objets détermine la cohérence d'un système de navigation spatial. Nous utilisons la méthode* `addChild()` *afin de redistribuer les objets selon leur position sur l'index z.*

Synthèse

Dans ce chapitre, vous avez appris à créer des présentations 3D à partir des classes natives de Flash, compatibles avec Flash 10 (CS4) et versions ultérieures. La 3D dans Flash est simple, mais, ne permet pas encore d'importer directement des objets modélisés en 3D sans recourir à des classes externes. Elle apparaît malgré tout en voie de standardisation. Le lecteur 10 est, fin 2009, déjà implanté en Europe à près de 92 % des postes utilisateurs disposant d'un lecteur Flash. La 3D reste donc à ce jour largement compatible avec les configurations existantes des utilisateurs (source : **http://www.adobe.com/products/player_census/flashplayer/version_penetration.html**). Seule la solidité de la configuration (ressources graphiques et bandes passantes) peuvent en tempérer encore la diffusion.