第 3 章

图 像 处 理

在移动平台开发中图像处理是非常重要的技术，可以开发很多有趣的应用和游戏。或许你已经使用它给你的女朋友拍一张照片，然后利用它处理图片添加效果。SketchMe（图 3-1 所示）是一款可以把照片添加素描或卡通效果图像处理应用。照片的来源可以是设备中图片库和通过照相机拍照，图 3-1 右下角所示效果是“3D 戈登雕刻”的处理效果，这些处理效果称为滤镜（Filter），学过 PhotoShop 的读者对滤镜应该比较熟悉。

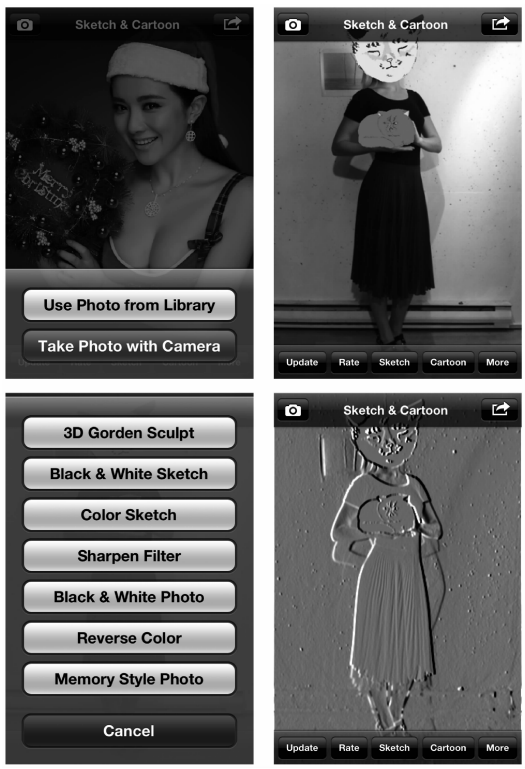

图 3-1　SketchMe 应用

本章将介绍：创建和使用 UIImage 对象、CIImage 对象和使用滤镜、人脸识别等知识。

3.1 使用图像

为了便于操作图像 iOS 中定义图像类，UIImage 是 UIKit 框架中定义的图像类，其封装了高层次图像类，可以通过多种方式创建这些对象。在 Core Graphics 框架（或 Quartz 2D）中也定义了 CGImage，它表示位图图像，因为 CGImage 被封装起来了，所以通常通过 CGImageRef 来使用 CGImage。

除了 UIImage 和 CGImage 外，在 Core Image 框架中也有一个图像类 CIImage，CIImage 封装的图像类能够很好地进行图像效果处理，例如，滤镜的使用。UIImage、CGImage 和 CIImage 之间可以互相转化，这个过程中需要注意内存释放问题，特别是 CGImage 与 UIImage 之间转化，涉及从 C 变量到 Objective-C 对象转化，如果这里使用了 ARC（Automatic Reference Counting，自动引用计数）技术，反而会使内存释放问题更复杂。随着后面的学习会逐步了解这些内存释放问题。

3.1.1 创建图像

本节介绍的创建图像主要是创建 UIImage 对象，CGImage 一般不直接创建，因此不打算特意介绍了。CIImage 对象创建在滤镜一节介绍，这一节中暂时不介绍。

下面介绍如何创建 UIImage 类图像对象。UIImage 有一些构造方法和静态创建方法（即直接通过类名调用静态方法创建）。

- + imageNamed:，静态创建方法，从应用程序包（资源文件）中加载图片创建对象，name 参数是指定文件名字，这种方法会建立图像缓存，第一次从文件中加载，以后从缓存中加载；
- + imageWithContentsOfFile:，静态创建方法，通过文件路径创建图像对象；
- + imageWithData:，静态创建方法，通过内存中 NSData 对象创建图像对象；
- + imageWithCGImage:，静态创建方法，通过 CGImageRef 创建图像对象；
- + imageWithCIImage:，静态创建方法，通过 CIImage 创建图像对象；
- - initWithContentsOfFile:，构造方法，通过文件路径创建图像对象；
- - initWithData:，构造方法，通过内存中 NSData 对象创建图像对象；
- - initWithCGImage:，构造方法，通过内存中 CGImageRef 对象创建图像对象；
- - initWithCIImage:，构造方法，通过内存中 CIImage 对象创建图像对象。

对于上面介绍的方法可以根据图像来源的不同进行分类。在 iOS 设备中图像来源主要有以下 4 种不同渠道：

- 从应用程序包中（资源文件）加载；
- 从应用程序沙箱目录加载；
- 从云服务器端获取；

■ 从设备图片库选取或从照相机抓取。

如果一个 icon.png 文件放在应用程序包中(资源文件)加载图像,可以通过下面的几种代码实现:

```
UIImage * image = [UIImage imageNamed:@"icon.png"];
```

或

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"icon" ofType:@"png"];
UIImage * image = [UIImage imageWithContentsOfFile:path];
```

或

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"icon" ofType:@"png"];
UIImage * image = [[UIImage alloc] initWithContentsOfFile:path];
```

或

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"icon" ofType:@"png"];
NSData * data = [[NSData alloc] initWithContentsOfFile:path];
UIImage * image = [UIImage imageWithData:data];
```

或

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"icon" ofType:@"png"];
NSData * data = [[NSData alloc] initWithContentsOfFile:path];
UIImage * image = [[UIImage alloc] initWithData:data];
```

如果 icon.png 文件放在应用程序沙箱目录中的 Document 目录下面,可以通过下面的几种代码实现:

```
NSArray * paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                                       NSUserDomainMask, YES);
NSString * path = [[paths lastObject]
                                        stringByAppendingPathComponent:@"icon.png"];
UIImage * image = [UIImage imageWithContentsOfFile:path];
```

或

```
NSArray * paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                                       NSUserDomainMask, YES);
NSString * path = [[paths lastObject]
                                        stringByAppendingPathComponent:@"icon.png"];
UIImage * image = [[UIImage alloc] initWithContentsOfFile:path];
```

或

```
NSArray * paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                                       NSUserDomainMask, YES);
NSString * path = [[paths lastObject]
                                        stringByAppendingPathComponent:@"icon.png"];
NSData * data = [[NSData alloc] initWithContentsOfFile:path];
UIImage * image = [UIImage imageWithData:data];
```

或

```
NSArray * paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                                       NSUserDomainMask, YES);
```

```
NSString * path = [[paths lastObject]
                                    stringByAppendingPathComponent:@"icon.png"];
NSData * data = [[NSData alloc] initWithContentsOfFile:path];
UIImage * image = [[UIImage alloc] initWithData:data];
```

在上述代码中获得应用程序沙箱目录中 Document 目录语句如下：

```
NSArray * paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                                NSUserDomainMask, YES);
NSString * path = [[paths lastObject]
                                    stringByAppendingPathComponent:@"icon.png"];
```

如果 icon.png 文件放在云服务器端 http://xxx/icon.png 下，可以通过如下的几种方式创建 UIImage 图像对象：

```
NSURL * url = [NSURL URLWithString:@"http://xxx/icon.png"];
NSData * data = [[NSData alloc] initWithContentsOfURL:url];
UIImage * image = [UIImage imageWithData:data];
```

或

```
NSURL * url = [NSURL URLWithString:@"http://xxx/icon.png"];
NSData * data = [[NSData alloc] initWithContentsOfURL:url];
UIImage * image = [[UIImage alloc] initWithData:data];
```

上述代码介绍了 3 种情况下创建图像对象，而从设备图片库或从照相机抓取后面章节介绍。

下面通过一个具体的实例介绍这 3 种创建图像方式。实例通过屏幕下方的 3 个按钮分别从 3 个不同的来源创建 UIImage 对象显示在屏幕上方，如图 3-2 所示。

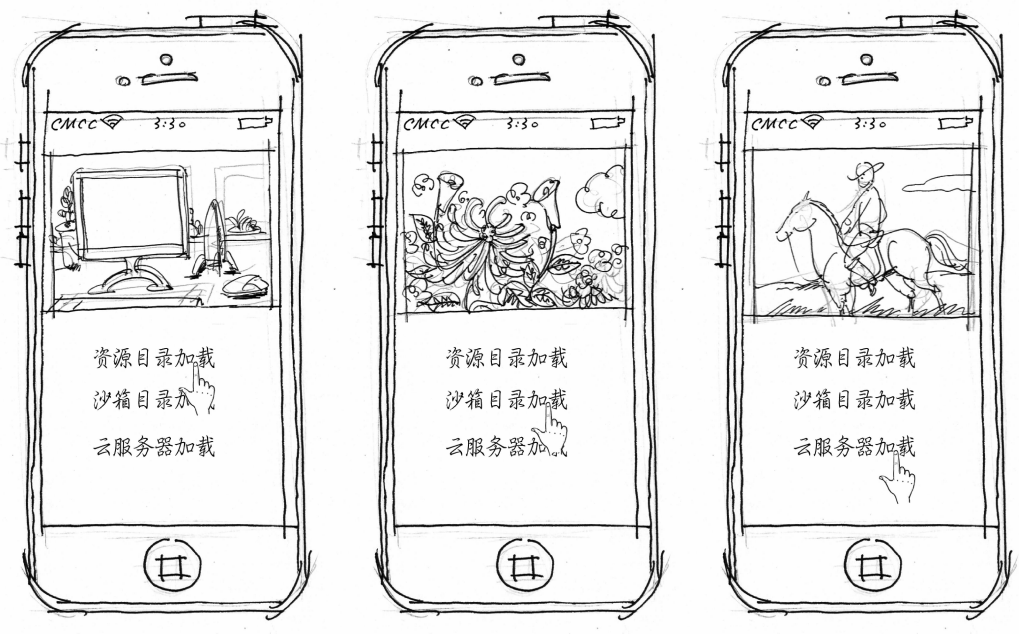

图 3-2 创建图像实例

首先使用 Xcode 选择 Single View Application 工程模板，创建一个 ImageSample 工程，并添加资源文件到工程中，具体的 UI 设计过程不再赘述，重点看看代码部分。其中视图控制器类 ViewController.h 代码如下所示。

```
#import <UIKit/UIKit.h>
#define FILE_NAME @"flower.png"

@interface ViewController: UIViewController

@property (retain, nonatomic) IBOutlet UIImageView * imageView;

- (IBAction)loadBundle:(id)sender;

- (IBAction)loadSandbox:(id)sender;

- (IBAction)loadWebService:(id)sender;

@end
```

视图控制器类 ViewController.m 文件代码如下所示。

```
#import "ViewController.h"

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];

    //复制图片到沙箱目录下
    [self createEditableCopyOfDatabaseIfNeeded];

    _imageView.image = [UIImage imageNamed:@"SkyDrive340.png"];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
}

- (void)createEditableCopyOfDatabaseIfNeeded {

    NSFileManager * fileManager = [NSFileManager defaultManager];                    ①
    NSString * writableDBPath = [self applicationDocumentsDirectoryFile];            ②

    BOOL dbexits = [fileManager fileExistsAtPath:writableDBPath];                    ③
    if (!dbexits) {
```

```
            NSString * defaultDBPath = [[[NSBundle mainBundle] resourcePath]
                    stringByAppendingPathComponent:FILE_NAME];                    ④

            NSError * error;
            BOOL success = [fileManager copyItemAtPath:defaultDBPath
                                    toPath:writableDBPath error:&error];          ⑤
            if (!success) {
                NSAssert1(0, @"错误写入文件: '%@'.", [error localizedDescription]);
            }
    }
}

- (NSString * )applicationDocumentsDirectoryFile {

  NSString * documentDirectory =
            NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                    NSUserDomainMask, YES) lastObject];
    NSString * path
        = [documentDirectory stringByAppendingPathComponent:FILE_NAME];
        return path;
}

- (IBAction)loadBundle:(id)sender {
    _imageView.image = [UIImage imageNamed:@"SkyDrive340.png"];
    NSString * path = [[NSBundle mainBundle]
                                pathForResource:@"SkyDrive340" ofType:@"png"];
    UIImage * image = [[UIImage alloc] initWithContentsOfFile:path];              ⑥
    _imageView.image = [UIImage imageNamed:@"SkyDrive340.png"];
    [image release];

}

- (IBAction)loadSandbox:(id)sender {

    NSString * path = [self applicationDocumentsDirectoryFile];
    UIImage * image = [UIImage imageWithContentsOfFile:path];                     ⑦
    _imageView.image = image;

}

- (IBAction)loadWebService:(id)sender {

    NSURL * url = [NSURL URLWithString:@"http://iosbook3.com/service/download.php?email=
<您的 http://iosbook3.com 网站注册邮箱>&FileName=2004-20H.jpg"];                  ⑧
```

```
    NSData * data = [[NSData alloc] initWithContentsOfURL:url];
    UIImage * image = [[UIImage alloc] initWithData:data];                    ⑨
    _imageView.image = image;

    [data release];
    [image release];
}


- (void)dealloc {
    [_imageView release];
    [super dealloc];
}

@end
```

上述代码中，viewDidLoad 是视图加载方法，其中[self createEditableCopyOfDatabase-IfNeeded]语句是把资源文件中的图片复制到沙箱中 Document 目录下面，这是因为沙箱目录一开始没有任何图片。还有该方法中_imageView.image = [UIImage imageNamed:@"SkyDrive340.png"]语句是初始化显示 SkyDrive340.png 图片，这样启动应用时候就可以看到有一张图片显示在界面中，而不是空白界面。

在 createEditableCopyOfDatabaseIfNeeded 方法中，第①行代码[NSFileManager defaultManager]是获得 NSFileManager 实例，是采用单例设计模式实现的。第②行代码调用 applicationDocumentsDirectoryFile 方法获得沙箱目录，这个方法是自己编写的。第③行代码是沙箱目录下文件是否存在，如果文件不存在则通过第⑤行代码调用 NSFileManager 的 copyItemAtPath:toPath:error:方法从应用程序包中复制一个图片文件到沙箱目录。第④行代码是获得应用程序包目录。

单击资源目录加载按钮触发-(IBAction)loadBundle:(id)sender 方法。在该方法中第⑥行代码是通过 UIImage 的 initWithContentsOfFile:方法从文件中创建图像对象。

单击沙箱目录加载按钮触发-(IBAction)loadSandbox:(id)sender 方法。在该方法中第⑦行代码是通过 imageWithContentsOfFile:方法从文件中创建图像对象。

单击云服务器加载按钮触发-(IBAction)loadWebService:(id)sender 方法。其中第⑧行代码是请求服务器地址，其中＜您的 http://iosbook3.com 网站注册邮箱＞需要读者换成自己的注册邮箱，如果没有，请读者到 iosbook3.com 网站注册成为会员，然后把注册的邮箱替换＜您的 http://iosbook3.com 网站注册邮箱＞，这样 iosbook3.com 才能提供服务，客户端才能显示图片。第⑨行代码是通过 initWithData:方法从服务器端取得数据来创建图像对象。

下面，可以运行以下实例，如图 3-3 所示。

图 3-3 实例运行

3.1.2 从设备图片库选取或从照相机抓取

图像的另外一个重要来源是从设备图片库选取或从照相机抓取。UIKit 中提供一个图像选择器 UIImagePickerController，UIImagePickerController 不仅可以实现选取图像还可以捕获视频信息。而且 UIImagePickerController 不仅可以从照相机中选取图像，还可以从相簿和相机胶卷中选择。相簿和相机胶卷是有区别的，相簿包含了相机胶卷，图 3-4 左所示是 iPod touch(或 iPhone)中的相簿，其中包含了相机胶卷、照片图库等内容，单击相机胶卷进入图 3-4 右所示是 iPod touch(或 iPhone)中的相机胶卷。相簿是可以查看所有图片，而相机胶卷是通过照相机拍摄或截屏获得的图片。

UIImagePickerController 的主要属性是 sourceType，sourceType 属性是在枚举 UIImagePickerControllerSourceType 中定义的 3 个常量：

- UIImagePickerControllerSourceTypePhotoLibrary，设置图片来源于“相簿”；
- UIImagePickerControllerSourceTypeCamera，设置图片来源于“照相机”；
- UIImagePickerControllerSourceTypeSavedPhotosAlbum，设置图片来源于“相机胶卷”。

UIImagePickerController 委托对象需要实现 UIImagePickerControllerDelegate 委托协议。UIImagePickerControllerDelegate 中定义了以下两个方法：

- － imagePickerController:didFinishPickingMediaWithInfo:，当选择完成时调用；
- － imagePickerControllerDidCancel:，当选择取消时调用。

下面通过一个实例具体介绍图像选择器过程。如图 3-5 和图 3-6 所示是图像选择器实例，其中图 3-5 左图是实例启动的第一个界面，单击“从图片库中选取”按钮会从设备的图片

图 3-4　iPod touch(或 iPhone)中的照片应用

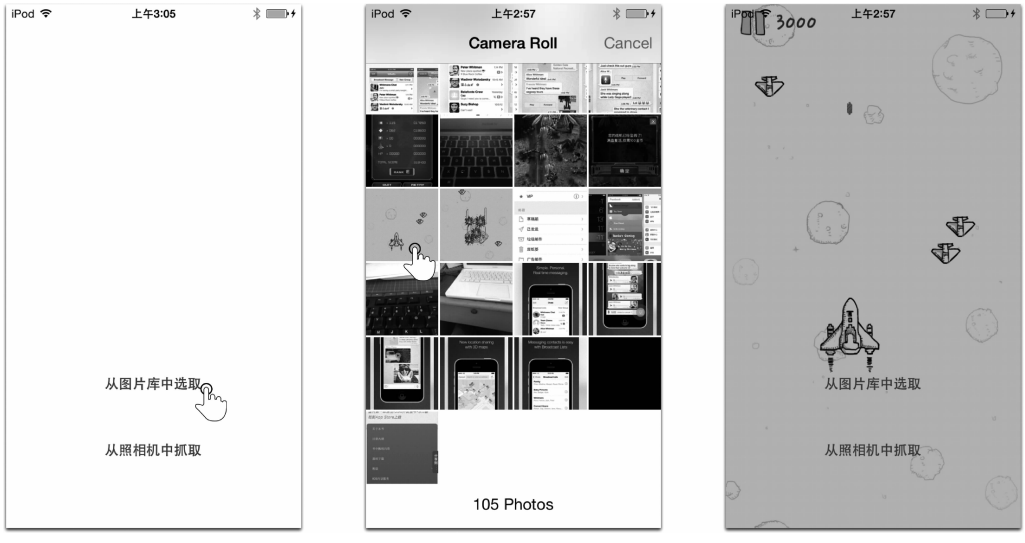

图 3-5　图像选择器实例(从图片库中选取)

库中选择图片(图 3-5 中),选择图片后回到开始界面,这时候选择的照片显示在屏幕上(图 3-5 右)。如果单击“从照相机中抓取”(图 3-6 左)按钮会从启动照相机预览(图 3-6 中),选择图片后回到开始界面,这时候选择的照片显示在屏幕上(图 3-6 右)。

图 3-6 图像选择器实例(从照相机中抓取)

首先使用 Xcode 选择 Single View Application 工程模板,创建一个 ImagePicker 工程。具体的 UI 设计过程不再赘述,重点看看代码部分。其中视图控制器类 ViewController.h 代码如下所示。

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

@interface ViewController: UIViewController
    <UIImagePickerControllerDelegate,UINavigationControllerDelegate>                ①

@property (strong, nonatomic) UIImagePickerController * imagePicker;                ②

@property (retain, nonatomic) IBOutlet UIImageView * imageView;

- (IBAction)pickPhotoLibrary:(id)sender;

- (IBAction)pickPhotoCamera:(id)sender;
@end
```

在 h 文件的第①行代码声明实现 UIImagePickerControllerDelegate 和 UINavigationControllerDelegate 委托协议,其中 UINavigationControllerDelegate 也是 UIImagePickerController 的 delegate 属性要求实现的协议,UINavigationControllerDelegate 协议定义了两个方法:

```
- navigationController:willShowViewController:animated:
- navigationController:didShowViewController:animated:
```

这两个方法是在抓取界面出现前后回调的方法。第②行代码 UIImagePickerController * imagePicker 定义了图像控制器的一个属性。

视图控制器类 ViewController.m 主要代码如下所示。

```
- (IBAction) pickPhotoCamera:(id)sender {                                    ①
    if ([UIImagePickerController isSourceTypeAvailable:
                UIImagePickerControllerSourceTypeCamera]) {
        _imagePicker = [[UIImagePickerController alloc] init];
        _imagePicker.delegate = self;
        _imagePicker.sourceType = UIImagePickerControllerSourceTypeCamera;

        [self presentViewController:_imagePicker animated:YES completion:nil];
    } else {
        NSLog(@"照相机不可用.");
    }
}

- (IBAction)pickPhotoLibrary:(id)sender {
    if (_imagePicker == nil) {
        _imagePicker = [[UIImagePickerController alloc] init];
    }
    _imagePicker.delegate = self;
    _imagePicker.sourceType
                    = UIImagePickerControllerSourceTypeSavedPhotosAlbum;

    [self presentViewController:_imagePicker animated:YES completion:nil];
}

- (void) imagePickerControllerDidCancel: (UIImagePickerController * )
                picker {                                                     ②
        _imagePicker.delegate = nil;
     [self dismissViewControllerAnimated:YES completion:nil];
}

- (void) imagePickerController: (UIImagePickerController * ) picker
    didFinishPickingMediaWithInfo: (NSDictionary * ) info {                  ③

    UIImage * originalImage = (UIImage * ) [info objectForKey:
                                    UIImagePickerControllerOriginalImage];

    self.imageView.image = originalImage;
    self.imageView.contentMode = UIViewContentModeScaleAspectFill;
    _imagePicker.delegate = nil;
    [self dismissViewControllerAnimated:YES completion:nil];

}
```

在上述代码中第①行 pickPhotoCamera:方法是从照相机抓取图片的方法,第②行和第③行代码是实现 UIImagePickerControllerDelegate 协议方法。

下面详细说明 pickPhotoCamera:和 imagePickerController:didFinishPickingMediaWithInfo:方法。takeImage:代码如下。

```
- (IBAction) pickPhotoCamera:(id)sender {
    if ([UIImagePickerController isSourceTypeAvailable:
                UIImagePickerControllerSourceTypeCamera]) {                    ①
        _imagePicker = [[UIImagePickerController alloc] init];                 ②
        _imagePicker.delegate = self;                                          ③
        _imagePicker.sourceType
                                    = UIImagePickerControllerSourceTypeCamera;  ④

        [self presentViewController:_imagePicker
                                                animated:YES completion:nil];  ⑤
    } else {
        NSLog(@"照相机不可用.");
    }
}
```

其中,第①行代码是通过 UIImagePickerController 的类方法 isSourceTypeAvailable:法判断是否设备支持照相机图像源(UIImagePickerControllerSourceTypeCamera),如果是在iOS 设备上运行该方法则返回 true,如果是在模拟器上运行则返回值是 false。

第②行代码是实例化 UIImagePickerController 对象。

第③行代码_imagePicker.delegate = self 是指定 UIImagePickerController 的委托对象为当前视图控制器,注意委托对象要求实现 UIImagePickerControllerDelegate 和 UINavigationControllerDelegate 委托协议。

第④行代码_imagePicker.sourceType = UIImagePickerControllerSourceTypeCamera 是指定 UIImagePickerController 对象的图像源为照相机。

第⑤行代码[self presentViewController:_imagePicker animated:YES completion:nil]是呈现系统提供的图像选择器界面。

imagePickerController:didFinishPickingMediaWithInfo:方法代码如下:

```
- (void) imagePickerController: (UIImagePickerController * ) picker
    didFinishPickingMediaWithInfo: (NSDictionary * ) info {

    UIImage * originalImage = (UIImage * ) [info objectForKey:
                                    UIImagePickerControllerOriginalImage];     ①

    self.imageView.image = originalImage;                                      ②
    self.imageView.contentMode = UIViewContentModeScaleAspectFill;             ③
```

```
    _imagePicker.delegate = nil;
    [self dismissViewControllerAnimated:YES completion:nil];                    ④

}
```

第①行代码是从参数 info 中取出原始图片数据，参数 info 如果抓取的是图片，则包含了原始或编辑后的图片数据，如果抓取的是视频，则包含的是视频存放路径。UIImagePickerControllerOriginalImage 键是获取原始图片数据，此外常用的键还有

- UIImagePickerControllerMediaType，由用户指定的媒体类型；
- UIImagePickerControllerEditedImage，编辑后的图片数据；
- UIImagePickerControllerCropRect，裁减后的图片数据；
- UIImagePickerControllerMediaURL，视频存放路径。

第②行代码 self.imageView.image = originalImage 是将从参数 info 中取出的图像对象保存到 Image View 控件上显示出来。

第③行代码 self.imageView.contentMode = UIViewContentModeScaleAspectFill 是设置 Image View 控件显示图像的方式，UIViewContentModeScaleAspectFill 是缩放图像填充视图，有可能被裁减掉一些内容。

第④行代码[self dismissViewControllerAnimated:YES completion:nil]是关闭图像选择器界面。

3.2　Core Image 框架

Core Image 是图像处理中非常重要的框架，如图 3-7 所示。Core Image 用来实时地处理和分析图像，它能处理来自于 Core Graphics，Core Video，and Image I/O 等框架的数据类型。并使用 CPU 或 GPU 进行渲染，Core Image 能够屏蔽很多低层次的技术细节，如 OpenGL ES 和 GCD(Grand Central Dispatch)等技术。

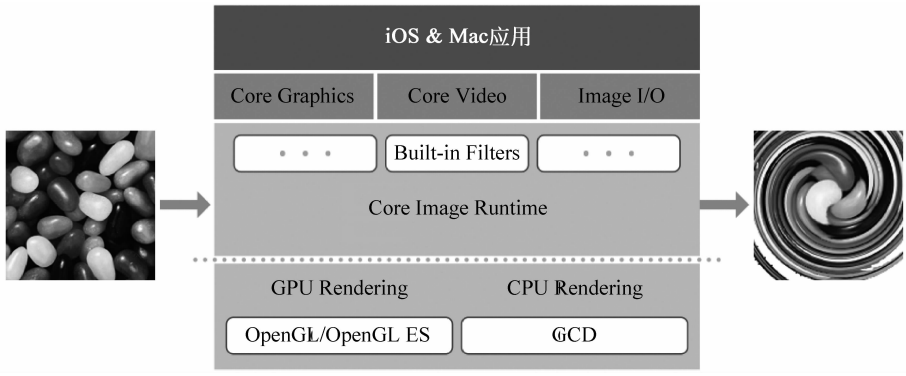

图 3-7　Core Image 框架

Core Image 框架中有以下几个非常重要的类。

- CIImage,Core Image 框架中的图像类;
- CIContext,上下文对象,所有图像处理都在一个 CIContext 中完成,通过 Quartz 2D 和 OpenGL 渲染 CIImage 对象;
- CIFilter,滤镜类包含一个字典结构,对各种滤镜定义了属于各自的属性;
- CIDetector,面部识别类,借助于 CIFaceFeature 可以识别嘴和眼睛的位置。

在 Core Image 框架中最常用的是 CIImage 类,有一些构造方法和静态创建方法(即直接通过类名调用静态方法创建)。这些方法如下。

- + imageWithCGImage:,静态创建方法,通过 CGImageRef 创建图像对象;
- + imageWithContentsOfURL:,静态创建方法,通过文件路径创建图像对象;
- + imageWithData:,静态创建方法,通过内存中 NSData 对象创建图像对象;
- - initWithCGImage:,构造方法,通过内存中 CGImageRef 对象创建图像对象;
- - initWithContentsOfURL:,构造方法,通过文件路径创建图像对象;
- - initWithData:,构造方法,通过内存中 NSData 对象创建图像对象。

在 iOS 设备中 CIImage 图像来源有主要有 4 种不同渠道。

- 从应用程序包中(资源文件)加载;
- 从应用程序沙箱目录加载;
- 从云服务器端获取;
- 从设备图片库选取或从照相机抓取。

如果一个 icon.png 文件放在应用程序包中(资源文件)加载图像,可以通过下面的几种代码实现。

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"icon" ofType:@"png"];
NSURL * fileNameAndPath = [NSURL fileURLWithPath:path];
CIImage * image = [CIImage imageWithContentsOfURL: fileNameAndPath];
```

或

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"icon" ofType:@"png"];
NSURL * fileNameAndPath = [NSURL fileURLWithPath:path];
CIImage * image = [[CIImage alloc] initWithContentsOfURL: fileNameAndPath];
```

或

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"icon" ofType:@"png"];
NSData * data = [[NSData alloc] initWithContentsOfFile:path];
CIImage * image = [CIImage imageWithData:data];
```

或

```
NSString * path = [[NSBundle mainBundle] pathForResource:@"icon" ofType:@"png"];
NSData * data = [[NSData alloc] initWithContentsOfFile:path];
CIImage * image = [[CIImage alloc] initWithData:data];
```

如果 icon.png 文件放在应用程序沙箱目录中的 Document 目录下面,可以通过以下几种代码实现。

```
NSArray * paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                    NSUserDomainMask, YES);
NSString * path = [[paths lastObject]
                                stringByAppendingPathComponent:@"icon.png"];
NSURL * fileNameAndPath = [NSURL fileURLWithPath:path];
CIImage * image = [CIImage imageWithContentsOfURL: fileNameAndPath];
```

或

```
NSArray * paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                 NSUserDomainMask, YES);
NSString * path = [[paths lastObject]
                                stringByAppendingPathComponent:@"icon.png"];
NSURL * fileNameAndPath = [NSURL fileURLWithPath:path];
CIImage * image = [[CIImage alloc] initWithContentsOfURL: fileNameAndPath];
```

或

```
NSArray * paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                    NSUserDomainMask, YES);
NSString * path = [[paths lastObject]
                                stringByAppendingPathComponent:@"icon.png"];
NSData * data = [[NSData alloc] initWithContentsOfFile:path];
CIImage * image = [CIImage imageWithData:data];
```

或

```
NSArray * paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                 NSUserDomainMask, YES);
NSString * path = [[paths lastObject]
                                stringByAppendingPathComponent:@"icon.png"];
NSData * data = [[NSData alloc] initWithContentsOfFile:path];
CIImage * image = [[CIImage alloc] initWithData:data];
```

在上述代码中获得应用程序沙箱目录中 Document 目录语句如下：

```
NSArray * paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                       NSUserDomainMask, YES);
NSString * path = [[paths lastObject]
                                       stringByAppendingPathComponent:@"icon.png"];
```

如果 icon.png 文件放在云服务器端 http://xxx/icon.png 下，可以通过如下的几种方式创建 CIImage 图像对象。

```
NSURL * url = [NSURL URLWithString:@"http://xxx/icon.png"];
NSData * data = [[NSData alloc] initWithContentsOfURL:url];
CIImage * image = [CIImage imageWithData:data];
```

或

```
NSURL * url = [NSURL URLWithString:@"http://xxx/icon.png"];
NSData * data = [[NSData alloc] initWithContentsOfURL:url];
CIImage * image = [[CIImage alloc] initWithData:data];
```

3.3 滤镜

使用过 Photoshop 的人对于滤镜(Filter)应该有很深刻的印象,那么滤镜到底是什么?维基百科关于滤镜的解释如下:

滤镜通常用于相机镜头作为调色、添加效果之用。如 UV 镜、偏振镜、星光镜、各种色彩滤光片。滤镜也是绘图软件中用于制造特殊效果的工具统称,以 Photoshop 为例,它拥有风格化、画笔描边、模糊、扭曲、锐化、视频、素描、纹理、像素化、渲染、艺术效果、其他等 12 个滤镜。

在 iOS 中滤镜的 API 是指 Core Image 框架定义好的,并且非常重要。

3.3.1 使用滤镜

iOS 有 90 多种滤镜,而 Mac OS X 10.8 提供了 120 多种滤镜。滤镜数量很多,而且又有很多参数和属性使用起来有点麻烦。下面介绍一下滤镜的使用流程,滤镜使用流程可以分成以下 3 个步骤。

(1) 创建滤镜 CIFilter 对象;

(2) 设置滤镜参数;

(3) 输出结果。

实例代码如下:

```
CIContext * context = [CIContext contextWithOptions:nil];
CIImage * cImage = [CIImage imageWithCGImage:[imageView.image CGImage]];
CIFilter * invert = [CIFilter filterWithName:@"CIColorInvert"];                    ①
[invert setDefaults];                                                              ②
[invert setValue: cImage forKey:@"inputImage"];                                    ③
CIImage * result = [invert valueForKey:@"outputImage"];                            ④
```

其中,第①行代码是创建滤镜对象,使用 filterWithName:方法创建,还可以使用 filterWithName:keysAndValues:方法创建,filterWithName:keysAndValues:在创建滤镜对象的同时可以设置其参数,使用实例代码如下:

```
CIFilter * invert = [CIFilter filterWithName:@"CIColorInvert"
                                keysAndValues:@"inputImage", cImage,
                                              nil];
```

第②行代码[invert setDefaults]是设置滤镜的默认参数,由于每个滤镜都有很多参数,这些参数不需要一一设置,可以通过[invert setDefaults]语句设置默认值。

第③行代码[invert setValue: cImage forKey:@"inputImage"]是设置输入参数(inputImage),是必须要设定的参数。

第④行代码是获得输出的 CIImage 图像对象,可以调用滤镜的 outputImage 方法获得

输出图像对象，代码如下所示。

```
CIImage * result = [invert outputImage];
```

3.3.2　实例：旧色调和高斯模糊滤镜

下面通过一个具体的实例介绍滤镜使用，通过屏幕下方的两个按钮分别从两种不同的滤镜（旧色调和高斯模糊）。图 3-8 左所示，选择“旧色调”段后拖曳下面的滑块，可以改变色调强度。图 3-8 右所示，选择“高斯模糊”段后拖曳下面的滑块，改变高斯模糊半径。

图 3-8　滤镜实例

首先使用 Xcode 选择 Single View Application 工程模板，创建一个 FilterEffects 工程。具体的 UI 设计过程不再赘述，重点看看代码部分。其中视图控制器类 ViewController.h 代码如下所示。

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

@interface ViewController: UIViewController
{
    int flag; // 0 为 CISepiaTone 1 为 CIGaussianBlur
}
```

```
@property (retain, nonatomic) IBOutlet UIImageView * imageView;

@property (retain, nonatomic) IBOutlet UISlider * slider;

@property (retain, nonatomic) UIImage * image;

@property (retain, nonatomic) IBOutlet UILabel * label;

- (IBAction)changeValue:(id)sender;

- (IBAction)segmentSelected:(id)sender;
@end
```

在上面的代码中 int flag 是定义的成员变量，用来记录单击了旧色调按钮还是单击了高斯模糊按钮，当单击旧色调按钮设置为 0，当单击高斯模糊按钮设置为 1。

视图控制器 ViewController.m 文件中操作旧色调 filterSepiaTone 方法代码如下所示。

```
- (void)filterSepiaTone {
    CIContext * context = [CIContext contextWithOptions:nil];                    ①
    CIImage * cImage = [CIImage imageWithCGImage:[_image CGImage]];              ②
    CIImage * result;

    CIFilter * sepiaTone = [CIFilter filterWithName:@"CISepiaTone"];             ③
    [sepiaTone setValue: cImage forKey: @"inputImage"];                          ④
    double value = [_slider value];                                              ⑤

    NSString * text = [[NSString alloc]
                    initWithFormat:@"旧色调 Intensity : %.2f",value];
    _label.text = text;
    [text release];

    [sepiaTone setValue: [NSNumber numberWithFloat: value]
                    forKey: @"inputIntensity"];                                  ⑥

    result = [sepiaTone valueForKey:@"outputImage"];                             ⑦

    CGImageRef imageRef = [context createCGImage:result fromRect:
                        CGRectMake(0, 0, self.imageView.image.size.width,
                                self.imageView.image.size.height)];              ⑧
    UIImage * image = [[UIImage alloc] initWithCGImage:imageRef];                ⑨

    _imageView.image = image;

    CFRelease(imageRef);                                                         ⑩
    [image release];
    flag = 0;
}
```

在上述代码中，第①行 CIContext * context = [CIContext contextWithOptions:nil]是创建 CIContext 对象，CIContext 构造方法是一个 NSDictionary 类型参数，它规定了各种选项，包括颜色格式以及内容是否应该运行在 CPU 或是 GPU 上。本例是默认值，所以只需要传入 nil。

第②行代码通过 CGImage 创建 CIImage 对象。第③行代码是创建 CISepiaTone(旧色调)滤镜。第④行代码是设置滤镜的输入参数(inputImage)。第⑤行代码是获取滑块的值，默认情况下滑块的取值是 0.0～1.0 之间，而旧色调滤镜色调强度取值范围也是 0.0～1.0 之间，因此可以把滑块的值直接作为旧色调滤镜色调强度值使用。第⑥行代码是设置旧色调滤镜色调强度(inputIntensity)值。

第⑦行代码 result = [sepiaTone valueForKey:@"outputImage"]是取得滤镜之后的图像对象，类型为 CIImage。

第⑧行代码是使用 CIContext 的 createCGImage:fromRect:方法创建 CGImageRef 对象。fromRect:部分参数是设置图像大小。第⑨行代码 UIImage * image = [[UIImage alloc] initWithCGImage:imageRef]是通过 CGImageRef 创建 UIImage 图像对象，因为 UIImage 图像对象是可以放置在 UIImageView 控件上显示的。第⑩行代码 CFRelease(imageRef)是释放 CGImageRef 对象。

视图控制器 ViewController.m 文件中操作高斯模糊 filterGaussianBlur 方法代码如下所示。

```
- (void)filterGaussianBlur {

    CIContext * context = [CIContext contextWithOptions:nil];
    CIImage * cImage = [CIImage imageWithCGImage:[_image CGImage]];
    CIImage * result;

    CIFilter * gaussianBlur = [CIFilter filterWithName:@"CIGaussianBlur"];                ①
    [gaussianBlur setValue: cImage forKey: @"inputImage"];
    double value = [_slider value];
    value * = 10;                                                                         ②

    NSString * text = [[NSString alloc]
                          initWithFormat:@"高斯模糊 Radius : %.2f",value];
    _label.text = text;
    [text release];

    [gaussianBlur setValue: [NSNumber numberWithFloat: value]
                    forKey: @"inputRadius"];                                              ③

    result = [gaussianBlur valueForKey:@"outputImage"];

    CGImageRef imageRef = [context createCGImage:result fromRect:
                    CGRectMake(0, 0, self.imageView.image.size.width,
```

```
                                                    self.imageView.image.size.height)];
        UIImage * image = [[UIImage alloc] initWithCGImage:imageRef];

        _imageView.image = image;

        CFRelease(imageRef);
        [image release];

        flag = 1;
    }
```

filterGaussianBlur:方法与 filterSepiaTone:方法非常相似,其中,第①行代码是创建高斯模糊滤镜(CIGaussianBlur)对象。第②行代码 value * =10 是将滑块取得的值乘 10,使得其他的取值范围为 0.0～10.0,这个取值将作为高斯模糊半径参数(Radius)。第③行代码是设置高斯模糊半径参数(Radius)取值,其中 inputRadius 是输入参数名。

3.4 人脸识别

人脸识别特指利用分析比较人脸视觉特征信息进行身份鉴别的计算机技术,主要用于身份识别。采用快速人脸检测技术可以从监控视频图像中实时查找人脸,并与人脸数据库进行实时比对,从而实现快速身份识别。

3.4.1 人脸识别开发

人脸的识别过程一般分以下 3 个步骤:

(1) 首先建立人脸的面纹数据库。可以通过照相机或摄像机采集人脸的面像图片,将这些面像图片生成面纹编码保存到数据库中。

(2) 获取当前人脸面像图片。即通过照相机或摄像机采集人脸的面像图片,将当前的面像文件生成面纹编码。

(3) 用当前的面纹编码与数据库中的面纹编码进行比对。

由于面纹编码不受抗光线、皮肤色调、面部毛发、发型、眼镜、表情和姿态变化的影响,具有可靠性强,从而使其可以精确地辨认出某个人。

在 iOS 5 之后提供人脸识别的 API,通过提供的 CIDetector 类可以进行人脸特征识别,CIDetector 是 Core Image 框架中的一个特征识别滤镜,CIDetector 主要用于人脸特征识别,通过它还可以获得眼睛和嘴的特征信息。但是 CIDetector 并不包括面纹编码提取,面纹编码提取还需要更为复杂的算法处理。也就是说使用 CIDetector 类可以找到一张图片中的人脸,但是这张脸是谁,CIDetector 无法判断,这需要有一个面纹数据库,把当前人脸提取面纹编码然后与数据库进行比对,这些内容超出了本书的范围,我们不再介绍了。

在此之前 iOS 开发这方面的应用可以采用 OpenCV 和 Face.com。其中,OpenCV (http://opencv.org/)是开源的 C 编写的图像处理和识别库,它提供了图像处理和计算机

视觉方面的很多通用算法。Face.com 提供了在线方式人脸识别服务，开发人员需要在 Face.com 网站注册 key，才可以使用 REST Web Service 风格的 API，提交人脸图片，Face.com 返回识别结果。

3.4.2　实例：是猩猩还是小女孩

下面通过一个具体的实例介绍使用 CIDetector 识别人脸过程，如图 3-9 所示。屏幕的上半部分是要识别的图片，为了进行比较图片中一个小女孩和一只猩猩，正常情况下应该只能识别出小女孩的脸。单击识别按钮可以将识别出来的脸图片放置在屏幕的下半部分显示出来。

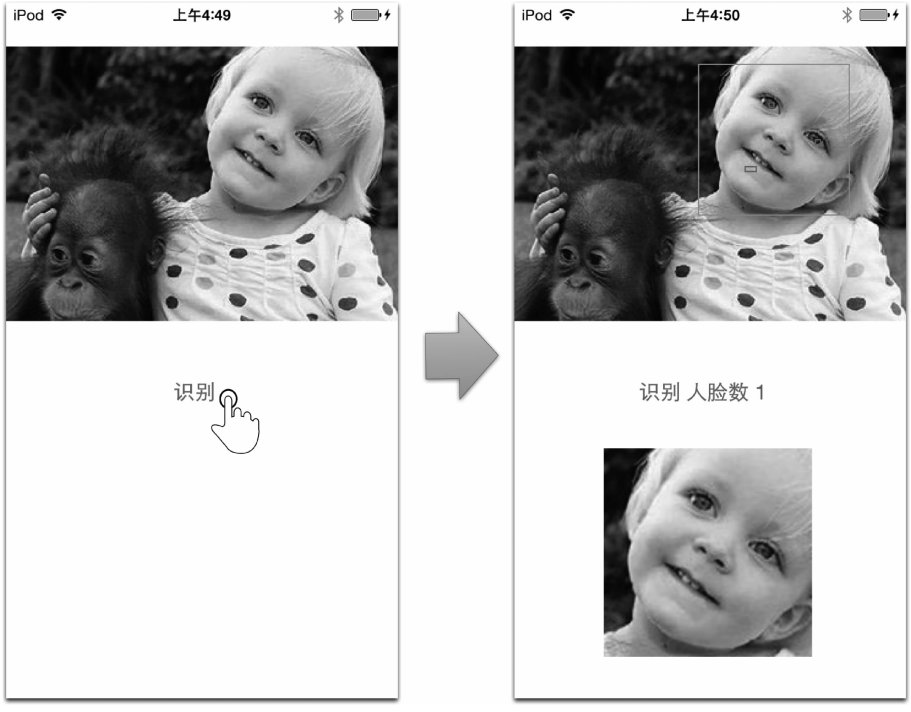

图 3-9　人脸识别实例

首先使用 Xcode 选择 Single View Application 工程模板，创建一个 FaceDetect 工程。具体的 UI 设计过程不再赘述，重点看看代码部分。其中视图控制器类 ViewController.h 代码如下所示。

```
@interface ViewController: UIViewController

@property (retain, nonatomic) IBOutlet UIImageView * inputImageView;
@property (retain, nonatomic) IBOutlet UIImageView * outputImageView;
@property (retain, nonatomic) IBOutlet UIButton * button;
- (IBAction)detect:(id)sender;
```

@end

视图控制器 ViewController.m 中，单击识别按钮调用方法 detect:，代码如下所示。

```
- (IBAction)detect:(id)sender {

    CIContext * context = [CIContext contextWithOptions:nil];

    UIImage * imageInput = [_inputImageView image];
    CIImage * image = [CIImage imageWithCGImage:imageInput.CGImage];

    //设置识别参数
    NSDictionary * param = [NSDictionary
                                    dictionaryWithObject:CIDetectorAccuracyHigh
                                        forKey:CIDetectorAccuracy];                    ①
    //声明一个 CIDetector,并设定识别类型
    CIDetector * faceDetector = [CIDetector detectorOfType:CIDetectorTypeFace
                                        context:context options:param];                ②

    //取得识别结果
    NSArray * detectResult = [faceDetector featuresInImage:image];                     ③

    UIView * resultView = [[UIView alloc]
                               initWithFrame:_inputImageView.frame];                   ④

    [self.view addSubview:resultView];


    for(CIFaceFeature * faceFeature in detectResult) {                                 ⑤

        //脸部
        UIView * faceView = [[UIView alloc] initWithFrame:faceFeature.bounds];
        faceView.layer.borderWidth = 1;
        faceView.layer.borderColor = [UIColor orangeColor].CGColor;
        [resultView addSubview:faceView];
        [faceView release];

        //左眼
        if (faceFeature.hasLeftEyePosition) {                                          ⑥
            UIView * leftEyeView = [[UIView alloc]
                                               initWithFrame:CGRectMake(0, 0, 5, 5)];
            [leftEyeView setCenter:faceFeature.leftEyePosition];                       ⑦
            leftEyeView.layer.borderWidth = 1;
            leftEyeView.layer.borderColor = [UIColor redColor].CGColor;
            [resultView addSubview:leftEyeView];
            [leftEyeView release];
```

```
        }

        //右眼
        if (faceFeature.hasRightEyePosition) {
            UIView * rightEyeView = [[UIView alloc]
                                        initWithFrame:CGRectMake(0, 0, 5, 5)];
            [rightEyeView setCenter:faceFeature.rightEyePosition];
            rightEyeView.layer.borderWidth = 1;
            rightEyeView.layer.borderColor = [UIColor redColor].CGColor;
            [resultView addSubview:rightEyeView];
            [rightEyeView release];
        }

        //嘴巴
        if (faceFeature.hasMouthPosition) {
            UIView * mouthView = [[UIView alloc]
                                        initWithFrame:CGRectMake(0, 0, 10, 5)];
            [mouthView setCenter:faceFeature.mouthPosition];
            mouthView.layer.borderWidth = 1;
            mouthView.layer.borderColor = [UIColor redColor].CGColor;
            [resultView addSubview:mouthView];
            [mouthView release];
        }

    }

    [resultView setTransform:CGAffineTransformMakeScale(1, -1)];                    ⑧

    [resultView release];

    if ([detectResult count] > 0)                                                   ⑨
    {
        CIImage * faceImage = [image imageByCroppingToRect:
                                [[detectResult objectAtIndex:0] bounds]];           ⑩

        UIImage * face = [UIImage imageWithCGImage:
                    [context createCGImage:faceImage fromRect:faceImage.extent]];
        self.outputImageView.image = face;

        [self.button setTitle:[NSString stringWithFormat:@"识别 人脸数 %i",
                    [detectResult count]] forState:UIControlStateNormal];
    }

}
```

其中，第①行代码是准备识别参数，参数是放在 NSDictionary 中的，本例中的 CIDetector-Accuracy 代表识别精度，CIDetectorAccuracyHigh 代表识别精度为“高”。

第②行代码是创建 CIDetector 对象，需要指定识别类型，目前只有 CIDetectorTypeFace 类型（人脸类型）。

第③行代码 NSArray * detectResult = [faceDetector featuresInImage:image]是进行识别，识别的结果放到 NSArray 集合中，集合中的每一个元素是 CIFaceFeature 类型。

第④行代码创建一个 UIView 对象，将在上面标识出脸、眼睛和嘴巴的位置。

第⑤行代码是从 NSArray 集合中取出 CIFaceFeature 元素，CIFaceFeature 元素代表一个识别出来的对象。

第⑥行代码 faceFeature. hasLeftEyePosition 是判断是否识别左眼，类似还有 faceFeature. hasRightEyePosition 判断是否识别右眼，faceFeature. hasMouthPosition 判断是否识别嘴巴。

第⑦行代码 faceFeature. leftEyePosition 是获得左眼的位置，类似还有 faceFeature. rightEyePosition 获得右眼的位置，faceFeature. mouthPosition 获得嘴巴的位置。

第⑧行代码[resultView setTransform:CGAffineTransformMakeScale(1, -1)]是沿 y 轴进行镜像变换，因为 Core Image 坐标是在左下角，UIKit 的坐标在左上角，需要进行坐标变换。

第⑨行之后的处理是将识别出来的脸部图像，显示在屏幕的下半部分。

第⑩行代码是 CIImage * faceImage = [image imageByCroppingToRect:[[detectResult objectAtIndex:0] bounds]]，是从识别出人脸位置中裁剪出 CIImage 图像，其中，[detectResult objectAtIndex:0]是取出第一个识别出来的 CIFaceFeature 对象，CIFaceFeature 对象的 bounds 方法可以获得其位置信息。imageByCroppingToRect:方法可以按照指定的矩形裁剪 CIImage 图像。

本章小结

通过对本章的学习，读者可以了解 UIImage、CIImage 和 CGImage 对象的不同及其应用场景。此外还介绍了滤镜的使用和人脸识别。