Interactivité avec les touches du clavier

La représentation d'objets en volume implique la nécessité de contrôler la navigation dans l'espace. Dans cette section, nous allons voir comment reconstituer un décor virtuel à partir de formes primitives texturées, puis comment y naviguer à l'aide des commandes du clavier. Pour cela, nous devons déterminer la possibilité d'effectuer une combinaison de touches (Maj+Flèche droite, par exemple). Enfin, nous modifierons le sens de défilement des flèches du clavier, en fonction de l'orientation de la caméra dans le décor. Pour guider l'utilisateur, une boussole est placée sur la scène du document Flash et donne l'orientation de la caméra dans l'espace 3D.

Exemples > ch10_3DPaperVision_3.fla

Dans le document "ch10_3DPaperVision_3.fla", sur la scène principale, au-dessus du calque `fond_mc` et de l'habillage gris, apparaît une boussole. À l'intérieur du symbole qui le compose, deux clips sont répartis vers les calques et possèdent leur nom d'occurrence.

En publiant le document, des images tapissent des pans de murs virtuels. En activant les flèches du clavier, vous avancez, reculez ou vous déplacez à droite et à gauche. En appuyant simultanément sur majuscule et la touche flèche latérale, vous effectuez une rotation à 90° et pouvez reprendre une circulation vers l'avant en appuyant de nouveau sur la flèche du haut (voir Figures 10.30 et 10.31). L'aiguille de la boussole tourne en fonction de l'orientation définie pour la caméra.

Figure 10.30
Aperçu du document publié.

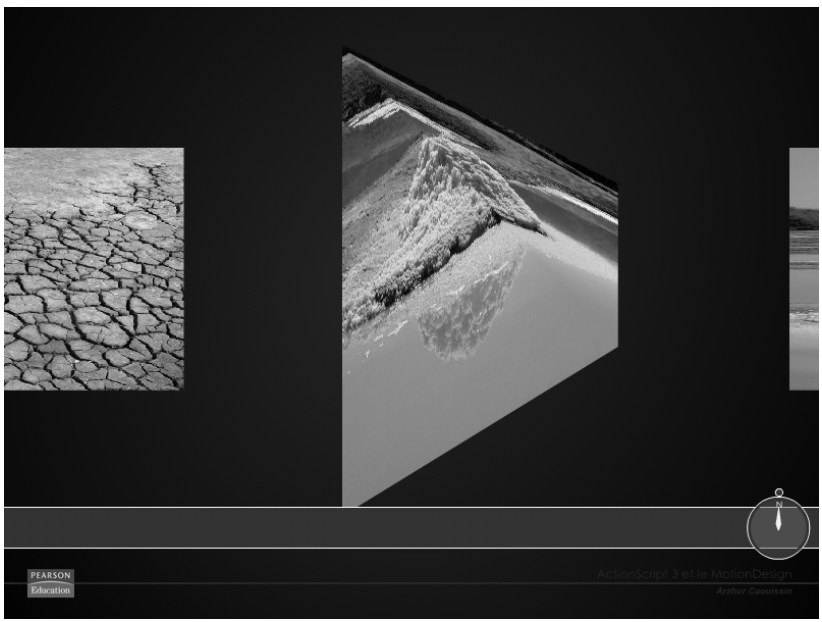
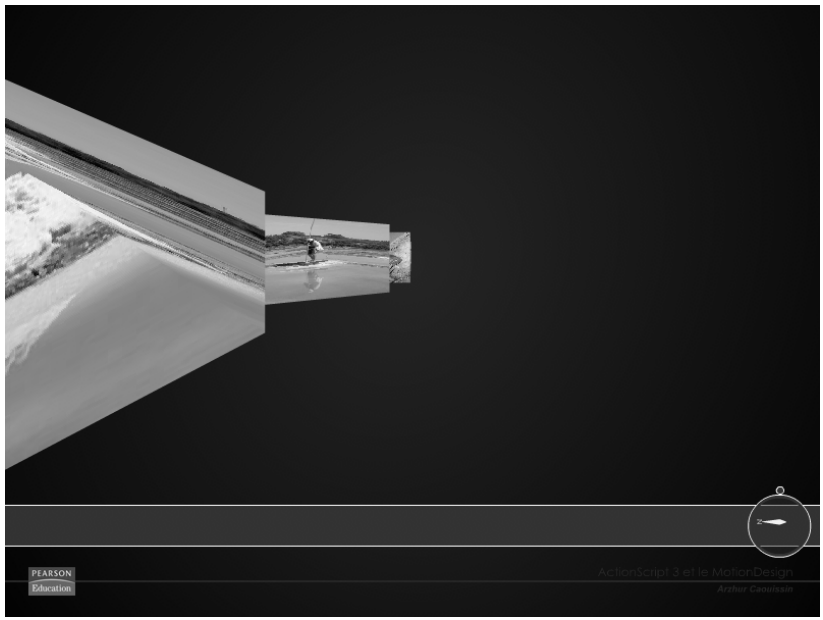
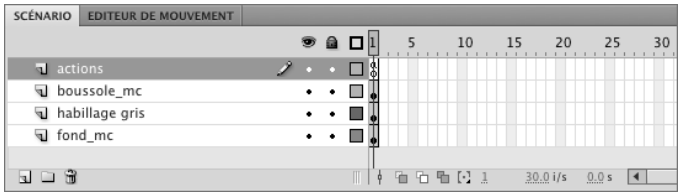


Figure 10.31
Aperçu du document publié, plus loin dans l'animation.



Dans la fenêtre de scénario, l'habillage gris et la boussole apparaissent (voir Figure 10.32).

Figure 10.32
Aperçu du scénario de la scène principale.



Dans le calque `Actions`, nous pouvons lire le code suivant :

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

import org.papervision3d.view.*;
import org.papervision3d.cameras.*;
import org.papervision3d.scenes.*;
import org.papervision3d.render.*;
import org.papervision3d.objects.primitives.*;
import org.papervision3d.events.*;
import org.papervision3d.materials.*;
import org.papervision3d.objects.parsers.DAE;

var espace:Viewport3D;
var rendu:BasicRenderEngine;
var scene:Scene3D;
var camera:Camera3D;

var plan0:Plane=new Plane(new BitmapFileMaterial("images/galerie/photo0.jpg"),400,200);
```

```
var plan1:Plane=new Plane(new BitmapFileMaterial("images/galerie/photo1.jpg"),400,200);
var plan2:Plane=new Plane(new BitmapFileMaterial("images/galerie/photo2.jpg"),400,200);
var plan3:Plane=new Plane(new BitmapFileMaterial("images/galerie/photo3.jpg"),400,200);
var plan4:Plane=new Plane(new BitmapFileMaterial("images/galerie/photo4.jpg"),400,200);
var plan5:Plane=new Plane(new BitmapFileMaterial("images/galerie/photo5.jpg"),400,200);

//-------------------- actions
initialisation();
function initialisation() {
   activerPapervision();
   //
   camera.y=10;
   plan0.rotationY=0;
   plan0.x=0;
   plan1.rotationY=90;
   plan1.x=2000;
   plan2.rotationY=-45;
   plan2.x=500;
   plan3.rotationY=45;
   plan3.x=1500;
   plan4.rotationY=0;
   plan4.x=900;
   plan5.rotationY=90;
   plan5.x=3000;
   //
   activerAffichage3d();
}
//
function activerPapervision() {
   espace=new Viewport3D(stage.stageWidth,stage.stageHeight-110);
   addChild(espace);
   rendu = new BasicRenderEngine();
   scene = new Scene3D();
   camera = new Camera3D();
}
function activerAffichage3d() {
   scene.addChild(plan0);
   scene.addChild(plan1);
   scene.addChild(plan2);
   scene.addChild(plan3);
   scene.addChild(plan4);
   scene.addChild(plan5);
   addEventListener(Event.ENTER_FRAME, enBoucle);
   stage.addEventListener(KeyboardEvent.KEY_DOWN,navigationClavierDOWN);
   stage.addEventListener(KeyboardEvent.KEY_UP,navigationClavierUP);
}
//
function enBoucle(evt:Event) {
   rendu.renderScene(scene, camera, espace);
}


//-------------------------- contrôle clavier
var sens:Number=0;
var toucheBase:Number=0;
var toucheSupplementaire:Number=0;
```

```
//
function navigationClavierDOWN(evt:KeyboardEvent) {
   trace ("le code de la touche enfoncée est : "+evt.keyCode)
   //
   if (evt.keyCode==16) {
      toucheBase=16;
   }
}
function navigationClavierUP (evt:KeyboardEvent) {
   trace ("le code de la touche enfoncée est : "+evt.keyCode)
   trace ("sens = "+sens);
   trace("toucheBase = "+toucheBase)
   trace("toucheSupplementaire = "+toucheSupplementaire)
   //
   if (toucheBase==16 && toucheSupplementaire==37) {
      TweenMax.to(camera, 1, {rotationY:camera.rotationY-90, delay:0,
      ➥ ease:Strong.easeOut});
      toucheBase=999;
      toucheSupplementaire=999;
      sens-=90;
      if (sens<=-360) {
         sens=0;
      }
   }
   if (toucheBase==16 && toucheSupplementaire==39) {
      TweenMax.to(camera, 1, {rotationY:camera.rotationY+90, delay:0,
      ➥ ease:Strong.easeOut});
      toucheBase=999;
      toucheSupplementaire=999;
      sens+=90;
      if (sens>=360) {
         sens=0;
      }
   }
   //
   if (sens==0 || sens==360) {
      if (evt.keyCode==37) {
         toucheSupplementaire=37;
         TweenMax.to(camera, 1, {x:camera.x-200, delay:0, ease:Strong.easeOut});
      }
      if (evt.keyCode==38) {
         toucheSupplementaire=38;
         TweenMax.to(camera, 1, {z:camera.z+200, delay:0, ease:Strong.easeOut});
      }
      if (evt.keyCode==39) {
         toucheSupplementaire=39;
         TweenMax.to(camera, 1, {x:camera.x+200, delay:0, ease:Strong.easeOut});
      }
      if (evt.keyCode==40) {
         toucheSupplementaire=40;
         TweenMax.to(camera, 1, {z:camera.z-200, delay:0, ease:Strong.easeOut});
      }
   }
   //
   if (sens==90 || sens==270) {
      if (evt.keyCode==37) {
         toucheSupplementaire=37;
         TweenMax.to(camera, 1, {z:camera.z+200, delay:0, ease:Strong.easeOut});
      }
      if (evt.keyCode==38) {
         toucheSupplementaire=38;
```

```
            TweenMax.to(camera, 1, {x:camera.x+200, delay:0, ease:Strong.easeOut});
         }
         if (evt.keyCode==39) {
            toucheSupplementaire=39;
            TweenMax.to(camera, 1, {z:camera.z-200, delay:0, ease:Strong.easeOut});
         }
         if (evt.keyCode==40) {
            toucheSupplementaire=40;
            TweenMax.to(camera, 1, {x:camera.x-200, delay:0, ease:Strong.easeOut});
         }
      }
      //
      if (sens==-90 || sens==-270) {
         if (evt.keyCode==37) {
            toucheSupplementaire=37;
            TweenMax.to(camera, 1, {z:camera.z-200, delay:0, ease:Strong.easeOut});
         }
         if (evt.keyCode==38) {
            toucheSupplementaire=38;
            TweenMax.to(camera, 1, {x:camera.x-200, delay:0, ease:Strong.easeOut});
         }
         if (evt.keyCode==39) {
            toucheSupplementaire=39;
            TweenMax.to(camera, 1, {z:camera.z+200, delay:0,
            ease:Strong.easeOut});
         }
         if (evt.keyCode==40) {
            toucheSupplementaire=40;
            TweenMax.to(camera, 1, {x:camera.x+200, delay:0,
            ease:Strong.easeOut});
         }
      }
      //
      if (sens==180 || sens==-180) {
         if (evt.keyCode==37) {
            toucheSupplementaire=37;
            TweenMax.to(camera, 1, {x:camera.x-200, delay:0, ease:Strong.easeOut});
         }
         if (evt.keyCode==38) {
            toucheSupplementaire=38;
            TweenMax.to(camera, 1, {z:camera.z+200, delay:0, ease:Strong.easeOut});
         }
         if (evt.keyCode==39) {
            toucheSupplementaire=39;
            TweenMax.to(camera, 1, {x:camera.x+200, delay:0, ease:Strong.easeOut});
         }
         if (evt.keyCode==40) {
            toucheSupplementaire=40;
            TweenMax.to(camera, 1, {z:camera.z-200, delay:0, ease:Strong.easeOut});
         }
      }
      //
      TweenMax.to(boussole_mc.fondBoussole_mc, 1, {rotation:-sens, delay:0,
      ➥ ease:Strong.easeOut});
   }
```

Dans ce programme, basé sur les documents précédents, nous avons substitué l'objet DAE par des formes primitives de type `Plane` (plan) :

```
var plan0:Plane=new Plane(new
BitmapFileMaterial("images/galerie/photo0.jpg"),400,200);
```

En paramètre de la méthode `new Plane()`, nous ciblons l'image que nous voulons afficher en tant que texture, puis, sa largeur et sa hauteur, qui sont exprimées en pixels.

Les primitives de `PaperVision`. *Il existe plusieurs types de primitives disponibles dans* `PaperVision`. *Chaque objet peut recevoir une texture et posséder des dimensions libres si bien qu'une sphère ou un cylindre sont couramment employés pour créer des univers entiers (des ciels ou des panoramiques par exemple). Ainsi, pour créer une sphère de position x=0 et y=0, par exemple, vous inscrivez :*

```
var sphere:Sphere=new Sphere(new BitmapFileMaterial("images/ciel.jpg"),0,0);
```

Puis :

```
scene.addChild(sphere);
```

Vous pouvez éventuellement ajuster l'échelle des formes primitives avec la propriété `scale`.

Sur le même principe, la méthode `Plane()` *génère une forme plate, un plan rectangulaire.* `Cube()` *désigne un cube (voir aussi* **http://forum.hardware.fr/hfr/Programmation/Divers-6/cube-site-entrees-sujet_125429_1.htm**). `Cone()`, *crée un cône et* `Cylinder()` *crée un cylindre. Pour le descriptif détaillé de construction de ces objets, avec des textures bitmaps, consultez l'adresse suivante :* **http://astrois.info/blog/flash/realiser-une-visionneuse-de-panorama-vr-en-as3**.

Plus loin, nous gérons d'abord l'emplacement des objets :

```
plan0.rotationY=0;
plan0.x=0;
```

Puis, nous les ajoutons à la liste d'affichage :

```
scene.addChild(plan0);
```

Viennent ensuite les contrôles du clavier, activés par deux gestionnaires de la classe `KeyBoardEvent` :

```
stage.addEventListener(KeyboardEvent.KEY_DOWN,navigationClavierDOWN);
stage.addEventListener(KeyboardEvent.KEY_UP,navigationClavierUP);
```

Nous utilisons deux gestionnaires, l'un pour détecter les touches enfoncées (`KEY_DOWN`) et le second pour les touches relevées (`KEY_UP`).

Puisque le contrôle de deux touches enfoncées simultanément n'est pas pris en compte dans une même structure conditionnelle, nous avons contourné ce problème en distinguant deux fonctions. Ainsi, la première fonction se charge de détecter une touche enfoncée pendant que la seconde détecte une autre touche, relevée. Deux touches appuyées, puis relâchées presque simultanément, peuvent donc être vérifiées.

Dans la première fonction, nous gérons la combinaison avec la touche Majuscule (`keyCode=16`) et dans la seconde fonction, les touches flèches (37 = flèche gauche, 38 = flèche haut, 39 = flèche droite et 40 = flèche bas).

La première fonction commence par identifier la touche enfoncée en affichant son nom. Cela permet d'évaluer, en fonction du système et du clavier, si les valeurs utilisées correspondent bien à l'environnement de développement.

```
trace ("le code de la touche enfoncée est : "+evt.keyCode)
```

Cette fonction définit ensuite une variable qui nous renseigne sur le fait que la touche Majuscule (`keyCode==16`) est enfoncée :

```
if (evt.keyCode==16) {
   toucheBase=16;
}
```

Les touches de commande

En ActionScript 3, les touches de commande sont également identifiables par des noms de propriétés :

- La propriété `shiftKey` correspond à la touche Shift ou Majuscule.
- La propriété `altKey` correspond à la touche Alt.
- La propriété `ctrlKey` correspond à la touche Contrôle.

Cette valeur est initialisée dans la fonction suivante (`navigationClavierUP`), lorsque l'utilisateur relâche les touches du clavier. En d'autres termes, les actions associées à la touche Majuscule n'ont d'effet que si celle-ci reste enfoncée pendant que la deuxième fonction est exécutée. Seule la combinaison des deux touches modifie l'action à l'exécution de la deuxième fonction.

La deuxième fonction vérifie la touche complémentaire lorsqu'elle est relâchée en nous informant d'abord, dans la fenêtre de sortie, sur son nom, ainsi que différentes valeurs utilisées dans les autres conditions :

```
trace ("le code de la touche enfoncée est : "+evt.keyCode)
trace ("sens = "+sens);
trace("toucheBase = "+toucheBase)
trace("toucheSupplementaire = "+toucheSupplementaire)
```

Les deux conditions suivantes appliquent, si la touche Majuscule est effectivement enfoncée, une rotation en Y de 90°, dans un sens ou dans l'autre :

```
if (toucheBase==16 && toucheSupplementaire==37) {
   TweenMax.to(camera, 1, {rotationY:camera.rotationY-90, delay:0,
   ➥ ease:Strong.easeOut});
   toucheBase=999;
   toucheSupplementaire=999;
   sens-=90;
   if (sens<=-360) {
      sens=0;
   }
}
if (toucheBase==16 && toucheSupplementaire==39) {
   TweenMax.to(camera, 1, {rotationY:camera.rotationY+90, delay:0,
   ➥ ease:Strong.easeOut});
```

```
      toucheBase=999;
      toucheSupplementaire=999;
      sens+=90;
      if (sens>=360) {
         sens=0;
      }
   }
```

La valeur affectée à la touche Majuscule (`toucheBase=16`) est initialisée dès que l'action est activée. Ainsi, l'utilisateur peut effectuer d'autres commandes, qui n'impliquent pas la touche Majuscule, sans risquer de compromettre les autres actions du programme. Pour être initialisée, la valeur est passée à 999, qui ne correspond à aucune touche.

Si, en plus, la touche relâchée correspond à une des quatre flèches du clavier (`&& toucheSupplementaire==37`), alors, le sens de la caméra est modifié :

```
if (toucheBase==16 && toucheSupplementaire==37) {
      TweenMax.to(camera, 1, {rotationY:camera.rotationY-90, delay:0,
      ➥ ease:Strong.easeOut});
      toucheBase=999;
      toucheSupplementaire=999;
      sens-=90;
      if (sens<=-360) {
         sens=0;
      }
   }
```

L'action est déclinée aussi pour la flèche droite (39). Plus bas, des conditions lancent les différents mouvements de caméra, sans rotation, pour les combinaisons de touche simples. Ces animations sont exécutées même si l'utilisateur a enfoncé la touche Majuscule :

```
if (sens==0 || sens==360) {
   if (evt.keyCode==37) {
      toucheSupplementaire=37;
      TweenMax.to(camera, 1, {x:camera.x-200, delay:0, ease:Strong.easeOut});
   }
//actions des autres flèches du clavier
}
```

Le principe est décliné pour chaque touche (37, 38, 39 ou 40) et, surtout, selon chaque valeur d'orientation de caméra définie par sens.

Selon la position relative de la caméra, il faut en effet ajuster les commandes de sorte qu'un utilisateur qui a effectué une rotation de 90˚ à droite (Maj+Flèche droite) puisse continuer d'avancer en appuyant sur la flèche du haut, et non en appuyant sur la flèche de droite.

Voici plus précisément le mécanisme qui régit l'ensemble des instructions à exécuter : dans un espace 3D, au sol, sont imprimés les axes X et Z. Pour avancer tout droit, nous incrémentons notre position sur l'axe Z. Pour nous déplacer latéralement, en crabe, nous utilisons donc l'axe X. Si maintenant nous faisons pivoter la caméra de 90˚ à droite, nous percevons désormais ce qui longe l'axe X, anciennement à notre droite, en face de nous. Mais, la commande du clavier Flèche haut, elle, continue de contrôler le déplacement sur Z. Si nous activons la touche Flèche haut, nous nous déplaçons donc maintenant sur la gauche (axe Z). Pour que le déplacement à gauche se fasse désormais avec la flèche de gauche, nous devons définir une condition. Si nous voulons autoriser l'utilisateur à s'orienter indifféremment à droite ou à gauche et d'avant en arrière, nous devons définir quatre blocs de conditions.

Ces conditions vérifient l'orientation appliquée à la caméra à l'aide de la variable `sens`. Les valeurs sont incrémentées ou diminuées de 90 à chaque modification de la rotation de la caméra en Y. Lorsque cette valeur atteint la limite de +360 ou –360, nous l'initialisons à 0.

À l'intérieur de chaque condition, nous ajoutons donc des contrôles qui déterminent le type de déplacement à effectuer selon l'angle observé. Nous ajustons pour cela deux paramètres : le signe plus ou moins attribué au pas d'incrémentation ainsi que l'axe de défilement X ou Z :

```
TweenMax.to(camera, 1, {x:camera.x-200, delay:0, ease:Strong.easeOut});
```

Pour terminer, une interpolation indique à l'aiguille de la boussole de pivoter en fonction de cette orientation :

```
TweenMax.to(boussole_mc.fondBoussole_mc, 1, {rotation:-sens, delay:0,
➥ ease:Strong.easeOut});
```

Vérifier les valeurs de ses propres touches clavier. *Pour vérifier instantanément les valeurs de l'ensemble de vos touches clavier, placez le code suivant dans la fenêtre Actions d'un document Flash et publiez-le. Le document "ch10_3DpaperVision_3b.fla" contient ce programme :*

```
// afficher les commandes de clavier
for (var i:int = 33; i< 126;i++){
    trace(i + "\t: " + String.fromCharCode(i))
}
```

Dans les deux tableaux qui suivent, nous indiquons les valeurs numériques qui correspondent aux claviers ASCII français et américain. Vous y trouverez les valeurs vous permettant de développer d'autres actions à partir des commandes du clavier, selon sa configuration.

Tableau 10.1 : Valeurs des principales commandes de clavier ASCII

Dénomination des touches	Valeur numérique
Retour	8
Tabulation	9
Entrée	13
Majuscule	16
Ctrl/Contrôle	17
Alt	18
Pause/Break/Attn	19
Verrouillage majuscule	20
Escape	27
Page haut	33
Page bas	34
Fin	35
Sommaire (flèche haut et gauche, au-dessus de la touche fin)	36
Flèche gauche	37
Flèche haut	38
Flèche droite	39

Tableau 10.1 : Valeurs des principales commandes de clavier ASCII *(suite)*

Dénomination des touches	Valeur numérique
Flèche bas	40
Insert	45
Supprimer	46
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
a	65
b	66
c	67
d	68
e	69
f	70
g	71
h	72
i	73
j	74
k	75
l	76
m	77
n	78
o	79
p	80
q	81
r	82
s	83
t	84
u	85
v	86
w	87
x	88
y	89
z	90
Touche Windows de gauche	91

Tableau 10.1 : Valeurs des principales commandes de clavier ASCII *(suite)*

Dénomination des touches	Valeur numérique
Touche Windows de droite	92
0 (pavé numérique)	96
1 (pavé numérique)	97
2 (pavé numérique)	98
3 (pavé numérique)	99
4 (pavé numérique)	100
5 (pavé numérique)	101
6 (pavé numérique)	102
7 (pavé numérique)	103
8 (pavé numérique)	104
9 (pavé numérique)	105
Multiplier	106
Additionner	107
Soustraire	109
Point décimal	110
Diviser	111
f1	112
f2	113
f3	114
f4	115
f5	116
f6	117
f7	118
f8	119
f9	120
f10	121
f11	122
f12	123
Verrouillage pavé numérique	144
Verrouillage défilement	145
Point-virgule	186
Égal	187
Commande	188
Tiret	189
Point	190
Slash	191
Accent grave	192
Crochet ouvert	219
Back slash	220
Crochet fermé	221
Simple quote ou apostrophe simple	222

Tableau 10. 2 : Valeur des principales commandes de clavier US

Dénomination des touches	Valeur numérique
Alt	18
Arrow down	40
Arrow left	37
Arrow right	39
Arrow up	38
Backspace	8
Caps Lock	20
Ctrl	17
Delete	46
End	35
Enter	13
Esc	27
F1	112
F2	113
F3	114
F4	115
F5	116
F6	117
F7	118
F8	119
F9	120
F10	121
F11	122
F12	123
Home	36
Insert	45
Num Lock	144
(NumPad) -	109
(NumPad) *	106
(NumPad) .	110
(NumPad) /	111
(NumPad) +	107
(NumPad) 0	96
(NumPad) 1	97
(NumPad) 2	98
(NumPad) 3	99
(NumPad) 4	100
(NumPad) 5	101
(NumPad) 6	102
(NumPad) 7	103

Tableau 10. 2 : Valeur des principales commandes de clavier US *(suite)*

Dénomination des touches	Valeur numérique
(NumPad) 8	104
(NumPad) 9	105
Page Down	34
Page Up	33
Pause	19
Print Scrn	44
Scroll Lock	145
Shift	16
Spacebar	32
Tab	9
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90
1	49
2	50

Tableau 10. 2 : Valeur des principales commandes de clavier US *(suite)*

Dénomination des touches	Valeur numérique
3	51
4	52
5	53
6	54
7	55
8	56
9	57
0	48
`	222
-	189
,	188
.	190
/	191
;	186
[	219
\	220
]	221
`	192
=	187

Pour connaître la liste des principales commandes clavier supportées par le système Linux, reportez-vous à l'adresse suivante : **http://www.comptechdoc.org/os/linux/howlinuxworks/linux_hlkeycodes.html**.

Interactivité sur les objets 3D. Pour en savoir plus sur les interactions possibles avec les objets 3D, consultez la page suivante : **http://papervision3d-fr.com/2009/09/16/interactive-displayobject3d/**.

***Gestion des lumières dans* `PaperVision`.** Pour en savoir plus sur l'ajout de lumières dans la scène 3D, consultez la page : **http://papervision3d-fr.com/2009/09/22/la-lumiere-dans-papervision3d/**.

***Guide de référence* `PaperVision`.** Un guide de référence sur `PaperVision` est disponible à cette adresse : **http://papervision3d.googlecode.com/svn/trunk/as3/trunk/docs/class-summary.html**.

***Aide en ligne de* `PaperVision`.** De nombreux utilisateurs de la classe `PaperVision` travaillent en mode de programmation objet en externalisant leur code sous la forme de classes et de packages dans des fichiers ayant l'extension `.as`. Des sites communautaires et des blogs font référence à cette méthode. Vous y trouverez des aides complémentaires pour vous guider sur les techniques avancées de programmation avec `PaperVision` :

- **http://content.madvertices.com/articles/PV3DTraining/default.htm#setUp**.
- **http://wiki.mediabox.fr/tutoriaux/flash/papervision_3D_bases**.
- **http://blog.pixlp.com/index.php/**.

À retenir

- Les positions 3D des objets et des caméras peuvent être contrôlées par les touches du clavier à l'aide de la propriété `keyCode`.
- Pour construire une navigation viable, vous devez détecter la rotation de la caméra et en déterminer l'axe de défilement x ou z.

Synthèse

Dans ce chapitre, vous avez appris à intégrer des objets 3D au sein de l'environnement Flash, à l'aide de la classe `PaperVision`, en programmant les commandes `PaperVision` directement dans le scénario. Vous savez placer les scènes 3D dans un décor multicalques et interagir avec les trajectoires animées de la caméra et déplacer les objets 3D, y compris en utilisant les touches du clavier.

Les bogues d'affichage, le poids limité, comparé à la qualité de résolution disponible en flux vidéo F4V favorise cependant encore largement la vidéo interactive à la technologie 3D interactive sur le Web aujourd'hui. Mais les innovations en cours en font un outil à ne pas négliger dans les développements à venir.

11 API d'affichage et de colorimétrie

Introduction

Bien que Flash intègre déjà tout type de données, il peut être intéressant de développer des fonctionnalités avancées sur des contenus traités dynamiquement. Par exemple, vous pouvez proposer une fonctionnalité de grossissement d'une image, une fonctionnalité de retouche colorimétrique d'une image, pour tester des ambiances et offrir de nouveaux services à des utilisateurs exigeants.

Flash propose un moteur de traitement d'images avancé. Il dispose, pour cela, de deux classes qui permettent de modifier intrinsèquement les pixels d'une image (`colorMatrixFilter`) et de déformer un objet graphique (`Matrix`).

Dans ce chapitre, nous abordons le traitement des pixels de l'image à travers ces différentes méthodes. Nous voyons comment réaliser, entre autres, des interfaces sans crénelage, une loupe, des filtres de traitement colorimétrique, des fonctionnalités d'impression sur des zones définies de l'interface. Nous verrons ensuite, dans le prochain chapitre, comment exploiter ces techniques pour mettre au point, par exemple, un moteur de rendu en relief.

Lissage des images bitmap

Le lissage des images est requis dès que l'image suit une transformation d'échelle ou de rotation. Sans lissage, l'image animée apparaît crénelée et un voile cristallin semble même la déformer dès qu'elle subit un mouvement.

Comme évoqué aux Chapitres 5 et 9, nous avons vu comment lisser une image en activant l'option Autoriser le lissage, depuis les propriétés de chaque image, dans la bibliothèque. Mais, dans de nombreux cas, pour une galerie d'images dynamique par exemple, il peut être intéressant d'activer cette option avec ActionScript. Pour ce faire, nous utilisons la propriété smoothing.

Pour appliquer cette propriété, nous devons au préalable comprendre qu'elle s'applique seulement à un objet image, et non à son conteneur. Il en résulte que, pour lisser un MovieClip, nous devons, au préalable, créer une enveloppe image (Bitmap) qui capture ce que représente le MovieClip, pour seulement ensuite le lisser par ActionScript.

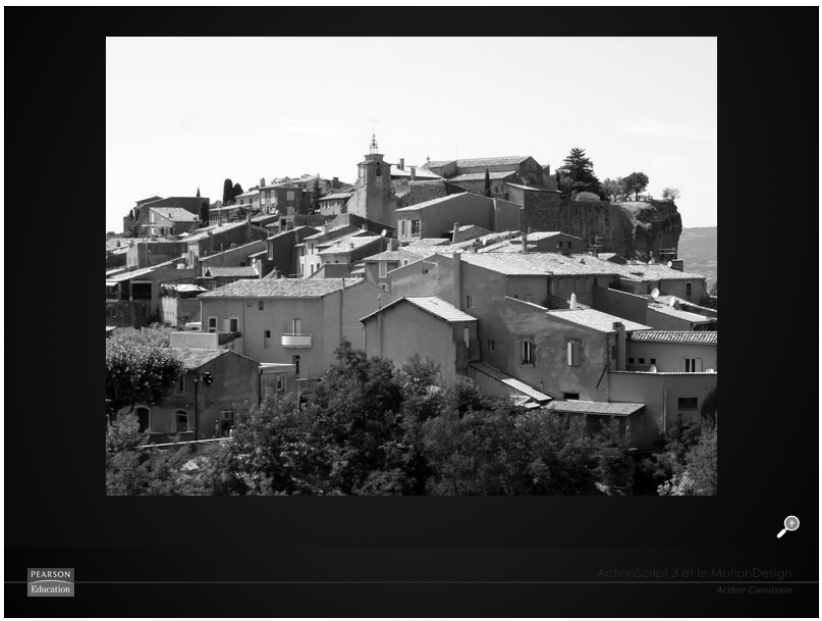
Dans cette section, nous allons voir comment lisser une image chargée dynamiquement sur la scène, dans un MovieClip.

Exemples > ch11_apiGraphisme_1.fla

Dans le document "ch11_apiGraphisme_1.fla", sur la scène principale, est positionné un MovieClip vide, nommé `contenu_mc`, ainsi qu'un bouton Loupe. Le bouton Loupe agrandit et effectue une rotation sur ce MovieClip, si bien que sans les propriétés de lissage, nous pouvons constater l'apparition d'un crénelage.

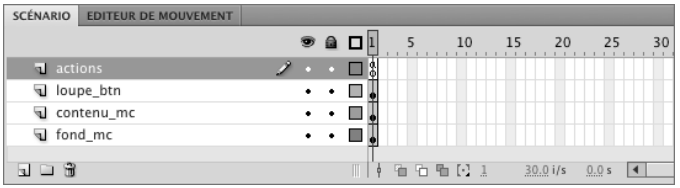
En publiant le document, l'image est d'abord chargée dans le MovieClip et affiche déjà un lissage. En activant le bouton Loupe, l'image effectue une rotation et subit un agrandissement, mais les pixels restent lisses, même à grande échelle (voir Figure 11.1).

Figure 11.1
Aperçu du document publié.



Dans la fenêtre de scénario de la scène principale, au-dessus du calque `fond_mc`, figurent le symbole `contenu_mc` (vide) et le bouton `loupe_btn` (voir Figure 11.2).

Figure 11.2
Aperçu du scénario de la scène principale.



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

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

```
import gs.events.*;

//---------------  actions
// chargement
var chargeur = new Loader();
chargeur.load(new URLRequest("images/provence/Roussillon.jpg"));
chargeur.contentLoaderInfo.addEventListener(Event.COMPLETE,chargementComplet);

function chargementComplet(evt:Event) {
   // lissage
   var imageDepart:Bitmap = evt.target.content;
   var informationImage:BitmapData=new
BitmapData(imageDepart.width,imageDepart.height);
   informationImage.draw(contenu_mc);
   imageDepart.smoothing=true;
   // affichage
   contenu_mc.addChild(imageDepart);
   contenu_mc.x=100;
   contenu_mc.y=30;
}

// bouton loupe
loupe_btn.addEventListener(MouseEvent.CLICK,tournerImage);
function tournerImage (evt:MouseEvent) {
   if (contenu_mc.rotation>-89) {
      TweenMax.to(contenu_mc, 2, {rotation:-90, scaleX:2, scaleY:2, x:0,
      ➥ y:520, delay:0, ease:Strong.easeInOut});
   } else {
      TweenMax.to(contenu_mc, 2, {rotation:0, scaleX:1, scaleY:1, x:100, y:30,
      ➥ delay:0, ease:Strong.easeInOut});
   }
}
```

Ce code est structuré en trois parties. La première appelle les classes utilisées pour les interpolations `TweenMax`. La deuxième charge l'image et la lisse. La troisième partie gère l'interactivité avec le bouton Loupe.

La première action de la deuxième partie charge l'image, comme vu au Chapitre 5. Aussitôt après, la fonction appelée affiche l'organisation de l'image chargée, pour lui permettre d'obtenir un lissage :

```
function chargementComplet(evt:Event) {
   // lissage
   var imageDepart:Bitmap = evt.target.content;
   var informationImage:BitmapData=new
BitmapData(imageDepart.width,imageDepart.height);
   informationImage.draw(contenu_mc);
   imageDepart.smoothing=true;
   // affichage
   contenu_mc.addChild(imageDepart);
   contenu_mc.x=100;
   contenu_mc.y=30;
}
```

Pour appliquer un lissage, nous définissons d'abord un objet vectoriel texturé d'une image. Et seulement une fois cet objet matérialisé, nous pouvons lui appliquer un lissage.

Pour lisser une image chargée dynamiquement, nous procédons donc en plusieurs étapes. D'abord, nous commençons par définir le flux chargé (`evt.target.content`) comme un objet image avec un type `Bitmap` :

```
var imageDepart:Bitmap = evt.target.content;
```

Puis, nous plaçons les données de l'image (ses informations colorimétriques), dans un nouvel objet `BitmapData` :

```
var informationImage:BitmapData=new
    ➥ BitmapData(imageDepart.width,imageDepart.height);
```

Nous indiquons, en paramètre de cette méthode, les dimensions de cet objet (largeur et hauteur). Les largeur et hauteur indiquées sont ici celles de l'objet `Bitmap` déjà identifié.

Les données véhiculées par l'objet `BitmapData` sont matérialisées sur le symbole `contenu_mc` par l'instruction suivante :

```
informationImage.draw(contenu_mc);
```

L'image est en fait placée sur l'enveloppe extérieure du MovieClip `contenu_mc` et non à l'intérieur. Pour le vérifier, nous pouvons lire le contenu du symbole en utilisant l'une des instructions suivantes. Ces instructions retournent chacune une valeur nulle. L'image n'est donc pas contenue dans l'objet, mais bien en surface :

```
trace(contenu_mc.numChildren) ; // 0
trace(contenu_mc.getChildAt(0)) ; // Index hors limites
```

Par contre, c'est bien à l'objet vectoriel texturé BitmapData que nous appliquons la propriété de lissage `smoothing` :

```
imageDepart.smoothing=true;
```

Nous pouvons alors afficher l'objet, le placer avec des propriétés `x` et `y`, voire, le déformer. Il restera lisse :

```
contenu_mc.addChild(imageDepart);
contenu_mc.x=100;
contenu_mc.y=30;
```

Le code se termine sur la gestion du bouton qui effectue les interpolations d'échelle et de rotation :

```
// bouton loupe
loupe_btn.addEventListener(MouseEvent.CLICK,tournerImage);
function tournerImage (evt:MouseEvent) {
   if (contenu_mc.rotation>-89) {
      TweenMax.to(contenu_mc, 2, {rotation:-90, scaleX:2, scaleY:2, x:0,
      ➥ y:520, delay:0, ease:Strong.easeInOut});
   } else {
      TweenMax.to(contenu_mc, 2, {rotation:0, scaleX:1, scaleY:1, x:100, y:30,
      ➥ delay:0, ease:Strong.easeInOut});
   }
}
```

Dans la structure conditionnelle, si le conteneur affiche une rotation inférieure à –89°, la première transition est jouée et porte ce conteneur à une rotation de 90°. Lorsque la condition

n'est plus vérifiée, une interpolation inverse est lancée et retourne l'objet à une rotation 0, supérieure donc à –89°. Chaque interpolation, en plus de faire tourner l'objet, applique une modification d'échelle. Sans le lissage, l'image laisserait apparaître un crénelage. Pour vérifier que le lissage est bien actif, vous pouvez neutraliser l'instruction suivante, en commentaire, puis la restaurer:

```
//imageDepart.smoothing=true;
```

À la Figure 11.3, nous soulignons la différence entre l'image affichée avec un lissage et celle qui bénéficie du lissage. La différence est surtout perceptible pendant le mouvement, où l'image saute, en plus d'apparaître crénelée si elle n'est pas lissée.

Figure 11.3
Comparaison des deux images.

Le document actuel traite une image chargée dynamiquement. Une déclinaison de cet exemple, adapté pour une image placée manuellement dans le scénario, est disponible dans le fichier "ch11_apiGraphisme_1b.fla".

À retenir

- Pour lisser une image, nous devons la placer en tant qu'objet Bitmap sur un conteneur et lui passer la propriété `smoothing` sur `true`.
- Le lissage augmente les ressources graphiques nécessaires pour l'exécution du programme. Il ne faut pas en abuser.

Lissage des graphismes vectoriels

Tout comme pour les images bitmap, il est possible d'optimiser aussi l'affichage des formes graphiques vectorielles. Pour cela, nous utilisons la propriété `cacheAsBitmap` que nous définissons sur `true`.

Dans cette section, nous appliquons cette propriété dynamiquement sur un symbole qui contient plusieurs occurrences d'une forme graphique, importée d'Illustrator.

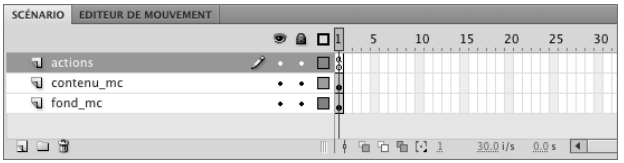
Exemples > ch11_apiGraphisme_2.fla

Dans le document "ch11_apiGraphisme_2.fla", sur la scène principale, apparaît un MovieClip. À l'intérieur, plusieurs occurrences d'une plume vectorielle sont superposées (voir Figure 11.4).

Figure 11.4
Aperçu du document.

Dans le scénario, seul le symbole `contenu_mc` apparaît entre le calque `fond_mc` et le calque `actions` (voir Figure 11.5).

Figure 11.5
Aperçu du scénario de la scène principale.



Dans la fenêtre Actions, une instruction applique la mise en cache sur le symbole conteneur `contenu_mc` :

```
//--------------- Mise en cache
contenu_mc.cacheAsBitmap=true;
```

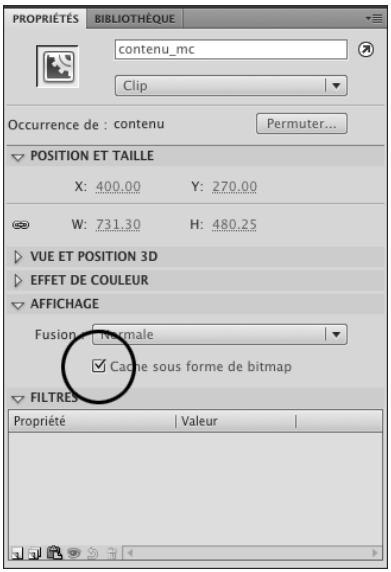
La mise en cache peut également être appliquée directement depuis l'Inspecteur de propriétés, en cliquant sur l'option intitulée Cache sous forme de bitmap (voir Figure 11.6).

Chaque mise en cache induit la création d'une image bitmap à partir des éléments vectoriels affichés dans un clip. En utilisant cette option, nous alourdissons considérablement le document. Quelques restrictions d'utilisation doivent donc être soulevées :

- Limitez le nombre de symboles qui utilisent cette propriété. Préférez rassembler les formes graphiques vectorielles dans un seul et unique symbole.

- N'animez pas le symbole dont le contenu est mis en cache (pas de rotation, pas de mise à l'échelle). Les déplacements simples sont possibles (animation par translation en X ou Y, ou déplacements avec `startDrag`). Car une déformation de l'image oblige le moteur de rendu à créer une nouvelle image à chaque modification. Vous ralentissez donc indéniablement les performances de l'application.
- Si l'image véhiculée doit apparaître à de nombreuses reprises, ou être superposée par des contenus déjà très gourmands en ressources, il est souhaitable d'activer cette option.
- Le texte contenu dans une zone défilante (avec un ascenseur par exemple) peut être mis en cache.
- Une illustration en arrière-plan d'un site, une carte du monde fixe et non zoomée, peut également être mise en cache.

Figure 11.6
Aperçu du document.



Info

Vous pouvez lisser les bords d'une forme vectorielle en activant, directement depuis l'Inspecteur de propriétés, et sur la forme elle-même, dans la catégorie Remplissage et trait, l'option Repères située à droite du champ Échelle. Cette option aligne automatiquement les formes graphiques sur des abscisses et des ordonnées entières. Elle évite donc l'affichage parfois bancal de certains contours.

À retenir

- *Il est possible d'optimiser les performances d'affichage pour les formes vectorielles. Nous utilisons pour cela la propriété* `cacheAsBitmap`.
- *La mise en cache des formes vectorielles ne doit cependant pas être systématique et ne doit concerner que les images non déformées, et non animées.*