Содержание

Глава 5

Автоматический поворот и изменение размеров

Устройства iPhone и iPad, а также другие устройства, работающие под управлением системы iOS, представляют собой изумительные изобретения. Инженеры компании Apple сумели втиснуть максимум функциональных возможностей в очень маленькую коробочку. Например, в этих устройствах существует механизм, позволяющий использовать их как в портретной (вытянутой в высоту), так в альбомной (растянутой в ширину) ориентации, а также изменять эту ориентацию при повороте устройства. Превосходное проявление этой функциональной возможности, которая называется *автоматическим поворотом* (autorotation), можно увидеть в веб-браузере системы iOS Mobile Safari (рис. 5.1).

Рис. 5.1. Как и многие приложения для системы iOS, браузер Mobile Safari изменяет свое представление в зависимости от того, как именно пользователь держит устройство. Это позволяет максимально использовать доступное пространство на экране

В этой главе мы подробно обсудим механизм автоматического поворота. Начнем с обзора деталей этого механизма.

Механизм автоматического поворота

Режим автоматического поворота есть не у всех приложений. Некоторые приложения для устройства iPhone поддерживают только одну ориентацию. Например, фильмы можно просматривать только в альбомной ориентации, а контакты можно редактировать только в портретной. Однако это не относится к устройству iPad, для которого компания Apple рекомендует, чтобы практически все приложения (за исключением игр, разработанных для конкретного режима) поддерживали любую ориентацию.

Практически все собственные приложения для устройства iPad, разработанные компанией Apple, прекрасно работают в обоих режимах. Многие из них используют разные ориентации, для того чтобы продемонстрировать разные представления ваших данных. Например, приложения Mail и Notes используют альбомный режим, чтобы показать список объектов (папок, сообщений или заметок) слева и выбранный объект — справа, а портретная ориентация позволяет сконцентрировать внимание пользователя на деталях только что выбранного объекта.

Итак, если автоматический поворот расширяет возможности пользователя, добавьте его в свое приложение. К счастью, компания Apple проделала огромную работу, чтобы скрыть сложности автоматического поворота в системе iOS и в пакете UIKit, поэтому реализация этой функциональной возможности в ваших приложениях для системы iOS является довольно простой.

Автоматический поворот реализуется в контроллере представления, поэтому, если пользователь поворачивает устройство, активный контроллер представления получает запрос о том, готов ли он изменить ориентацию (как это сделать, вы узнаете в этой главе). Если контроллер представления отвечает положительно, ориентация окна и представления приложения, а также их размеры будут изменены.

В устройствах iPhone и iPod touch представление начинает работу в портретном режиме, имеющем 320 пикселей в ширину и 480 пикселей в высоту. В устройстве iPad портретный режим подразумевает 768 пикселей в ширину и 1024 пикселя в высоту. Если ваше приложение имеет строку состояния (status bar), то размер экрана, действительно доступного для вашего приложения, может быть уменьшен на 20 пикселей в вертикальном направлении. Строка состояния — это полоска высотой 20 пикселей, расположенная в верхней части экрана (см. рис. 5.1). На нее выводятся сила сигнала, время и заряд батареи.

Когда телефон переключается в альбомный режим, представление поворачивается вдоль окна приложения и изменяет размеры, чтобы заполнить все окно: 480 пикселей в ширину и 320 в высоту (iPhone и iPod touch) или 1024 пикселя в ширину и 768 пикселей в высоту (iPad). Как и прежде, размер экрана по вертикали, выделенный для приложений, имеющих строку состояния (а это большинство приложений), уменьшается на 20 пикселей.

ЗАМЕЧАНИЕ. Здесь у читателей может возникнуть вопрос: почему мы не упоминаем о дисплее Retina в устройствах iPhone 4 и iPod touch? Дисплей Retina — это маркетинговый термин компании Apple, которым она обозначает экраны с высоким разрешением на устройствах iPhone 4 и iPod touch. Разрешение этих экранов вдвое выше обычного: 640×960 пикселей. Благодаря весьма остроумной обработке графических изображений, реализованной компанией Appl, нас обычно не интересует проблема более высокого разрешения (если это нам действительно не требуется). Разрешение экрана дисплея Retina вдвое выше в обоих направлениях, но, создавая приложения на платформе Cocoa Touch, мы обычно имеем дело с обычными устройствами iPhone и iPod, имеющими разрешение 320×480 пикселей. Считайте это "виртуальным разрешением", которое система iOS автоматически отображает в реальное физическое разрешение. Более подробно мы поговорим об этом в главе14.

Большую часть по фактическому переносу пикселей на экране выполняет система iOS. Основная работа, которую должно выполнить ваше приложение, сводится к тому, чтобы обеспечить правильный внешний вид после изменения размеров экрана.

Ваше приложение может реализовать три варианта реакции на поворот устройства. Какой из них выполнить, зависит от сложности вашего интерфейса. В данной главе мы рассмотрим все три варианта.

В простых интерфейсах можно просто задать корректные атрибуты **автоматического изменения размеров** (autosize) всех объектов, образующих интерфейс. Атрибуты автоматического изменения размеров сообщают устройству, работающему под управлением системы iOS, как должны функционировать элементы управления, когда содержащее их представление изменяет размеры. Если вы ранее работали на платформе Cocoa под управлением операционной системы Mac OS X, то уже знакомы с этим подходом, поскольку именно он используется для того, чтобы определить поведение элементов управления среды Cocoa в ситуации, когда пользователь изменяет размеры окна, которому они принадлежат.

Автоматическое изменение размеров выполняется быстро и просто, но подходит не для всех приложений. Более сложные интерфейсы должны реагировать на изменение размеров иначе. Для более сложных представлений используются два дополнительных подхода. Один из них подразумевает ручную перестановку объектов при повороте представления. Второй подход предусматривает реализацию двух разных версий представления в программе Interface Builder — одной для портретного режима, а другой для альбомного. В обоих случаях необходимо выполнить замещение методов из класса `UIViewController` в классе контроллера вашего представления.

Для начала рассмотрим автоматическое изменение размеров.

Обработка поворота с помощью автоматического изменения размеров

Начните новый проект в среде Xcode и назовите его `Autosize`. Для этого приложения мы собираемся использовать хорошо знакомый нам шаблон View-based Application, а в качестве цели снова будем использовать устройство iPhone. Прежде чем скомпоновать графический пользовательский интерфейс в программе Interface Builder, необходимо сообщить системе iOS, что наше представление поддерживает автоматический поворот. Для этого модифицируем класс контроллера представления.

Настройка поддержки поворота

Открыв проект в среде Xcode, раскройте папку `Classes` и щелкните на файле `AutoSizeViewController.m`. В коде, который в нем уже содержится, вы увидите закомментированный метод `shouldAutorotateToInterfaceOrientation:`. Вскоре мы заменим эту закомментированную версию своим вариантом.

```
...
/*
// Этот метод следует заменить, чтобы разрешить ориентацию,
// отличную от портретной.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
        interfaceOrientation {
    // Возвращает YES для поддерживаемых ориентаций
```

```
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
*/
...
```

Этот метод описывает способ, с помощью которого система запрашивает у контроллера представления, поддерживает ли он переход в конкретную ориентацию. Устройство, работающее под управлением системы iOS, может осуществлять такую поддержку четырьмя способами.

- `UIInterfaceOrientationPortrait`
- `UIInterfaceOrientationPortraitUpsideDown`
- `UIInterfaceOrientationLandscapeLeft`
- `UIInterfaceOrientationLandscapeRight`

Для устройства iPhone шаблон проекта по умолчанию поддерживает единственную ориентацию: `UIInterfaceOrientationPortrait`. Если бы мы выбрали проект для устройства iPad, то по умолчанию шаблонная версия метода `shouldAutorotateToInterfaceOrientation:` была бы другой. В этом случае метод должен просто возвращать значение `YES`, поскольку компания Apple рекомендует, чтобы разработчики приложения для устройства iPad позволяли поворачивать устройство в любой из четырех вариантов ориентации.

Когда устройство, работающее под управлением системы iOS, изменит пространственную ориентацию, этот метод будет вызван из активного контроллера представления. Параметр `interfaceOrientation` будет содержать одно из четырех значений, указанных выше, и этот метод должен вернуть либо `YES`, либо `NO`, чтобы указать, следует ли повернуть окно приложения, чтобы учесть новую ориентацию. Поскольку каждый подкласс контроллера представления может по-разному реализовывать этот поворот, одни приложения могут поддерживать автоматический поворот некоторых из своих представлений, а другие нет.

Вы заметили, что некоторые системные константы в устройстве iPhone всегда начинаются с одних и тех же букв? Одна из причин, по которым имена `UIInterfaceOrientationPortrait`, `UIInterfaceOrientationPortraitUpsideDown`, `UIInterfaceOrientationLandscapeLeft` и `UIInterfaceOrientationLandscapeRight` начинаются со слова `UIInterfaceOrientation`, заключается в желании использовать технологию Code Sense, реализованную в среде Xcode.

СМЫСЛ КОДА

Возможно, вы заметили, что, когда вы набираете символы на клавиатуре, среда Xcode часто пытается дополнить слово, которое вы печатаете. Это — технология Code Sense в действии. Разработчики часто не могут запомнить все константы, определенные в системе, но могут запомнить имена групп, которым они принадлежат. Когда нужно указать ориентацию, просто наберите слово `UIInterfaceOrientation` (или даже `UIInterf`), а затем нажмите клавишу <Esc>, чтобы увидеть список имен, начинающихся с этих букв. (В настройках среды Xcode можно изменять клавишу <Esc> для открытия этого списка на другую.) Для просмотра списка можно использовать навигационные клавиши, а выбор осуществляется нажатием клавиши <Tab> или <Enter>. Этот способ намного быстрее просмотра значений в документации или заголовочных файлах.

Реализация этого метода, принятая по умолчанию, анализирует значение параметра `interfaceOrientation` и возвращает значение `YES` тогда и только тогда, когда он равен `UIInterfaceOrientationPortrait`, ограничивая приложение одной ориентацией и, по существу, отменяя автоматический поворот.

Для того чтобы получить возможность любой ориентации, измените метод так, чтобы он возвращал значение YES для любого полученного значения.

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
   (UIInterfaceOrientation)interfaceOrientation {
   return YES;
}
```

На самом деле, как указывалось ранее, именно так работает метод, когда вы создаете проект для устройства iPad, а не iPhone.

Для того чтобы поддержать некоторые, но не все виды ориентации, необходимо проверить значение параметра interfaceOrientation и вернуть значение YES для тех вариантов, которые мы хотим поддерживать, и значение NO для вариантов, которые поддерживать не хотим. Например, для того чтобы включить поддержку портретного и альбомного режимов в обоих направлениях, но не допустить поворот в портретном режиме вверх ногами, следует использовать следующий код:

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
    (UIInterfaceOrientation)interfaceOrientation {
    return (interfaceOrientation !=
    UIInterfaceOrientationPortraitUpsideDown);
}
```

Теперь изменим закомментированный метод shouldAutorotateToInterfaceOrientation:, чтобы он соответствовал предыдущей версии. Как правило, компания Apple не рекомендует использовать режим UIInterfaceOrientationPortraitUpsideDown в устройствах iPhone, поскольку, если телефон позвонит в момент, когда он ориентирован вверх ногами, он в таком положении и останется во время разговора. Это не относится к устройству iPad, в котором смысл выражения "вверх ногами" не так очевиден, и устройство может поворачиваться как угодно.

Сохраните проект. Теперь посмотрим на атрибуты автоматического изменения размера в nib-файле с помощью программы Interface Builder.

Проектирование интерфейса с помощью атрибутов автоматического изменения размеров

Находясь в среде Xcode, откройте папку Resources и дважды щелкните на файле AutosizeViewController.xib, чтобы открыть его в программе Interface Builder. Атрибуты автоматического изменения размеров имеют одно хорошее свойство — требуют очень мало кода. Нет необходимости указывать, какую ориентацию мы поддерживаем, как это требовалось в коде контроллера представления, а остальная реализация автоматического изменения размеров может быть осуществлена непосредственно в программе Interface Builder.

Для того чтобы увидеть, как это работает, перетащите шесть кнопок Round Rect Buttons из библиотеки на свое представление и разместите их так, как показано на рис. 5.2. Дважды щелкните на каждой кнопке и присвойте им заголовки. Мы пронумеровали их от 1 до 6.

А теперь посмотрим на то, что произойдет, если мы укажем поддержку автоматического поворота, но не зададим ни один атрибут автоматического изменения размеров.

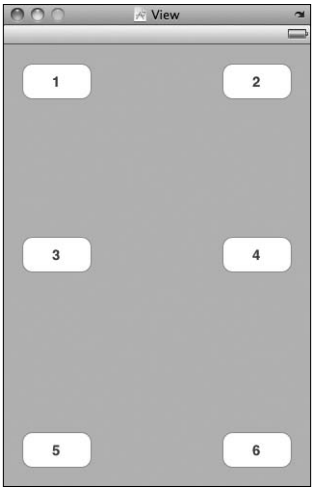

Рис. 5.2. Добавление шести кнопок в интерфейс

Во-первых, сохраните nib-файл. Затем перейдите к среде Xcode, соберите и запустите новое приложение. Когда начнет работу симулятор устройства iPhone, выберите команду Hardware⇨Rotate Left, которая будет имитировать перевод устройства iPhone в альбомный режим (рис. 5.3).

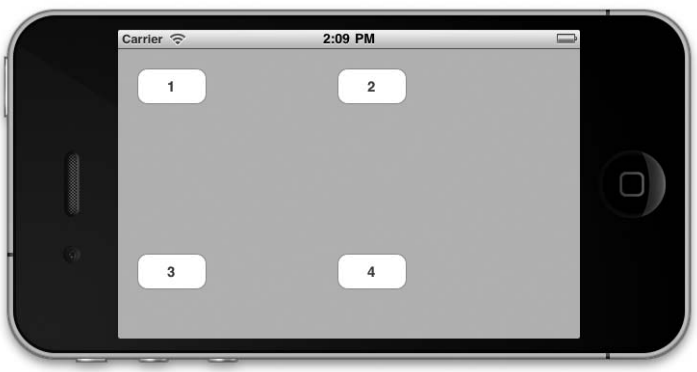

Рис. 5.3. Не слишком хорошо, не так ли? А где кнопки 5 и 6?

По умолчанию большинство элементов управления остаются там же, где были, сохраняя ориентацию по отношению к левому и верхнему краям экрана. Для некоторых элементов управления это вполне приемлемо. Например, верхняя левая кнопка с номером 1, вероятно, находится именно там, где вы и хотели. Однако остальные кнопки далеки от идеала.

Выйдите из симулятора и перейдите к программе Interface Builder. Давайте исправим графический пользовательский интерфейс, чтобы разумно адаптировать его к размерам экрана.

Использование атрибутов автоматического изменения размеров в окне Size Inspector

Щелкните один раз на левой верхней кнопке на вашем представлении, а затем нажмите комбинацию клавиш <⌘+3>, чтобы открыть окно инспектора размеров (size inspector), показанное на рис. 5.4.

Инспектор размеров (size inspector) позволяет задать **атрибуты автоматического изменения размеров** (autosize atributes) объекта. На рис. 5.5 показана часть окна инспектора размеров, управляющая атрибутами автоматического изменения размеров объекта.

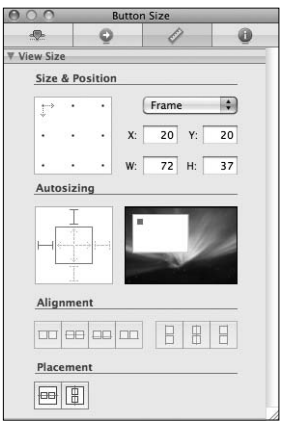

Рис. 5.4. Инспектор размеров позволяет задавать атрибуты автоматического изменения размеров

Рис. 5.5. Раздел Autosizing в окне инспектора размеров

Прямоугольник на рис. 5.5, *слева*, представляет собой именно то место, где на самом деле задаются атрибуты. Прямоугольник справа содержит небольшую анимацию, которая показывает, как объект ведет себя при изменении размеров.

Обратите внимание на то, что эта анимация воспроизводится только в те моменты, когда курсор перемещается по области анимации. Внутренний квадрат в левом прямоугольнике обозначает текущий объект. Если выбрана кнопка, то внутренний квадрат представляет эту кнопку.

Красные стрелки во внутреннем квадрате задают ширину и высоту выбранного объекта. Щелкнув на любой из стрелок, можно изменить ее стиль, сделав либо пунктирной, либо сплошной. Если горизонтальная стрелка проведена с помощью сплошной линии, то ширина объекта может свободно изменяться, когда окно изменяет свои размеры. Если же горизонтальная стрелка проведена с помощью пунктирной линии, то устройство iPhone по возможности попытается сохранить ширину объекта. То же самое относится к высоте объекта и вертикальной стрелке.

Четыре красные I-образные фигуры снаружи от внутреннего прямоугольника задают расстояния между краями выбранного объекта и аналогичными краями представления, которое его содержит. Если эта фигура нарисована пунктиром, то это расстояние может изменяться, а если сплошной красной линией, то по возможности это расстояние должно оставаться постоянным.

Непонятно?

Возможно, эта концепция станет более понятной, когда вы увидите ее в действии. Еще раз обратитесь к рис. 5.5, на котором показаны настройки автоматического изменения размеров, принятые по умолчанию. Эти настройки свидетельствуют о том, что при изменении их родительских представлений размеры объектов будут оставаться постоянными, и расстояния от левого и верхнего краев также должны оставаться постоянными.

Переместите курсор на анимацию, расположенную рядом с элементом управления, автоматически изменяющем размеры, чтобы увидеть его поведение при изменении размеров. Обратите внимание на то, что внутренние прямоугольники остаются на месте по отношению к левому и верхнему краям родительского представления, когда это представление изменяет размеры.

Проведите такой эксперимент. Щелкните на обеих красных I-образных фигурах (сверху и слева от внутреннего прямоугольника), чтобы они стали пунктирными, как показано на рис. 5.6.

Теперь щелкните на вертикальной стрелке внутри прямоугольника и на I-образных фигурах сверху и снизу от прямоугольника, чтобы атрибуты автоматического изменения размеров стали такими, как на рис. 5.7.

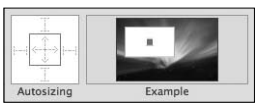

Рис. 5.6. Когда все линии проведены пунктиром, элемент управления свободно перемещается в пределах родительского представления, сохраняя свои размеры

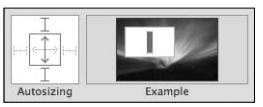

Рис. 5.7. Эта конфигурация допускает изменение высоты объекта

Здесь мы указали, что высота объекта может изменяться и что расстояние от верхнего края объекта до верхнего края окна, а также расстояние от нижнего края объекта до нижнего края окна должны оставаться постоянными. В этой конфигурации ширина объекта не может изменяться, а высота может. Измените атрибуты автоматического изменения размера еще несколько раз и посмотрите на анимацию, пока не поймете, как разные настройки влияют на поведение объекта при повороте и изменении размера объектов.

Настройки атрибутов автоматического изменения размеров кнопок

Теперь установим атрибуты автоматического изменения размеров наших шести кнопок. Сначала выясните, понимаете ли вы их смысл. Если нет, то взгляните на рис. 5.6, на котором показаны атрибуты автоматического изменения размера, которые необходимы, чтобы они оставались на экране при повороте телефона.

Если ваши атрибуты установлены так, как показано на рис. 5.8, сохраните nib-файл, перейдите к среде Xcode, а затем соберите и выполните приложение. На этот раз, когда будет запущен симулятор устройства iPhone, выберите команду Hardware⇨Rotate Left или Rotate Right, и тогда все кнопки останутся на экране (см. рис. 5.8). Если вы повернете устройство в прежнее положение, то кнопки должны занять свои прежние позиции. Этот прием прекрасно работает во многих приложениях.

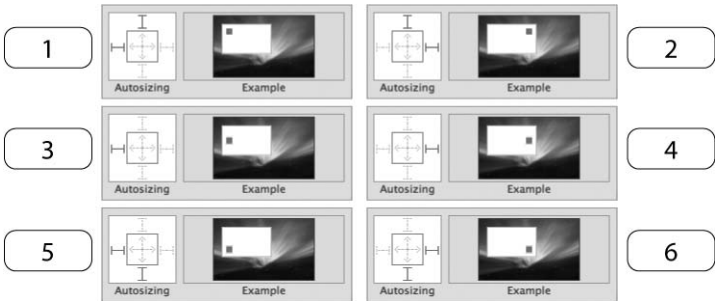

Рис. 5.8. Атрибуты автоматического изменения размеров для всех шести кнопок

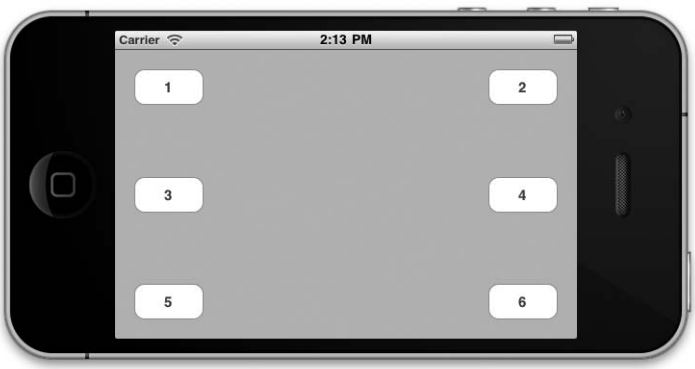

Рис. 5.9. Кнопки, занимающие новое положение после поворота

В этом примере мы сохранили одинаковый размер всех кнопок, поэтому на этот раз все кнопки видны на экране и готовы к работе, но, к сожалению, огромное пространство экрана осталось незанятым. Возможно, было бы лучше допустить изменение размеров кнопок, чтобы они заняли пустующее пространство? Поэкспериментируйте с атрибутами автоматического изменения размеров шести кнопок и добавьте новые кнопки, если хотите. Продолжайте это делать до тех пор, пока не почувствуете, что хорошо понимаете, как работает механизм автоматического изменения размеров.

В ходе экспериментов вы обязательно заметите, что некоторые комбинации атрибутов автоматического изменения размеров не приводят к желаемому результату. Иногда необходимо существенно переделать свой интерфейс, чтобы можно было применить описанный метод. В этих ситуациях обычно приходится немного программировать. Посмотрим, как это делается.

Перестройка представления при повороте

Находясь в программе Interface Builder, щелкните по одному разу на каждой кнопке и, используя инспектор размера, измените поля W (для ширины) и H (для высоты) на `125`. В результате ширина и высота кнопок будет равна 125 пикселям. Расставьте кнопки, используя голубые линии разметки, чтобы представление выглядело так, как на рис. 5.10.

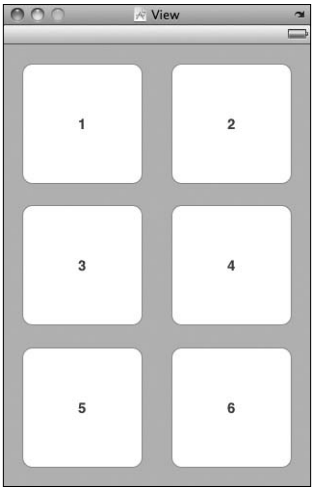

Рис. 5.10. Представление после изменения размеров всех кнопок

Сохраните nib-файл, перейдите к среде Xcode и выполните проект снова. Можете ли вы предсказать, что произойдет, когда вы повернете экран? Если вы установили атрибуты так, как показано на рис. 5.8, то, вероятно, будете недовольны. Кнопки перекрываются и выглядят так, как на рис. 5.11, поскольку высоты экрана в альбомном режиме недостаточно, чтобы поместить три кнопки высотой 125 пикселей.

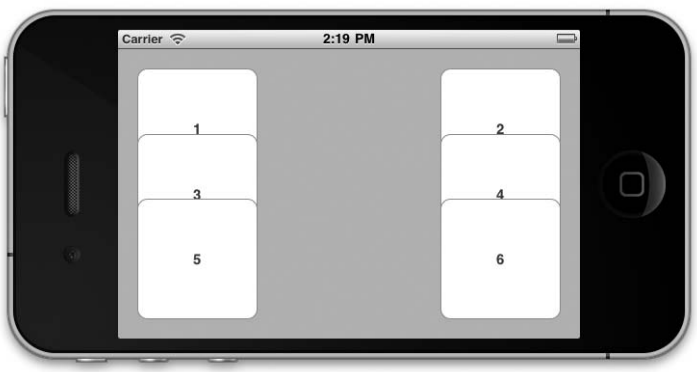

Рис. 5.11. Не совсем то, что вы хотели, — слишком большое перекрытие. Необходимо другое решение

Эту проблему можно решить, используя атрибуты автоматического изменения размеров, чтобы позволить изменение высоты, но при этом пространство экрана будет использоваться неэффективно, поскольку всередине экрана останется большой просвет. Если для шести кнопок достаточно места в портретном режиме, то его должно хватить для шести квадратных кнопок в альбомном режиме — достаточно просто их переставить. Для этого можно вычислить новое положение для каждой кнопки при повороте представления.

Объявление и связывание выходов

Для того чтобы изменить атрибуты элемента управления, нам нужен выход, ссылающийся на объект, который мы хотим изменить. Это значит, что мы должны объявить выход для каждой из шести кнопок, чтобы изменить их расстановку. Добавим в файл `AutosizeViewController.h` следующий код:

```
#import <UIKit/UIKit.h>

@interface AutosizeViewController : UIViewController {
    UIButton *button1;
    UIButton *button2;
    UIButton *button3;
    UIButton *button4;
    UIButton *button5;
    UIButton *button6;
}
@property (nonatomic, retain) IBOutlet UIButton *button1;
@property (nonatomic, retain) IBOutlet UIButton *button2;
@property (nonatomic, retain) IBOutlet UIButton *button3;
@property (nonatomic, retain) IBOutlet UIButton *button4;
@property (nonatomic, retain) IBOutlet UIButton *button5;
@property (nonatomic, retain) IBOutlet UIButton *button6;
@end
```

Сохраните этот файл и перейдите к программе Interface Builder. Проведите линию от пиктограммы File's Owner к каждой из шести кнопок и соедините их с соответствующим выходом. Соединив все шесть кнопок, сохраните nib-файл и вернитесь в среду Xcode.

СОВЕТ. Обратите внимание на описанную выше шаблонную процедуру. Мы добавили объявления выходов в заголовочный файл и сохранили его, так что программа Interface Builder знает об их существовании. Затем отредактировали nib-файл, чтобы связать выходы. Мы будем часто выполнять эту операцию. Одна из самых больших ошибок, которые делают новички, разрабатывающие приложения для системы iOS, заключается в том, что они забывают сохранить заголовочный файл, прежде чем переключиться обратно на программу Interface Builder, чтобы связать выходы. Если вы забудете сохранить файл до выбора nib-файла и возврата к программе Interface Builder, то, когда будете соединять пиктограмму File's Owner с объектом интерфейса, программа Interface Builder не включит не сохраненные выходы в представленный список. Итак, необходимо всегда сохранять заголовочный файл после добавления выходов. Это должно стать привычкой.

Перемещение кнопок при повороте

Для того чтобы переместить кнопки и лучше использовать пространство экрана, необходимо заменить метод `willAnimateRotationToInterfaceOrientation:duration:` в файле `in AutosizeViewController.m`. Этот метод автоматически вызывается при повороте, но до появления его окончательной анимации.

Добавьте в программу код, приведенный ниже, и мы обсудим, что он означает.

```
#import "AutosizeViewController.h"

@implementation AutosizeViewController
@synthesize button1;
@synthesize button2;
```

```
@synthesize button3;
@synthesize button4;
@synthesize button5;
@synthesize button6;

- (void)willAnimateRotationToInterfaceOrientation:(UIInterfaceOrientation)
    interfaceOrientation duration:(NSTimeInterval)duration {
    if (interfaceOrientation == UIInterfaceOrientationPortrait
            || interfaceOrientation ==
            UIInterfaceOrientationPortraitUpsideDown)
    {
        button1.frame = CGRectMake(20, 20, 125, 125);
        button2.frame = CGRectMake(175, 20, 125, 125);
        button3.frame = CGRectMake(20, 168, 125, 125);
        button4.frame = CGRectMake(175, 168, 125, 125);
        button5.frame = CGRectMake(20, 315, 125, 125);
        button6.frame = CGRectMake(175, 315, 125, 125);
    }
    else
    {
        button1.frame = CGRectMake(20, 20, 125, 125);
        button2.frame = CGRectMake(20, 155, 125, 125);
        button3.frame = CGRectMake(177, 20, 125, 125);
        button4.frame = CGRectMake(177, 155, 125, 125);
        button5.frame = CGRectMake(328, 20, 125, 125);
        button6.frame = CGRectMake(328, 155, 125, 125);
    }
}
- (BOOL)shouldAutorotateToInterfaceOrientation:
   (UIInterfaceOrientation)interfaceOrientation {
   return (interfaceOrientation !=
   UIInterfaceOrientationPortraitUpsideDown);
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Удаляем представление из памяти,
    // если оно не имеет родительского представления
    // Удаляем из памяти все несущественное,
    // например кэшированные данные
}
- (void)viewDidUnload {
   // Удаляем все удерживаемые дочерние представления главного представления.
   // например, myOutlet = nil;
   self.button1 = nil;
   self.button2 = nil;
   self.button3 = nil;
   self.button4 = nil;
   self.button5 = nil;
   self.button6 = nil;
   [super viewDidUnload];
}

- (void)dealloc {
```

```
    [button1 release];
    [button2 release];
    [button3 release];
    [button4 release];
    [button5 release];
    [button6 release];
    [super dealloc];
}
```

Размеры и положения всех представлений, включая элементы управления, такие как кнопки, определены в свойстве `frame`, представляющем собой структуру типа `CGRect`. Функция `CGRectMake()` предоставлена компанией Apple, чтобы упростить создание структуры `CGRect`, указывая координаты `x` и `y`, а также ширину и высоту.

ЗАМЕЧАНИЕ. Имя функции `CGRect()` начинается с букв `CG`. Это означает, что она принадлежит библиотеке Core Graphics. Как следует из ее имени, библиотека Core Graphics содержит код, связанный с графикой и рисованием. В более ранних версиях пакета iOS SDK библиотеки Core Graphics не было среди шаблонов проектов в среде Xcode, и ее следовало добавлять вручную. Теперь этот этап не обязателен, потому что библиотека Core Graphics автоматически включается при использовании шаблонов Xcode для создания приложения для устройства iPhone или iPad.

Сохраните этот код. Потом соберите и выполните приложение, чтобы увидеть его в действии. Попробуйте повернуть устройство и посмотреть, как изменится положение кнопок.

Переключение представлений

Перемещение элементов управления, описанное в предыдущем разделе, может быть слишком утомительным занятием, особенно для сложных интерфейсов. Было бы прекрасно, если бы мы могли спроектировать разные представления для портретного и альбомного режимов, а затем переключать их при повороте телефона.

Да, мы можем это сделать. Однако это довольно сложная процедура, которую следует применять только для очень сложных интерфейсов. В то время как элементы управления обоих представлений могут инициировать одни и те же действия, нам потребуются два совершенно разных набора выходов — по одному для каждого представления, — что значительно увеличивает сложность программы. Тем не менее эту сложность ни в коем случае нельзя считать непреодолимой, и существуют ситуации, в которых именно это решение является оптимальным. Испытаем его. Для того чтобы продемонстрировать его применение, создадим приложение с разными представлениями для портретной и альбомной ориентации. Несмотря на то что наш интерфейс недостаточно сложен, чтобы обосновать использование такого изощренного метода, именно его простота позволит прояснить этот процесс.

Создайте новый проект в среде Xcode, используя шаблон проекта View-based Application, как всегда (с другими шаблонами мы начнем работать в следующей главе). Назовите этот проект `Swap`. Приложение будет запускаться в портретном режиме с двумя кнопками, расположенными одна над другой (рис. 5.12).

Поворот телефона приводит к переключению на совершенно другое представление, специально разработанное для альбомной ориентации. Это представление также содержит две кнопки с теми же самыми метками (рис. 5.13), поэтому пользователю не обязательно знать, что перед ним совершенно другое представление.

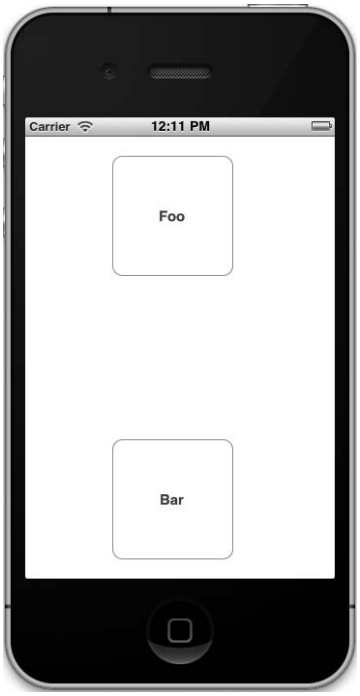

Рис. 5.12. Запуск приложения `Swap` в портретном режиме с двумя кнопками

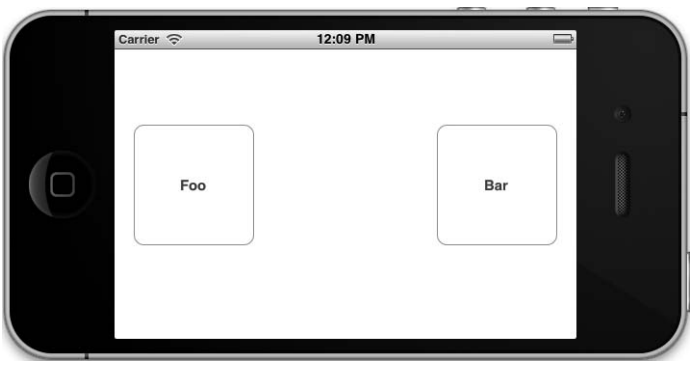

Рис. 5.13. Похоже, но не совпадает. Это альбомное представление с двумя другими кнопками

При постукивании по кнопке она скрывается. В реальном приложении может понадобиться показывать и скрывать кнопки. Например, можно создать кнопку, которая запускает продолжительный процесс, и скрыть ее, чтобы пользователь не смог постукивать по ней, пока процесс не будет завершен.

Определение выходов и действий

Поскольку каждое представление имеет две кнопки, а выход не может указывать на несколько объектов одновременно, необходимо объявить четыре выхода: два для альбомного

представления и два для портретного. При использовании этого метода важно правильно выбрать имена выходов, чтобы не запутаться.

Но постойте! А кто сказал, что нам действительно необходимы выходы для всех этих кнопок? Поскольку мы деактивируем кнопку, по которой слегка стукнул пользователь, может, использовать не выход, а аргумент `sender`? Как и в сценарии для приложения с одним представлением, именно так и следует сделать.

Подумайте об этом. Что произойдет, если пользователь слегка стукнет на кнопке Foo, а затем повернет телефон? Кнопка Foo в другом представлении — это совершенно другая кнопка, которая остается видимой, но работает не так, как мы ожидаем. Мы бы не хотели сообщать пользователям, что объект, с которым они работают в данный момент, — уже не тот объект, с которым они работали секунду назад.

Кроме выходов для кнопок, нам необходимы еще два выхода, указывающие на два разных варианта нашего представления. Когда мы работали с единственным представлением, нам было достаточно свойства родительского класса представления. Но так как мы собираемся изменять индикатор представления в ходе выполнения программы, необходимо гарантировать, что мы можем работать с обоими представлениями, а значит, нам нужны два выхода `UIView`.

Наши кнопки должны инициировать действие, поэтому нам нужен один метод, выполняющий действие, который будет вызываться при нажатии любой из кнопок, и мы просто объявим единственное действие `buttonPressed:` в классе контроллера нашего представления.

Объявление действий и выходов

Для того чтобы создать необходимые выходы, перейдите к программе Interface Builder и добавьте следующий код в файл `SwapViewController.h`:

```
#import <UIKit/UIKit.h>

#define degreesToRadians(x) (M_PI * (x) / 180.0)

@interface SwapViewController : UIViewController {
    UIView *landscape;
    UIView *portrait;

    // Foo
    UIButton *landscapeFooButton;
    UIButton *portraitFooButton;

    // Bar
    UIButton *landscapeBarButton;
    UIButton *portraitBarButton;
}
@property (nonatomic, retain) IBOutlet UIView *landscape;
@property (nonatomic, retain) IBOutlet UIView *portrait;
@property (nonatomic, retain) IBOutlet UIButton *landscapeFooButton;
@property (nonatomic, retain) IBOutlet UIButton *portraitFooButton;
@property (nonatomic, retain) IBOutlet UIButton *landscapeBarButton;
@property (nonatomic, retain) IBOutlet UIButton *portraitBarButton;
- (IBAction)buttonPressed:(id)sender;
@end
```

Следующая строка кода представляет собой обычный макрос, который выполняет преобразование градусов в радианы:

```
#define degreesToRadians(x) (M_PI * (x) / 180.0)
```

Мы будем использовать этот макрос при вызове функции, получающей в качестве аргумента радианы. Большинство людей не измеряют углы в радианах, поэтому этот макрос сделает программу более понятной, позволив пользователям задавать углы в градусах, а не в радианах. Все остальное в этом заголовочном файле должно быть знакомо вам.

Реализовав выходы, перейдите к программе Interface Builder и соберите два требуемых представления. Дважды щелкните на файле `SwapViewController.xib` в папке Resources, чтобы открыть его в программе Interface Builder.

Проектирование двух представлений

В идеале все, что вы видите сейчас в программе Interface Builder, должно быть хорошо знакомым вам. Нам нужны два представления в nib-файле. Мы не хотим использовать существующее представление, которое является частью шаблона, потому что его размеры нельзя изменить. Вместо этого удалим представление, заданное по умолчанию, и создадим два новых.

Находясь в главном окне, щелкните на пиктограмме View и нажмите клавишу <Delete>. Затем перетащите два представления из библиотеки в главное окно. После этого вы получите две пиктограммы с меткой View. Это может показаться непонятным, поэтому переименуйте их, чтобы они назывались по-разному.

Для того чтобы переименовать пиктограмму в главном окне nib-файла, необходимо щелкнуть на пиктограмме, чтобы выбрать ее, подождать секунду-другую, а затем щелкнуть на имени пиктограммы. Через несколько секунд имя можно будет редактировать, и вы сможете ввести новое имя. Обратите внимание на то, что этот трюк работает только в режиме пиктограмм. Назовите одно из представлений Portrait, а другое — Landscape. Теперь проведите соединительные линии от пиктограммы File's Owner к пиктограмме Portrait, и когда на экране появится серое всплывающее меню, выберите выход `portrait`. После этого перетащите соединительную линию от пиктограммы File's Owner к пиктограмме Landscape и выберите выход `landscape`. В третий раз проведите соединительную линию от пиктограммы File's Owner к представлению Portrait, а затем выберите выход `view`, для того чтобы указать, какое из представлений должно быть показано в момент запуска.

Дважды щелкните на пиктограмме Landscape и нажмите комбинацию клавиш <⌘+3>, чтобы вызвать инспектор размеров. Теперь это представление должно иметь 320 пикселей в ширину и 460 в высоту. Измените значения так, чтобы представление имело 480 пикселей в ширину и 300 в высоту, или нажмите маленькую пиктограмму в виде стрелки на правой стороне заголовка представления, которая автоматически изменяет пропорции представления для альбомного режима. Перетащите два объекта Round Rect Buttons из библиотеки на представление Landscape. Точный размер и положение этих кнопок не имеет значения. Подойдет 125 пикселей в ширину и 125 в высоту. Дважды щелкните на левой кнопке и присвойте ей название Foo, затем дважды щелкните на правой кнопке и присвойте ей название Bar.

Проведите соединительную линию от пиктограммы File's Owner к кнопке Foo и присвойте ей выход `landscapeFooButton`; затем сделайте то же самое, чтобы привязать кнопку Bar к выходу `landscapeBarButton`. Щелкните на кнопке Foo и вызовите инспектор связей, нажав комбинацию клавиш <⌘+2>. Проведите линию от кружочка, соответствующего со-

бытию Touch Up Inside, к пиктограмме File's Owner и выберите действие buttonPressed:. Повторите эту процедуру для кнопки Bar, чтобы обе кнопки могли инициировать метод buttonPressed:, выполняющий действие. Теперь можно закрыть окно Landscape.

Дважды щелкните на пиктограмме Portrait, чтобы открыть это представление для редактирования. Захватите в библиотеке еще два объекта Round Rect Buttons и на этот раз поместите их один над другим. Снова задайте их ширину и высоту равными 125 пикселям. Дважды щелкните на верхней кнопке и присвойте ей название Foo. Затем дважды щелкните на нижней кнопке и присвойте ей название Bar.

Проведите соединительную линию от пиктограммы File's Owner к кнопке Foo и свяжите ее с выходом portraitFooButton. Затем снова проведите соединительную линию от пиктограммы File's Owner к кнопке Bar и свяжите ее с выходом portraitBarButton.

Щелкните на кнопке Foo и проведите соединительную линию от события Touch Up Inside к инспектору связей через пиктограмму File's Owner и выберите действие buttonPressed:. Повторите эту процедуру для кнопки Bar.

Сохраните nib-файл и перейдите к среде Xcode.

Реализация переключения и действия

Мы практически закончили наше приложение. Осталось только написать код, выполняющий переключение, и запрограммировать реакцию на постукивание по кнопкам. Добавьте код, приведенный ниже, в файл SwapViewController.m.

ЗАМЕЧАНИЕ. Этот листинг не содержит закомментированные методы-заглушки. Можете смело удалять закомментированные методы, содержащиеся в классе контроллера.

```
#import "SwapViewController.h"

@implementation SwapViewController
@synthesize landscape;
@synthesize portrait;
@synthesize landscapeFooButton;
@synthesize portraitFooButton;
@synthesize landscapeBarButton;
@synthesize portraitBarButton;

- (void)willAnimateRotationToInterfaceOrientation:(UIInterfaceOrientation)
    interfaceOrientation duration:(NSTimeInterval)duration {
    if (interfaceOrientation == UIInterfaceOrientationPortrait) {
       self.view = self.portrait;
       self.view.transform = CGAffineTransformIdentity;
       self.view.transform =
           CGAffineTransformMakeRotation(degreesToRadians(0));
       self.view.bounds = CGRectMake(0.0, 0.0, 320.0, 460.0);
   }
   else if (interfaceOrientation == UIInterfaceOrientationLandscapeLeft) {
       self.view = self.landscape;
       self.view.transform = CGAffineTransformIdentity;
       self.view.transform =
       CGAffineTransformMakeRotation(degreesToRadians(-90));
```

```
        self.view.bounds = CGRectMake(0.0, 0.0, 480.0, 300.0);
    }
    else if (interfaceOrientation ==
            UIInterfaceOrientationPortraitUpsideDown) {
        self.view = self.portrait;
        self.view.transform = CGAffineTransformIdentity;
        self.view.transform =
          CGAffineTransformMakeRotation(degreesToRadians(180));
        self.view.bounds = CGRectMake(0.0, 0.0, 320.0, 460.0);
    }
    else if (interfaceOrientation ==
             UIInterfaceOrientationLandscapeRight) {
        self.view = self.landscape;
        self.view.transform = CGAffineTransformIdentity;
        self.view.transform =
             CGAffineTransformMakeRotation(degreesToRadians(90));
        self.view.bounds = CGRectMake(0.0, 0.0, 480.0, 300.0);
    }
}

- (IBAction)buttonPressed:(id)sender {
    if (sender == portraitFooButton || sender == landscapeFooButton) {
        portraitFooButton.hidden = YES;
        landscapeFooButton.hidden = YES;
    } else {
        portraitBarButton.hidden = YES;
        landscapeBarButton.hidden = YES;
    }
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
    interfaceOrientation {
    return YES;
}

- (void)didReceiveMemoryWarning {
    // Удаляем представление, если у него нет родительского представления
    [super didReceiveMemoryWarning];
    // Удаляем все несущественное, например, кэшированные данные
}

- (void)viewDidUnload {
    // Удаляем все удерживаемые дочерние представления главного представления.
    // например, self.myOutlet = nil;
    self.landscape = nil;
    self.portrait = nil;
    self.landscapeFooButton = nil;
    self.portraitFooButton = nil;
    self.landscapeBarButton = nil;
    self.portraitBarButton = nil;
    [super viewDidUnload];
}
```

```
- (void)dealloc {
    [landscape release];
    [portrait release];
    [landscapeFooButton release];
    [portraitFooButton release];
    [landscapeBarButton release];
    [portraitBarButton release];
    [super dealloc];
}
@end
```

Первый метод в нашем новом коде называется `willAnimateRotationToInterfaceOrientation:duration:`. Именно этот замещенный метод из родительского класса вызывается, когда начинается поворот, но до его завершения. Действия, которые выполняются в данном методе, анимируются как первая часть анимации всего поворота.

В этом методе определяется ориентация, в которой будет находиться телефон после поворота, и задается свойство представления — альбомное или портретное, — подходящее для новой ориентации. Затем мы вызываем метод `CGAffineTransformMakeRotation`, являющийся частью библиотеки Core Graphics, чтобы создать преобразование поворота (rotation transformation). Преобразование — это математическое описание изменения размеров, положения или угла поворота объекта. Обычно при повороте устройства система iOS задает значение преобразования автоматически. Однако при переключении в новое представление нам необходимо убедиться, что представление будет правильным, чтобы не запутывать операционную систему. Именно это делает метод `willAnimateRotationToInterfaceOrientation:duration:` каждый раз, когда задает свойство преобразования представлений. После поворота представления мы уточняем его рамку, чтобы оно точно вписывалось в окно при выбранной ориентации.

Перейдем к методу `buttonPressed:`. Он не должен содержать ничего удивительного. Мы определяем, по какой кнопке стукнул пользователь, скрываем ее, а затем скрываем соответствующую кнопку в другом представлении.

Все, что написано в этом классе, должно быть вам знакомо. Новый метод `shouldAutorotateToInterfaceOrientation:` возвращает значение `YES`, чтобы сообщить устройству iPhone, что он поддерживает поворот к любой ориентации, а код, добавленный в метод `dealloc`, просто очищает память.

Теперь скомпилируйте и выполните приложение.

ЗАМЕЧАНИЕ. Если вы случайно щелкнете на обеих кнопках, то единственный способ вернуть их на экран — выйти из симулятора и заново выполнить проект. Не используйте этот подход в своих собственных приложениях.

Поворачиваем

В данной главе мы попытались описать три разных подхода к реализации автоматического поворота в приложениях. Вы узнали, что такое атрибуты автоматического изменения размеров, как реструктурировать представления в коде при повороте устройства, работающего под управлением системы iOS. Кроме того, увидели, как переключаться между совершенно разными представлениями при повороте устройства, а также научились связывать новые библиотеки со своим проектом.

В этой главе вы получили первое впечатление о приложениях с несколькими представлениями на примере переключения между представлениями в одном и том же nib-файле. В следующей главе мы рассмотрим настоящие приложения с несколькими представлениями. Каждое представление, которое мы писали до сих пор, использовало единственный контроллер представления, и все они, за исключением последнего, имели единственное представление содержимого. Однако многие приложения для системы iOS, такие как Mail и Contacts, на самом деле имеют несколько представлений и контроллеров представлений, и мы рассмотрим, как они работают, в главе 6.