# DESSINER ET REMPLIR DES FORMES EN ACTIONSCRIPT 3.0

*par Dominique DOLÉ*

*mars 2011*

# Table des matières

## 1 Fonctionnalités graphiques d'ActionScript 3

L'ActionScript permet la création de dessin vectoriel. Contrairement à l'ActionScript 2, l'ActionScript 3 permet, en plus des lignes et des courbes, de créer des formes (carré, rectangle, ellipse, cercle...) via des fonctionnalités intégrés. Ces fonctionnalités sont appelées API de dessin et permettent de créer des dessins vectoriels.

Grâce à cette API de dessin, il nous sera possible de dessiner dynamiquement la forme de nos boutons, de présenter facilement des données numériques sous forme de graphiques...

L'ActionScript 3, à travers son lot de nouvelles classes, va nous permettre de séparer correctement les différents travaux de dessin que nous aurons à réaliser.

### 1.1 La classe Graphics

La classe de base du dessin vectoriel est la classe Graphics. Elle comporte des méthodes et des propriétés permettant de tracer des lignes, des remplissages et des formes.

**Il est impossible d'instancier directement la classe Graphics.** On utilisera la **propriété graphics** des objets d'affichage qui prennent en charge le dessin. Ces objets seront **des objets de type Shape, Sprite ou des classes dérivées**.

### 1.2 La classe Shape

La classe Shape sert à créer des formes simples via l'API de dessin. Cette classe comprend donc une propriété graphics qui vous permet d'accéder aux méthodes de dessin de la classe Graphics.

La classe Shape hérite de la classe DisplayObject, ce qui signifie qu'on ne pourra dessiner qu'une seule forme dans un objet de ce type.

D'autre part, cette classe n'hérite pas de la classe InteractiveObject, ce qui signifie qu'**il ne sera pas possible de définir des événements de souris ou de clavier** sur ces objets.

## 2 Dessiner un rectangle uni

Le dessin d'un rectangle en ActionScript 3 va être réalisé en différentes étapes :

- Création d'une instance de type Shape
- Définition du style de remplissage (couleur et opacité)
- Dessin du rectangle
- Affichage

Les classes Shape et Graphics vont nous permettre de créer du dessin vectoriel en ActionScript.

### 2.1 Création d'une instance de type Shape

La classe Shape possède un constructeur qui ne prend aucun paramètre.

```
var rectangleUni:Shape = new Shape(); // Instance de l'objet d'affichage
```

### 2.2 Définition du style de remplissage

Avant de dessiner le rectangle, nous devons d'abord définir le **style du remplissage (couleur, opacité...)**.

C'est la méthode **beginFill()** de la classe Graphics qui va vous permettre de le faire. **1 seul paramètre de cette méthode est obligatoire**, c'est la couleur du remplissage au format hexadécimal. Nous allons définir un remplissage de couleur bleu clair et opaque.

**Nous appellerons la méthode beginFill() au travers de la propriété graphics de notre objet Shape.**

```
var rectangleUni:Shape = new Shape(); // Instance de l'objet d'affichage
rectangleUni.graphics.beginFill(0x86B1FB); // Le rectangle sera rempli de bleu
```

### 2.3 Dessin du rectangle

Pour dessiner le rectangle, nous allons utiliser la méthode **drawRect()** de la classe Graphics.
Cette méthode attend **4 paramètres (x, y, width, height)** :

- x : Number - Nombre indiquant la position horizontale par rapport au point d'alignement de l'objet d'affichage parent (en pixels).
- y : Number - Nombre indiquant la position verticale par rapport au point d'alignement de l'objet d'affichage parent (en pixels).
- width : Number - Largeur du rectangle (en pixels).
- height : Number - Hauteur du rectangle (en pixels).

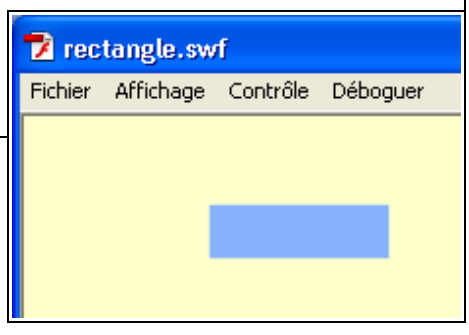
Nous allons dessiner un rectangle de 100 pixels de large par 30 pixels de haut, positionné en haut à gauche (0, 0) du shape.

```
var rectangleUni:Shape = new Shape(); // Instance de l'objet d'affichage
rectangleUni.graphics.beginFill(0x86B1FB); // Le rectangle sera rempli de bleu
rectangleUni.graphics.drawRect(0, 0, 100, 30); // dessin du rectangle avec la méthode drawRect(x, y, width, height)
```

### 2.4 Positionnement et affichage du rectangle dans la scène

Comme tous les objets d'affichage, nous positionnerons notre rectangle avec les **propriétés x et y** et nous l'afficherons avec la méthode **addChild()** .

```
var rectangleUni:Shape = new Shape(); // Instance de l'objet d'affichage
rectangleUni.graphics.beginFill(0x86B1FB); // Le rectangle sera rempli de bleu
rectangleUni.graphics.drawRect(0, 0, 100, 30); // dessin du rectangle avec la méthode drawRect(x, y, width, height)
// Positionnement et affichage du rectangle dans la séquence
rectangleUni.x = rectangleUni.y = 50;
addChild(rectangleUni);
```



## 3 Dessiner un rectangle avec un contour

Pour ajouter un contour à notre rectangle, nous allons utiliser la la méthode **lineStyle()** de la classe Graphics.

**1 seul paramètre de cette méthode est obligatoire**, c'est la largeur du trait. Elle sera exprimée par un entier qui indique l'épaisseur de la ligne en points. Les valeurs possibles varient entre 0 et 255.

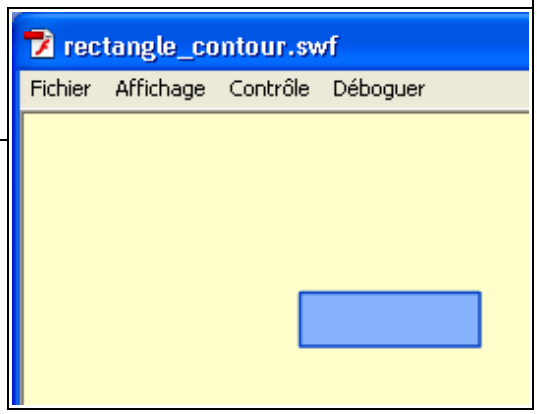
Le second paramètre, qui lui est facultatif, permet de définir la couleur de la ligne au format hexadécimal. Si aucune valeur n'est indiquée, la valeur par défaut est le noir (0x000000).

Le troisième paramètre, lui aussi facultatif, permet de définir l'opacité de la couleur de la ligne. Les valeurs varient entre 0 (transparent) et 1 (opaque). La valeur par défaut est 1.

Nous allons définir un trait d'une épaisseur de 2 points, de couleur marine et opaque.

**Nous appellerons la méthode lineStyle() au travers de la propriété graphics de notre objet Shape.**

```
var rectangleUniContour:Shape = new Shape(); // Instance de l'objet d'affichage
rectangleUniContour.graphics.beginFill(0x86B1FB); // Le rectangle sera rempli de bleu
rectangleUniContour.graphics.lineStyle(2, 0x0040C4); // Un contour de 2 pixels et de couleur marine est utilisé
rectangleUniContour.graphics.drawRect(0, 0, 100, 30); // Dessin du rectangle avec la méthode drawRect(x, y, width, height)
// Positionnement et affichage du rectangle dans la séquence
rectangleUniContour.x = rectangleUniContour.y = 100;
addChild(rectangleUniContour);
```



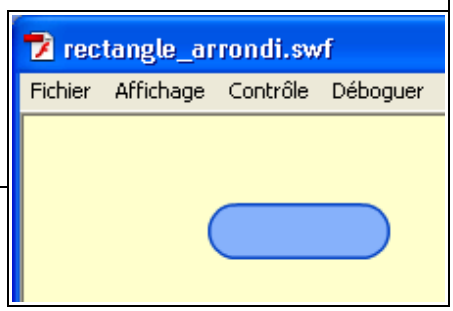

## 4 Dessiner un rectangle aux coins arrondis

Pour dessiner un rectangle aux coins arrondis, on utilisera la méthode **drawRoundRect()** de la classe graphics à la place de la méthode drawRect().

Cette méthode prend **2 paramètres en plus : la largeur de l'arrondi et la hauteur de l'arrondi**. Si la hauteur n'est pas spécifiée, elle prend la même valeur que la largeur.

Nous allons définir un arrondi d'angle homothétique de 30.

```
var rectangleArrondi:Shape = new Shape(); // Instance de l'objet d'affichage
rectangleArrondi.graphics.beginFill(0x86B1FB);// Le rectangle sera rempli de bleu
rectangleArrondi.graphics.lineStyle(2, 0x0040C4); // Un contour de 2 pixels et de couleur marine est utilisé
rectangleArrondi.graphics.drawRoundRect(0, 0, 100, 30, 30); // dessin du rectangle avec la méthode drawRect(x, y, width, height)
// Positionnement et affichage du rectangle dans la séquence
rectangleArrondi.x = 50;
rectangleArrondi.y = 50;
addChild(rectangleArrondi);
```



### *4.1 Paramètres de l'arrondi*

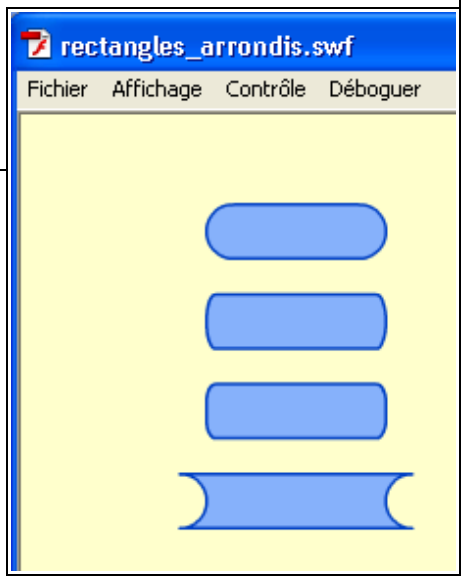
En modifiant les 2 paramètres de l'arrondi, on peut créer différentes variantes.

```
var rectangleArrondi:Shape = new Shape(); // Instance de l'objet d'affichage
rectangleArrondi.graphics.beginFill(0x86B1FB); // Le rectangle sera rempli de bleu
rectangleArrondi.graphics.lineStyle(2, 0x0040C4); // Un contour de 2 pixels et de couleur marine est utilisé
rectangleArrondi.graphics.drawRoundRect(0, 0, 100, 30, 30);// dessin du rectangle avec la méthode
drawRect(x, y, width, height) avec arrondi de 30 x 30
// Positionnement et affichage du rectangle dans la séquence
rectangleArrondi.x = 50;
rectangleArrondi.y = 50;
addChild(rectangleArrondi);

var rectangleArrondi2:Shape = new Shape(); // Instance de l'objet d'affichage
rectangleArrondi2.graphics.beginFill(0x86B1FB); // Le rectangle sera rempli de bleu
rectangleArrondi2.graphics.lineStyle(2, 0x0040C4); // Un contour de 2 pixels et de couleur marine est utilisé
rectangleArrondi2.graphics.drawRoundRect(0, 0, 100, 30, 10, 60); // dessin du rectangle avec la méthode
drawRect(x, y, width, height) avec arrondi de 10 x 60
// Positionnement et affichage du rectangle dans la séquence
rectangleArrondi2.x = 50;
rectangleArrondi2.y = 100;
addChild(rectangleArrondi2);

var rectangleArrondi3:Shape = new Shape(); // Instance de l'objet d'affichage
rectangleArrondi3.graphics.beginFill(0x86B1FB); // Le rectangle sera rempli de bleu
rectangleArrondi3.graphics.lineStyle(2, 0x0040C4); // Un contour de 2 pixels et de couleur marine est utilisé
rectangleArrondi3.graphics.drawRoundRect(0, 0, 100, 30, 10, 20);// dessin du rectangle avec la méthode
drawRect(x, y, width, height) avec arrondi de 10 x 20
// Positionnement et affichage du rectangle dans la séquence
rectangleArrondi3.x = 50;
rectangleArrondi3.y = 150;
addChild(rectangleArrondi3);

var rectangleArrondi4:Shape = new Shape(); // Instance de l'objet d'affichage
rectangleArrondi4.graphics.beginFill(0x86B1FB); // Le rectangle sera rempli de bleu
rectangleArrondi4.graphics.lineStyle(2, 0x0040C4); // Un contour de 2 pixels et de couleur marine est utilisé
rectangleArrondi4.graphics.drawRoundRect(0, 0, 100, 30, -30, 30);// dessin du rectangle avec la méthode
drawRect(x, y, width, height) avec arrondi de - 30 x 30
// Positionnement et affichage du rectangle dans la séquence
rectangleArrondi4.x = 50;
rectangleArrondi4.y = 200;
addChild(rectangleArrondi4);
```



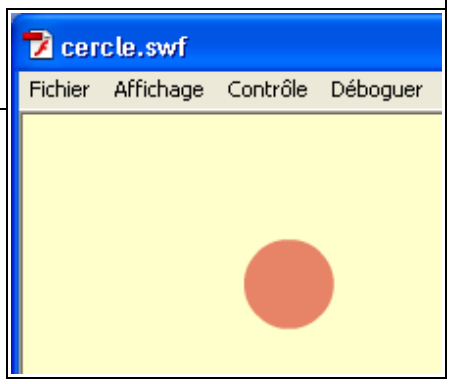
## 5 Dessiner un cercle

Pour dessiner un cercle, on utilisera la méthode **drawCircle()** de la classe Graphics.
Cette méthode attend **3 paramètres (x, y, radius)** :

- x : Number - Coordonnée x du centre du cercle par rapport au point d'alignement de l'objet d'affichage parent (en pixels).
- Y : Number - Coordonnée y du centre du cercle par rapport au point d'alignement de l'objet d'affichage parent (en pixels).
- Radius : Number - Rayon du cercle (en pixels).

Nous allons créer un cercle rouge transparent à 50%.

```
var cercle:Shape = new Shape(); // Instance de l'objet d'affichage
cercle.graphics.beginFill(0xCF0903, 0.5); // Le cercle sera rempli de rouge, transparent à 50%
cercle.graphics.drawCircle(25, 25, 25); // Dessin du cercle avec la méthode drawCircle(x, y, rayon)
// Positionnement et affichage du cercle dans la séquence
cercle.x = cercle.y = 70;
addChild(cercle);
```

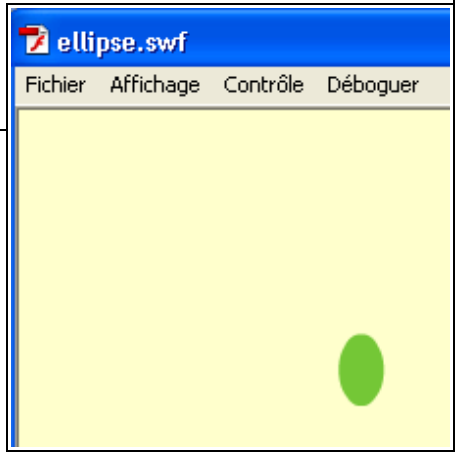



## 6 Dessiner une ellipse

Pour dessiner une ellipse, on utilisera la méthode **drawEllipse()** de la classe Graphics.
Cette méthode attend **4 paramètres (x, y, width, height)** :

- x : Number - Coordonnée x du centre de l'ovale par rapport au point d'alignement de l'objet d'affichage parent (en pixels).
- Y : Number - Coordonnée x du centre de l'ovale par rapport au point d'alignement de l'objet d'affichage parent (en pixels)
- width : Number - Largeur de l'ellipse (en pixels)
- height : Number - Hauteur de l'ellipse (en pixels)

```
var ellipse:Shape = new Shape() ; // Instance de l'objet d'affichage
ellipse.graphics.beginFill(0x46B504, 0.75); // L'ellipse sera rempli de vert, transparent à 75%
ellipse.graphics.drawEllipse(25, 25, 25, 40); // Dessin de l'ellipse avec la méthode drawEllipse(x, y, width,
height)
// Positionnement et affichage de l'ellipse dans la séquence
ellipse.x = ellipse.y = 100;
addChild(ellipse);
```



## 7 Dessiner des formes plus complexes

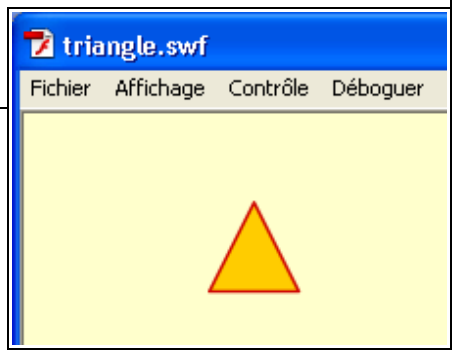
Pour dessiner une forme plus complexe, nous allons créer une **succession de droite ou de courbes**.

### 7.1 Dessiner un triangle

Nous allons créer un triangle avec un remplissage uni et un contour.

Pour tracer des droites, **nous appellerons la méthode lineTo() au travers de la propriété graphics de notre objet Shape.**

```
var triangle:Shape = new Shape(); // Instance de l'objet d'affichage
triangle.graphics.beginFill(0xFFCC00); // Le triangle sera rempli d'orange
triangle.graphics.lineStyle(2, 0xCF0903); // Contour rouge de 2 pixels
triangle.graphics.moveTo(25, 0); // Sommet du triangle
triangle.graphics.lineTo(50, 50); // Point bas droite
triangle.graphics.lineTo(0, 50); // Point bas gauche
triangle.graphics.lineTo(25, 0); // Retour au point de départ pour fermeture du tracé
// Positionnement et affichage du triangle dans la séquence
triangle.x = triangle.y = 50;
addChild(triangle);
```

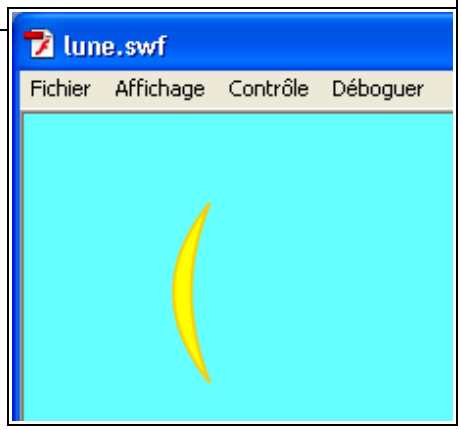



### 7.2 Dessiner une lune

Nous allons créer une lune avec un remplissage uni et un contour.

Pour tracer des courbes, **nous appellerons la méthode curveTo() au travers de la propriété graphics de notre objet Shape.**

```
var lune:Shape = new Shape(); // Instance de l'objet d'affichage
lune.graphics.beginFill(0xFFFF00); // La lune sera remplie de jaune
lune.graphics.lineStyle(2, 0xFFCC00); // Contour orange de 2 pixels
lune.graphics.moveTo(50, 50); // Position de départ
lune.graphics.curveTo(10, 100, 50, 150); // 1ère courbe pour aller à la position de destination (50, 150)
lune.graphics.curveTo(30, 100, 50, 50); // 2ème courbe pour retourner à la position de départ (50, 50)
addChild(lune);
```



## 8 Créer un remplissage dégradé

Jusqu'à présent, nous avons créé des remplissages utilisant des couleurs unies. La classe Graphics permet aussi de créer des dégradés.

Nous allons pour cela utiliser la méthode **beginGradientFill()**.

Cette méthode reçoit **4 paramètres obligatoires : type, couleur, alpha et ratio des couleurs**.

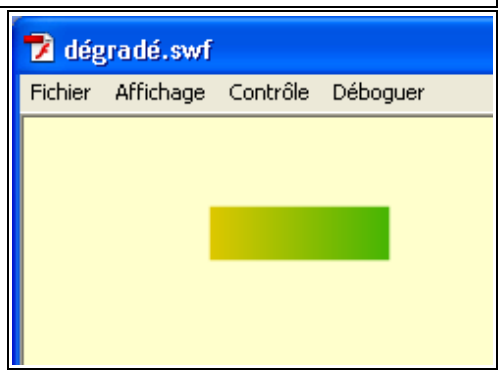
- Le premier paramètre spécifie le type de dégradé à créer. Les 2 valeurs possibles sont des constantes de la classe GradientType : GradientType.LINEAR pour un dégradé linéaire et GradientType.RADIAL pour un dégradé radial.
- Le second paramètre indique le tableau de valeurs colorimétriques à utiliser. Dans un dégradé linéaire, les couleurs sont organisées de gauche à droite. Dans un dégradé radial, les couleurs sont organisées de l'intérieur à l'extérieur. L'ordre des couleurs dans le tableau représente l'ordre dans lequel elles sont tracées dans le dégradé.
- Le troisième paramètre indique les valeurs de transparence alpha pour les couleurs du tableau des couleurs.
- Le quatrième paramètre spécifie les rapports, c'est-à-dire l'importance de chaque couleur dans le dégradé. La plage de valeurs possible va de 0 à 255. Ces valeurs représentent la position au sein du dégradé : 0 représente le début du dégradé, et 255 la fin. Cette plage de rapports doit augmenter séquentiellement et comporter le même nombre d'éléments que les tableaux des couleurs et des valeurs alpha spécifiés comme second et troisième paramètres.

Nous allons créer un dégradé linéaire de trois couleurs : rouge au début (0xCF0903), orange au milieu (0xFFCC00) et vert à la fin (0x46B504).

Les trois couleurs seront opaques : [1, 1, 1].

Les couleurs ne seront pas réparties de façon uniforme, le vert sera la couleur dominante. Le rouge démarrera à gauche (0), le vert sera à droite du dégradé (255) et l'orange sera au $\frac{1}{4}$ du dégradé (100). [0, 100, 255].

```
var rectangleDegrade:Shape = new Shape(); // Instance de l'objet d'affichage
rectangleDegrade.graphics.beginGradientFill(GradientType.LINEAR, [0xCF0903, 0xFFCC00, 0x46B504], [1, 1, 1], [0, 100, 255]); // Dégradé linéaire de trois couleurs
rectangleDegrade.graphics.drawRect(0, 0, 100, 30); // Dessin du rectangle avec la méthode drawRect(x, y, width, height)
// Positionnement et affichage du rectangle dans la séquence
rectangleDegrade.x = 200;
rectangleDegrade.y = 50;
addChild(rectangleDegrade);
```




Nous remarquons que par défaut le dégradé n'est pas étiré sur la largeur de la ligne (le rouge n'apparait pas, ...).

Pour remédier à ce problème, nous allons définir le **5ème paramètre** (**matrix**) de la méthode.

Ce paramètre défini une matrice de transformation du dégradé et accepte une instance de la classe Matrix.

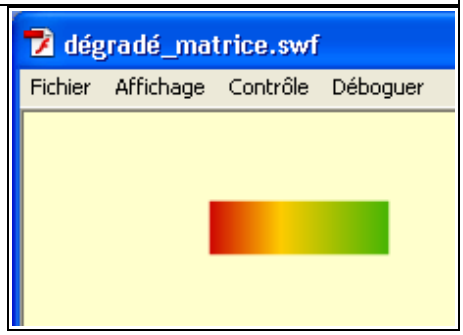
### *8.1 Création d'un objet de type Matrix pour le dégradé*

La classe **Matrix** représente une matrice de transformation que l'on peut utiliser pour appliquer diverses transformations graphiques à un objet d'affichage.

Nous allons utiliser la méthode **createGradientBox()** de la classe Matrix pour définir **la largeur et la hauteur du dégradé**.

Nous allons définir la largeur du dégradé sur 100, valeur correspondant à la largeur du rectangle et la hauteur sur 30, valeur correspondant à la hauteur du rectangle.

```
var rectangleDegrade:Shape = new Shape(); // Instance de l'objet d'affichage
var matrice:Matrix = new Matrix(); //Matrice de transformation du dégradé
matrice.createGradientBox(100, 30); // Largeur et épaisseur du dégradé
rectangleDegrade.graphics.beginGradientFill(GradientType.LINEAR, [0xCF0903, 0xFFCC00, 0x46B504], [1, 1, 1], [0, 100, 255, matrice]); // Dégradé linéaire de trois couleurs avec utilisation d'une matrice de transformation
rectangleDegrade.graphics.drawRect(0, 0, 100, 30); // Dessin du rectangle avec la méthode drawRect(x, y, width, height)
// Positionnement et affichage du rectangle dans la séquence
rectangleDegrade.x = 200;
rectangleDegrade.y = 50;
addChild(rectangleDegrade);
```




### *8.2 Répartition des couleurs dans le dégradé*

En faisant varier le quatrième paramètre (rapport) , son fait varier l'importance de chaque couleur dans le dégradé.

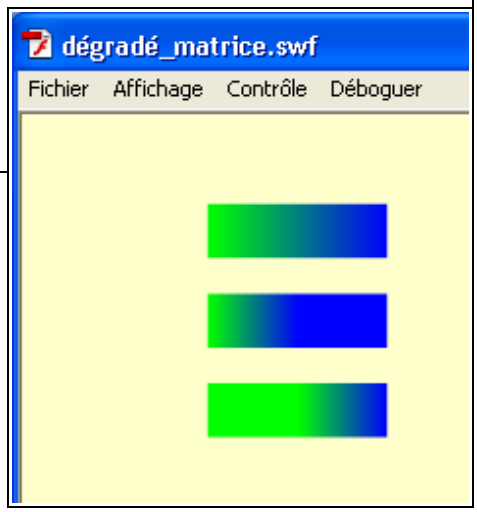
La plage de valeurs possible va de 0 à 255. Ces valeurs représentent la position au sein du dégradé : 0 représente le début du dégradé, et 255 la fin.

```
var matrice:Matrix = new Matrix(); //Matrice de transformation du dégradé
matrice.createGradientBox(100, 30); // Largeur et épaisseur du dégradé

var rectangleDegrade1:Shape = new Shape(); // Instance de l'objet d'affichage
rectangleDegrade1graphics.beginGradientFill(GradientType.LINEAR, [0x00FF00, 0x0000FF,], [1, 1], [0, 255] , matrice); // Dégradé linéaire de 2 couleurs et rapports de 0 et 255
rectangleDegrade1.graphics.drawRect(0, 0, 100, 30); // Dessin du rectangle avec la méthode drawRect(x, y, width, height)
// Positionnement et affichage du rectangle dans la séquence
rectangleDegrade1.x = 50;
rectangleDegrade1.y = 50;
addChild(rectangleDegrade1);
```

```
var rectangleDegrade2:Shape = new Shape(); // Instance de l'objet d'affichage
rectangleDegrade2.graphics.beginGradientFill(GradientType.LINEAR, [0x00FF00, 0x0000FF,], [1, 1], [0, 127] , matrice); // // Dégradé linéaire de 2 couleurs et rapports de 0 et 127
rectangleDegrade2.graphics.drawRect(0, 0, 100, 30); // Dessin du rectangle avec la méthode drawRect(x, y, width, height)
// Positionnement et affichage du rectangle dans la séquence
rectangleDegrade2.x = 50;
rectangleDegrade2.y = 100;
addChild(rectangleDegrade2);

var rectangleDegrade3:Shape = new Shape(); // Instance de l'objet d'affichage
rectangleDegrade3.graphics.beginGradientFill(GradientType.LINEAR, [0x00FF00, 0x0000FF,], [1, 1], [127, 255] , matrice); // // Dégradé linéaire de 2 couleurs et rapports de 127 et 255
rectangleDegrade3graphics.drawRect(0, 0, 100, 30); // Dessin du rectangle avec la méthode drawRect(x, y, width, height)
// Positionnement et affichage du rectangle dans la séquence
rectangleDegrade3.x = 50;
rectangleDegrade3.y = 150;
addChild(rectangleDegrade3);
```




### *8.3 Réglage de l'angle du dégradé*

Nous pouvons également contrôler l'**angle du dégradé grâce au troisième paramètre** de la méthode createGradientBox().

Nous allons appliquer un angle de 45°.

```
var rectangleDegrade:Shape = new Shape(); // Instance de l'objet d'affichage
var matrice:Matrix = new Matrix(); // Matrice de transformation du dégradé
matrice.createGradientBox(100, 30, 45); // Largeur, épaisseur et angle du dégradé (45°)
rectangleDegrade.graphics.beginGradientFill(GradientType.LINEAR, [0xCF0903, 0xFFCC00, 0x46B504], [1, 1, 1], [0, 100, 255, matrice]); // Dégradé linéaire de trois couleurs avec utilisation d'une matrice de transformation
rectangleDegrade.graphics.drawRect(0, 0, 100, 30); // Dessin du rectangle avec la méthode drawRect(x, y, width, height)
// Positionnement et affichage du rectangle dans la séquence
rectangleDegrade.x = 200;
rectangleDegrade.y = 50;
addChild(rectangleDegrade);
```