8

Le multimédia

Au sommaire de ce chapitre

- Les images et les photographies
- L'audio et la vidéo
- Le dessin et les animations
- Pour aller plus loin

Si l'iPhone est un grand communiquant, autant à l'aise dans les fonctions téléphoniques de base que dans l'interaction avec des services extérieurs ou des données associées à votre application, c'est aussi un mobile à l'incroyable potentiel multimédia. Appareil photo, baladeur dans la veine de l'iPod, lecture d'images sur lesquelles on zoome en écartant deux doigts, affichage de séquences vidéos, animations en 3D, etc. Les développeurs les plus ambitieux disposent d'une palette très large de fonctions pour étendre le champ de leurs applications et offrir une expérience plus riche à leurs utilisateurs. L'iPad embrasse encore davantage cet élan multimédia, haute résolution oblige : les applications de retouche photographique sont légion, sans parler de l'utilitaire de parcours de la photothèque qui permet de manipuler ses albums ou d'agrandir des clichés du bout des doigts, ou des nombreux jeux qui figurent à son catalogue. Des vastes encyclopédies richement illustrées aux clones de Photoshop en passant par les innombrables lecteurs vidéo, l'utilisation de ressources multimédias n'est plus réservée aux applications les plus ambitieuses : tous les développeurs y recourent régulièrement, quel que soit le sujet de leurs projets. Muni d'un double capteur vidéo, l'iPad 2 renforce encore le lien étroit qu'entretiennent

les développeurs avec le multimédia au sens large ; chaque jour, de nouvelles applications qui utilisent de manière originale les capacités de la tablette d'Apple signent leur entrée sur l'App Store.

À l'instar de tous les composants du kit de développement de l'iOS, la gestion des données multimédias obéit à un principe simple et vous permet de manipuler rapidement ce type de ressources. Comme au chapitre précédent, il est possible de stocker l'ensemble de ces données dans la sandbox de votre application afin de compacter toutes les ressources dont elle a besoin.

Mais avant de foncer tête baissée parmi ces possibilités et de parsemer vos projets de fichiers MP3 ou d'images en grand format, réfléchissez à deux fois. Comme nous l'avons vu, les applications pour iPhone visent avant tout à servir un rôle simple et précis. Les utilisateurs doivent être capables d'interroger immédiatement votre application et d'obtenir un résultat en moins de vingt secondes. Ne cédez donc pas envers et contre tout aux sirènes du multimédia, en bardant votre interface de contrôles superflus ou d'une bande-son en arrière-plan ! Mal exploité, ce type de données ne se contente pas d'accaparer inutilement l'attention de l'utilisateur ; il consomme également des ressources considérables jusqu'à faire planter le mobile d'Apple dans le pire des cas. Pour l'heure, l'iPad est moins concerné par ces précautions d'usage : son large écran autorise une meilleure immersion dans un environnement graphique riche et vous pouvez ainsi faire preuve d'une moindre sobriété. Les développeurs des encyclopédies de poche l'ont bien compris, à l'instar de l'application *Atomes* qui détaille la table périodique des éléments à travers une interface constellée de modèles et d'animations en trois dimensions (voir Figure 8.1).

Vous voilà mis en garde. L'iPhone et l'iPad peuvent d'ores et déjà rendre des services considérables avec les éléments de base du kit de développement : on ne reprochera jamais sa trop grande sobriété à une application, si elle remplit parfaitement le rôle qui lui a été fixé. Effectuez, en revanche, un usage intensif de ce type de ressources si votre projet s'articule précisément autour de la photographie ou de la musique. Lecteurs multimédias, retouches photographiques, création audio et gestion d'albums : les utilisateurs qui téléchargent ce type d'applications savent à quoi s'attendre et ils seront déçus de ne pas profiter d'un environnement riche ou de ne pas plonger dans de saisissants clichés à manipuler du bout des doigts.

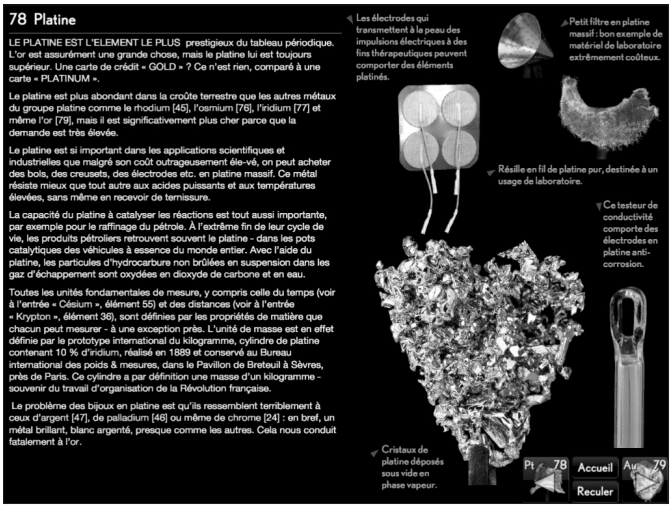

Figure 8.1
L'application Atomes *sur iPad brille par l'originalité de son interface et ses nombreuses ressources iconographiques.*

À titre d'exemple, l'application du Musée du Louvre fait figure de modèle et combine intelligemment tous les atouts de l'iPhone ou de l'iPad (voir Figure 8.2). Le contenu éditorial est riche et s'appuie sur des contrôles standard, en particulier des libellés, des champs de texte et des boutons. Toutes les vues s'enrichissent harmonieusement d'images ou de séquences vidéo et vous parcourez les collections en zoomant sur les détails d'une sélection d'œuvres. Si vous envisagez d'enrichir votre propre application de ce type d'éléments, prenez cet exemple en modèle. L'usage du multimédia y est constant, mais il se justifie sans cesse : aucune image n'est là par hasard et ces ressources soutiennent un contenu éditorial très riche. La Réunion des Musées Nationaux a d'ailleurs pris l'habitude de décliner chacune des grandes expositions parisiennes en une application de cet acabit, qui agit comme un programme interactif haut en couleurs.

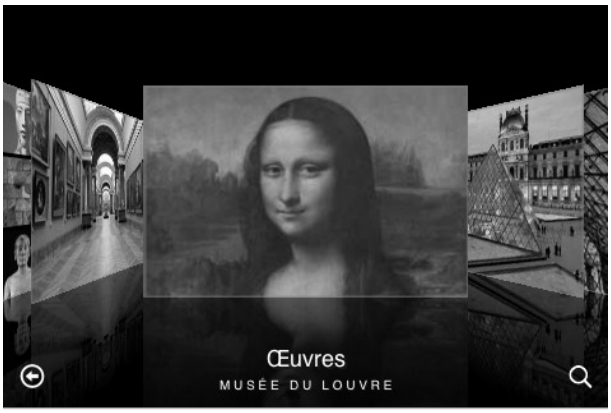

Figure 8.2
L'application du Musée du Louvre.

L'utilisation de l'appareil photo de l'iPhone ou de l'iPad 2 fait figure de cas particulier. Il ne s'agit pas d'imposer des images à l'utilisateur, mais bien de lui proposer d'utiliser le capteur de son mobile pour prendre le sujet de son choix en flagrant déclic, avant de manipuler éventuellement le résultat avec votre application. La tâche est clairement identifiée et il est le seul à décider des éléments figurant sur l'interface de son téléphone. L'intérêt artistique de ces éléments visuels ne vous incombe donc pas ! De nombreuses applications mettent ainsi en place un bouton déclenchant une photo ; c'est notamment le cas de la liste de tâches *ToDo*, qui vous permet d'archiver les facturettes liées à vos achats courants ou d'*UpCode* grâce auquel vous photographiez et convertissez les codes FlashCode présents sur certaines publicités (voir Figure 8.3).

Les éléments sonores sont moins envahissants et plus discrets. On les réservera essentiellement aux jeux vidéo et éventuellement aux messages d'alertes, afin d'attirer l'attention de l'utilisateur comme sur une application bureautique. Si vous envisagez de programmer un lecteur multimédia, inspirez-vous de l'application iPod installée par défaut – s'il paraît prétentieux de rivaliser avec cet auguste compétiteur, si complet et ergonomique, vous avez en revanche la possibilité de spécialiser votre lecteur dans certains formats ou genres musicaux. Là encore, vous pouvez profiter de l'immense variété des contrôles de l'iOS pour offrir une nouvelle manière d'écouter sa musique préférée. C'est notamment le pari du groupe britannique Snow Patrol qui propose une application officielle très déroutante dans laquelle vous écoutez des extraits des derniers albums tout en dépliant des origamis (voir Figure 8.4). Un vrai concept ! Par ailleurs, sachez

que vous pouvez faire appel aux fonctions internes de l'iPhone et de l'iPad et accéder à leur microphone afin d'enregistrer l'environnement extérieur. De nombreux dictaphones numériques de poche déferlent ainsi sur l'App Store d'iTunes. *Audiotorium* en est l'une des plus originales, sur iPad : cet outil vous permet de prendre des notes en classe ou lors d'une réunion, tout en les agrémentant d'un enregistrement audio du maître de séance (voir Figure 8.5).

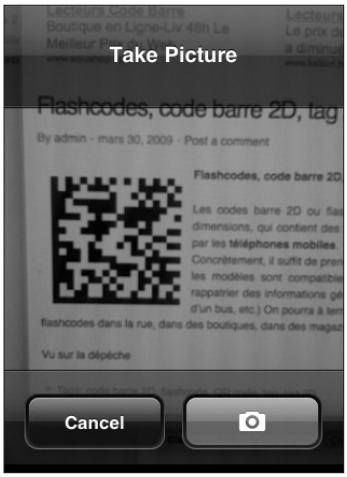

Figure 8.3
Grâce à UpCode*, vous prenez en photo les codes barre "FlashCodes" qui sont automatiquement convertis.*

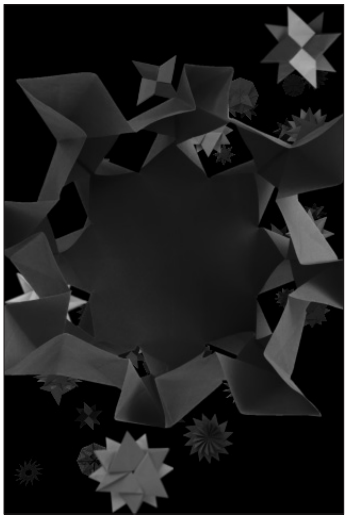

Figure 8.4
L'application officielle du groupe Snow Patrol déclenche des sons en fonction des pressions de l'utilisateur.

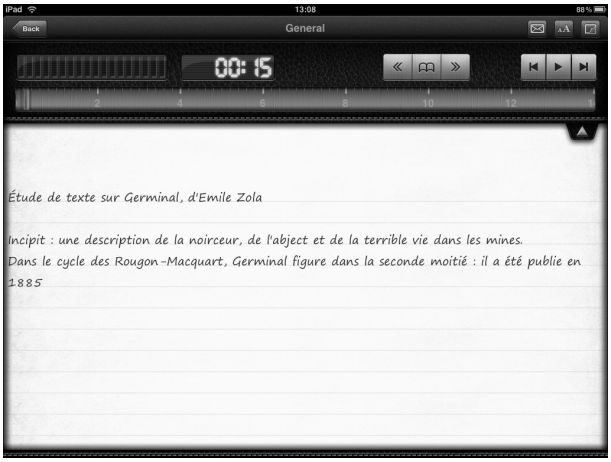

Figure 8.5
Audiotorium *s'adresse aux étudiants et leur permet de prendre des notes de cours tout en enregistrant les paroles du professeur.*

Passons sans plus tarder à la pratique et voyons ensemble comment les ressources multimédias contribuent à enrichir votre application.

Les images et les photographies

Afficher des images

Vous avez appris au Chapitre 6 à manipuler de petites illustrations sous Interface Builder et à les charger en guise de logo ou d'arrière-plan. Le chargement, puis le positionnement d'images en Objective-C, obéissent au même principe général : vous créez une vue d'image à travers une instance de la classe `UIImageView` et vous chargez les données correspondantes avec une instance de la classe `UIImage`. Il vous suffit ensuite d'ajouter la vue d'image à la vue principale pour faire apparaître l'illustration à l'écran. L'image en elle-même peut être stockée au sein de votre application, chargée depuis la photothèque de l'utilisateur ou téléchargée depuis une source externe.

Sous Xcode, déroulez le menu File, cliquez sur New Project et choisissez le modèle `View-based Application` – nous nous focaliserons ici sur l'iPhone. Baptisez votre application `Photo` et validez la création de votre nouveau projet. En guise de premier exemple, nous allons volontairement charger une image dont les dimensions dépassent celles de l'écran de l'iPhone. Préparez un fichier `photo.jpg` puis cliquez sur le dossier `Resources` en maintenant la touche Ctrl du clavier enfoncée. Choisissez Add, puis Existing Files et sélectionnez votre nouveau fichier. Il s'ajoute au projet. Nous allons créer une instance de la classe `UIImageView` et lui faire soutenir l'image que nous venons d'ajouter. Très simple, ce code ne comprend que quelques lignes. Ouvrez le fichier `PhotoViewController.m` et modifiez la déclaration de la méthode `loadView` comme au listing ci-dessous.

Listing 8.1 : *PhotoViewController.m*

```
- (void)loadView {
    UIImageView *vueImage = [[UIImageView alloc] initWithImage:
                                            [UIImage imageNamed:@"photo.jpg"]];
    self.view = vueImage;
    [vueImage release];
}
```

Enregistrez votre programme et compilez le projet (voir Figure 8.6).

Dans le Simulateur iOS, vous constatez que l'image dépasse largement de l'écran : elle s'affiche en taille réelle et seule sa partie supérieure gauche (sur 320 × 480 pixels) apparaît sur l'interface. La barre de statut recouvre même une partie de l'image, sur 20 pixels de hauteur. Pour y remédier, vous devez définir le cadre dans lequel s'inscrit votre vue d'image. Il existe plusieurs solutions pour aboutir à un tel résultat et redimensionner à la volée le cliché : remplacez la déclaration de `loadView` comme au Listing 8.2.

Figure 8.6
L'image chargée dépasse de l'écran de l'iPhone.

Listing 8.2 : Redimensionner la zone d'affichage

```
- (void)loadView {
    UIImageView *vueImage = [[UIImageView alloc] initWithFrame:
                                    [[UIScreen mainScreen] applicationFrame]];
    [vueImage setImage:[UIImage imageNamed:@"photo.jpg"]];
    self.view = vueImage;
    [vueImage release];}
```

Relancez le Simulateur iOS et constatez le résultat : quelles que soient ses dimensions, votre image apparaît désormais en entier sur l'écran, y compris lorsque vous testez l'affichage sur un iPhone Retina (voir Figure 8.7).

La classe `UIScreen` vous permet de délimiter le cadre de votre application. Sa méthode `mainScreen` retourne l'objet représenté par l'écran de l'iPhone et la propriété `applicationFrame` correspond exactement au cadre de votre application, soit l'écran entier du mobile sans sa barre de statut (320 × 460 pixels ou 640 × 920 sur un écran Retina). Dans notre exemple, nous avons ainsi réduit la vue d'image à ces dimensions et la photographie s'adapte au canevas en conséquence. Mais voilà : si votre image n'obéit pas au même ratio que l'iPhone, elle va paraître déformée. Grâce à la propriété `contentMode`, vous définissez le type de redimensionnement du contenu de la vue. Il existe treize constantes différentes (voir Tableau 8 .1) ; dans notre exemple, nous allons utiliser `UIViewContentModeScaleAspectFit` afin de préserver l'aspect de l'image. Modifiez une nouvelle fois la déclaration de `loadView` avec le code figurant au Listing 8.3.

Figure 8.7
L'image chargée s'adapte désormais aux dimensions de l'écran de l'iPhone.

Listing 8.3 : Préserver l'aspect de l'image

```
- (void)loadView {
    UIImageView *vueImage = [[UIImageView alloc] initWithFrame:
                                        [[UIScreen mainScreen] applicationFrame]];
    [vueImage setImage:[UIImage imageNamed:@"photo.jpg"]];
    vueImage.contentMode = UIViewContentModeScaleAspectFit;
    self.view = vueImage;
    [vueImage release];
}
```

Cette fois, l'image conserve ses propriétés d'origine et se voit simplement contrainte dans le conteneur de 320x460 pixels. Si elle est au format portrait, sa hauteur est redimensionnée à 460 pixels, et si elle est au format paysage, c'est sa largeur qui est réduite à 320 pixels (les dimensions sont automatiquement doublées sur un écran Retina). Son aspect est préservé et des zones transparentes apparaissent éventuellement autour de l'image afin de meubler l'espace vacant. Reportez-vous au Tableau 8.1 pour découvrir les autres constantes de la propriété `contentMode` et leur rôle. Essayez de modifier l'exemple précédent afin de tester toutes vos possibilités (voir Figure 8.8).

Tableau 8.1. Les constantes de la propriété *contentMode*

Constante	*Rôle*
`UIViewContentModeScaleToFill`	L'image occupe tout l'espace de la vue et son aspect est éventuellement modifié (mode par défaut).
`UIViewContentModeScaleAspectFit`	L'image adapte ses dimensions en préservant son aspect.
`UIViewContentModeScaleAspectFill`	L'image occupe tout l'espace de la vue en préservant son aspect : certaines zones peuvent ainsi être rognées.
`UIViewContentModeRedraw`	Redessine l'image lorsque les dimensions de la vue changent.
`UIViewContentModeCenter`	L'image est centrée au milieu de la vue, sans adapter ses dimensions.
`UIViewContentModeTop`	L'image est centrée en haut de la vue, sans adapter ses dimensions.
`UIViewContentModeBottom`	L'image est centrée en bas de la vue, sans adapter ses dimensions.
`UIViewContentModeLeft`	L'image s'aligne à gauche de la vue, sans adapter ses dimensions.
`UIViewContentModeRight`	L'image s'aligne à droite de la vue, sans adapter ses dimensions.
`UIViewContentModeTopLeft`	L'image est chargée à partir de son bord supérieur gauche, sans redimensionnement.
`UIViewContentModeTopRight`	L'image est chargée à partir de son bord supérieur droit, sans redimensionnement.
`UIViewContentModeBottomLeft`	L'image est chargée à partir de son bord inférieur gauche, sans redimensionnement.
`UIViewContentModeBottomRight`	L'image est chargée à partir de son bord inférieur droit, sans redimensionnement.

L'intérêt des propriétés de redimensionnement

Ce premier exemple illustre la nécessité de préparer soigneusement les images que vous affichez à travers votre application. À l'aide de votre logiciel de création graphique préféré, n'hésitez pas à rogner ou à changer l'échelle de vos illustrations de telle sorte qu'elles épousent le ratio de l'écran de l'iPhone : vous limiterez ainsi l'apparition

de zones transparentes. Vous réduirez au passage la taille du fichier correspondant. À moins de prévoir l'utilisation de zooms sur l'image, évitez d'intégrer des fichiers trop volumineux au sein de votre projet. Par ailleurs, les propriétés de `contentMode` *présentent un grand intérêt lors de la rotation de l'écran de l'iPhone ou de l'iPad ; lorsque l'utilisateur pivote son mobile ou sa tablette en mode paysage, la vue contenant l'image est redessinée et vous vous assurez ainsi que celle-ci n'est pas tronquée à l'écran. L'application* Photos *préinstallée sur l'iPhone et l'iPad exploite précisément cette solution et utilise la propriété* `UIViewContentModeScaleAspectFit` *pour préserver l'aspect de vos clichés et les adapter quel que soit le mode d'affichage choisi. Nous y reviendrons largement au cours du Chapitre 10.*

Figure 8.8
La constante `UIViewContentModeScaleAspectFill` *oblige parfois à rogner l'image d'origine.*

Adapter les dimensions de la vue d'image

Au cours des chapitres précédents, nous avions essentiellement défini les dimensions des instances de la classe `UIImageView` à la souris, à travers Interface Builder. Si la manipulation est simple et intuitive, elle peut parfois s'avérer rébarbative dans le cas du chargement d'images. Vous devez en effet définir des outlets dans le contrôleur de la vue puis les associer à vos éléments d'interface. Vous effectuez ainsi un incessant va-et-vient entre Xcode et Interface Builder. En considérant l'expérience dont vous disposez aujourd'hui, vous constatez qu'il est parfois plus rapide de se passer d'Interface Builder et de modifier la méthode `loadView` afin de définir directement vos éléments d'interface en Objective-C, comme nous l'avons fait à travers notre premier exemple. Attention toutefois : en procédant de la sorte, vous ne vous inscrivez plus pleinement dans le schéma MVC dans la mesure où la vue ne s'isole pas dans un fichier nib. Mais pour des programmes simples, vous aboutissez rapidement à un résultat.

Nous avons initialisé notre instance de la classe `UIImageView` à travers la méthode `initWithFrame` en adaptant ses dimensions à celles de l'écran de l'iPhone. Dans le même ordre d'idée, vous avez la possibilité de positionner très précisément vos vues d'images à l'aide de la structure `CGRect`. Comme son nom l'indique, cet objet appartient au framework Core Graphics dont Quartz est le moteur graphique. On retrouve naturellement ce framework sous Mac OS X : il est responsable de l'affichage de l'interface Aqua du système. Le framework Core Graphics fait partie de la couche Media de Cocoa Touch, au même titre que Core Audio et Core Animation sur lesquels nous reviendrons dans la suite de ce chapitre.

La structure `CGRect` définit un rectangle aux dimensions et au positionnement précis. En utilisant la propriété `frame` de tous les types de vues, vous modifiez ainsi l'interface de votre application en quelques lignes d'Objective-C. Vous devez impérativement vous familiariser avec cette technique afin de gérer au mieux l'auto-rotation de l'iPhone ou de l'iPad et le repositionnement d'éléments d'interface, comme nous le verrons par la suite. Par ailleurs, cette méthode vous oblige à maîtriser au mieux le système de coordonnées du kit de développement – un atout précieux lorsque vous souhaitez disposer des onglets, des boutons ou une barre de navigation sur une interface très complexe !

Sur l'écran de l'iPhone ou de l'iPad, les coordonnées s'expriment en pixels et l'origine correspond au coin supérieur gauche. L'axe des abscisses (`x`) est horizontal et l'axe des ordonnées (`y`) est vertical, que l'on tienne le mobile en mode portrait ou paysage. En considérant que l'écran de l'iPhone fait 320 × 480 pixels dans le mode normal d'affichage (soit 640 × 960 sur l'écran Retina de l'iPhone 4, dont le nombre de pixels est doublé), on comprend ainsi le système de coordonnées :

- le coin supérieur droit correspond au point (320, 0) ;
- le coin inférieur gauche correspond au point (0, 480) ;
- le coin inférieur droit correspond au point (320, 480) ;

Il en va de même pour l'iPad, qui présente une résolution de 1024 × 768 en mode paysage :

- le coin supérieur droit correspond au point (1024, 0) ;
- le coin inférieur gauche correspond au point (0, 768) ;
- le coin inférieur droit correspond au point (1024, 768).

Vous pouvez ainsi vous repérer dans le plan et disposer précisément toutes les vues de votre application (voir Figures 8.9 et 8.10).

À ce titre, passons à la pratique en adaptant notre dernier exemple à un cas concret. Nous souhaitons disposer une légende en-dessous du cliché chargé précédemment : nous devons ainsi prévoir une zone pour abriter un libellé. La vue d'images se réduit en hauteur et nous ajoutons une instance de la classe `UIView` pour soutenir la légende. Elle contient elle-même une instance de la classe `UILabel` correspondant au libellé. Celle-ci dérive d'`UIView` et nous pouvons donc également préciser ses coordonnées à l'aide de la structure `CGRect`.

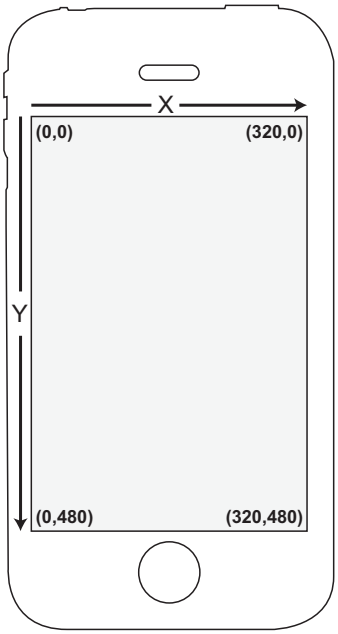

Figure 8.9
Le système de coordonnées de l'iPhone.

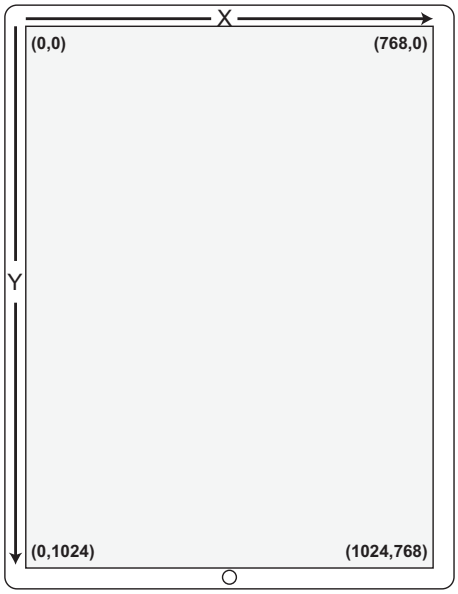

Figure 8.10
… et celui de l'iPad.

Afin d'illustrer le principe de chargement des vues, nous allons modifier cette fois la méthode `viewDidLoad` de notre contrôleur. Elle intervient juste après l'exécution de `loadView` : la vue est chargée mais elle n'est pas encore affichée. La méthode `viewWillAppear` vient ensuite. La vue est sur le point d'apparaître à l'écran : vous procédez éventuellement à vos ultimes réglages. À l'inverse, la méthode `viewDidAppear` est appelée après son affichage final. Il est intéressant d'utiliser `viewDidLoad` lorsque vous souhaitez changer les propriétés d'objets instanciés par la vue d'un fichier nib. Dans ce cas, la méthode `loadView` a en effet chargé tous les objets de la

vue : on les initialise avec du code inclus dans la méthode `viewDidLoad`. Si les traitements sont particulièrement lourds, vous limitez ainsi la consommation mémoire en distinguant le chargement des objets de leur code d'initialisation. Dans notre exemple, nous souhaitons ajouter des sous-vues à la vue principale : nous pouvons ainsi la désigner par `self.view` dans la mesure où elle a été chargée. Recopiez le code du Listing 8.4.

Listing 8.4 : Chargement d'une image avec une légende

```
- (void)viewDidLoad {

    // Préparation de la vue pour l'image
    UIImageView *vueImage = [[UIImageView alloc]
                                    initWithFrame:CGRectMake(0, 0, 320, 420)];
    [vueImage setImage:[UIImage imageNamed:@"photo.jpg"]];
    vueImage.contentMode = UIViewContentModeScaleAspectFill;

    // Préparation de la vue pour la légende
    UIView *vueLegende = [[UIView alloc] initWithFrame:CGRectMake(0, 420, 320, 40)];
    vueLegende.backgroundColor = [UIColor darkGrayColor];

    // Déclaration de la légende
    UILabel *legende = [[UILabel alloc] initWithFrame:CGRectMake(20, 5, 280, 30)];
    legende.text = @"Temple à proximité de Tokyo";
    legende.textAlignment = UITextAlignmentCenter;
    legende.textColor = [UIColor whiteColor];
    legende.backgroundColor = [UIColor clearColor];

    // Ajout de la vue de la légende
    [vueLegende addSubview:legende];
    [self.view addSubview:vueLegende];

    // Ajout de la vue de l'image
    [self.view addSubview:vueImage];

    [vueLegende release];
    [vueImage release];
    [legende release];
}
```

Enregistrez le projet puis compilez-le : l'interface principale de votre application affiche bien deux vues, l'image et la légende, avec un libellé positionné en bas de l'écran (voir Figure 8.11). Vous noterez au passage qu'en simulant un écran Retina (déroulez le menu Matériel et choisissez Appareil > iPhone Retina), les mêmes proportions s'appliquent strictement : vous pouvez ainsi adopter ce système de coordonnées, quelle que soit la version de l'iPhone que vous visez.