GPUImage 官方文档翻译

翻译者:CC老师

> - **对GPUImage 框架的文档,做了一个简单翻译.希望能给这一块学习的同学帮助.当然其中的生僻词汇,借助了翻译工具.本着分享和自我英语和技术提升的目的,特此翻译了此文档.**
> - **如有错误,欢迎指正!勿喷!文明指正!**
> - **此文档只做技术分享,不能用于商用!**
> - **转载文章,请标明文章出处!**

GPUImage 介绍

- GPUImage框架下载地址:https://github.com/BradLarson/GPUImage

The GPUImage framework is a BSD-licensed iOS library that lets you apply GPU-accelerated filters and other effects to images, live camera video, and movies. In comparison to Core Image (part of iOS 5.0), GPUImage allows you to write your own custom filters, supports deployment to iOS 4.0, and has a simpler interface. However, it currently lacks some of the more advanced features of Core Image, such as facial detection.

（GPUImage框架是一个BSD(伯克利软件套件)许可iOS库，能让你的APP应用GPU加速的过滤器和其他图像处理效果，现场摄像机视频和movies。在Core Image对比（iOS 5的一部分），GPUImage允许你添加自己的自定义过滤器，支持部署到iOS 4，并有一个简单的接口。然而，它目前缺乏Core Image的一些更高级的特性，如人脸检测。）

For massively parallel operations like processing images or live video frames, GPUs have some significant performance advantages over CPUs. On an iPhone 4, a simple image filter can be over 100 times faster to perform on the GPU than an equivalent CPU-based filter.

（大规模用来处理图像或视频直播框架，GPUImage框架有显著的性能优

势。在iPhone 4上，一个简单的图像过滤器在GPU上执行的速度比同等CPU的过滤器快100倍以上。）

However, running custom filters on the GPU requires a lot of code to set up and maintain an OpenGL ES 2.0 rendering target for these filters. I created a sample project to do this:

（然而，在GPU上运行自定义过滤器需要大量的代码来设置和维护这些过滤器的OpenGL ES 2渲染目标。我创建了一个示例项目来做这件事：）

and found that there was a lot of boilerplate code I had to write in its creation. Therefore, I put together this framework that encapsulates a lot of the common tasks you'll encounter when processing images and video and made it so that you don't need to care about the OpenGL ES 2.0 underpinnings.

(你会发现有大量的样板代码我已经写在其创作中。因此，我将这个框架封装起来，封装了处理图像和视频时遇到的许多常见任务，使您不必关心OpenGL ES 2基础。)

This framework compares favorably to Core Image when handling video, taking only 2.5 ms on an iPhone 4 to upload a frame from the camera, apply a gamma filter, and display, versus 106 ms for the same operation using Core Image. CPU-based processing takes 460 ms, making GPUImage 40X faster than Core Image for this operation on this hardware, and 184X faster than CPU-bound processing. On an iPhone 4S, GPUImage is only 4X faster than Core Image for this case, and 102X faster than CPU-bound processing. However, for more complex operations like Gaussian blurs at larger radii, Core Image currently outpaces GPUImage.

（GPUImage框架在处理视频时与Core Image相比是有利的，在iPhone 4上只需2.5毫秒就可以从照相机上传帧，应用gamma滤波器，并使用Core Image对同一操作显示106毫秒。基于CPU的处理需要460毫秒，使GPUImage 40x核心图像比这个操作在该硬件更快，和184x速度比CPU绑定的处理。在iPhone 4S，GPUImage只有快4倍比核心的形象，这种情况下，和102x速度比CPU绑定的处理。然而，对于更复杂的操作，如高斯模糊半径较大，目前超过GPUImage核心形象。）

Technical requirements（技术支持）

OpenGL ES 2.0: Applications using this will not run on the original iPhone, iPhone 3G, and 1st and 2nd generation iPod touches

（OpenGL ES 2：应用程序将不会运行在最初的iPhone，例如iPhone 3G和第一代和第二代iPod触摸）

iOS 4.1 as a deployment target (4.0 didn't have some extensions needed for movie reading). iOS 4.3 is needed as a deployment target if you wish to show live video previews when taking a still photo.

（iOS 4.1作为部署目标的（4.0比没有电影阅读所需的扩展）。如果您希望在拍摄静止照片时显示实时视频预览，则需要iOS 4.3作为部署目标。）

iOS 5.0 SDK to build

Devices must have a camera to use camera-related functionality (obviously)

（显然需要必须有一个摄像机来应用与相机相关的功能）

The framework uses automatic reference counting (ARC), but should support projects using both ARC and manual reference counting if added as a subproject as explained below. For manual reference counting applications targeting iOS 4.x, you'll need add -fobjc-arc to the Other Linker Flags for your application project.

(GPUImage框架使用自动引用计数（ARC），但要支持的项目，如果添加一个子项目解释如下使用手动引用计数。手动引用计数的应用针对iOS 4.X系统，你需要添加-fobjc-arc的其他连接标记到你的应用程序项目。)

General architecture （普遍结构）

GPUImage uses OpenGL ES 2.0 shaders to perform image and video manipulation much faster than could be done in CPU-bound routines. However, it hides the complexity of interacting with the OpenGL ES API in a simplified Objective-C interface. This interface lets you define input sources for images and video, attach filters in a chain, and send the resulting processed image or video to the screen, to a UIImage, or to a movie on disk.

GPUImage使用OpenGL ES 2着色器进行图像和视频处理速度远远超过可以在CPU绑定的程序做的。然而，它隐藏在OpenGLES API简化Objective-C接口OpenGL交互的复杂性。这个接口允许您定义的图像和

视频输入源，链连接的过滤器，并发送处理结果的图像或视频的画面到屏幕，一个UIImage，或磁盘上的一个movie。

Images or frames of video are uploaded from source objects, which are subclasses of GPUImageOutput. These include GPUImageVideoCamera (for live video from an iOS camera), GPUImageStillCamera (for taking photos with the camera), GPUImagePicture (for still images), and GPUImageMovie (for movies). Source objects upload still image frames to OpenGL ES as textures, then hand those textures off to the next objects in the processing chain.

（视频图像或帧从源对象的上传，这是GPUImageOutput。这些包括GPUImageVideoCamera（从iOS相机录制视频）、GPUImageStillCamera（带相机的照片），GPUImagePicture（静态图片），和GPUImageMovie（电影）。源对象将图像帧上传到OpenGL ES作为纹理，然后将这些纹理传递给处理链中的下一个对象。）

Filters and other subsequent elements in the chain conform to the GPUImageInput protocol, which lets them take in the supplied or processed texture from the previous link in the chain and do something with it. Objects one step further down the chain are considered targets, and processing can be branched by adding multiple targets to a single output or filter.

（链中的过滤器和其他随后的元素符合GPUImageInput协议，这让他们以提供或加工纹理从链中的上一个链接，用它做什么。在链上一步一步的对象被认为是目标，并且处理可以通过将多个目标添加到单个输出或过滤器来进行分支）

For example, an application that takes in live video from the camera, converts that video to a sepia tone, then displays the video onscreen would set up a chain looking something like the following:

(例如，一个应用程序，需要在摄像头获取视频，再转换视频到深褐色调，然后显示视频屏幕将建立一个链，看起来过程有点像下面：)

> GPUImageVideoCamera -> GPUImageSepiaFilter -> GPUImageView

Adding the static library to your iOS project（添加静态库到iOS项目中）

Note: if you want to use this in a Swift project, you need to use the steps in the "Adding this as a framework" section instead of the following. Swift needs modules for third-party code.

（注意：如果你现在将CPUImage使用在Swift 项目，你需要使用“Adding this as a framework”这章的步骤。Swift需要使用第三方代码）

Once you have the latest source code for the framework, it's fairly straightforward to add it to your application. Start by dragging the GPUImage.xcodeproj file into your application's Xcode project to embed the framework in your project. Next, go to your application's target and add GPUImage as a Target Dependency. Finally, you'll want to drag the libGPUImage.a library from the GPUImage framework's Products folder to the Link Binary With Libraries build phase in your application's target.

一旦你有了框架的最新源代码，将其添加到应用程序中就相当简单了。首先拖动GPUImage.xcodeproj到你的应用程序的Xcode项目嵌入框架在你的项目中。接下来，去你的应用程序的目标和添加GPUImage作为Target的依赖。最后，你要把libGPUImage.a静态库。添加到从GPUImage框架的产品Link Binary With Libraries build phase在你的项目中。

GPUImage needs a few other frameworks to be linked into your application, so you'll need to add the following as linked libraries in your application target:

GPUImage需要一些其他的框架被连接到你的应用程序，所以您需要添加以下链接库应用程序中的目标：

> 1.CoreMedia
> 2.CoreVideo
> 3.OpenGLES
> 4.AVFoundation
> 5.QuartzCore

You'll also need to find the framework headers, so within your project's build settings set the Header Search Paths to the relative path from your application to the framework/ subdirectory within the GPUImage source

directory. Make this header search path recursive.

你还需要找到框架的头文件，所以在项目GPUImage源目录应用程序的生成设置头文件搜索路径的framework/ subdirectory。使这个头文件搜索路径递归。

To use the GPUImage classes within your application, simply include the core framework header using the following:

在你的应用中使用GPUImage类，仅包括使用以下核心框架头文件：

```
#import "GPUImage.h"
```

As a note: if you run into the error "Unknown class GPUImageView in Interface Builder" or the like when trying to build an interface with Interface Builder, you may need to add -ObjC to your Other Linker Flags in your project's build settings.

作为一个提示：如果你遇到错误“Unknown class GPUImageView in Interface Builde（未知类GPUImageView在界面生成”或类似于试图用界面生成器生成界面，您可能需要add -ObjC to your Other Linker Flags在项目的生成设置中。

Also, if you need to deploy this to iOS 4.x, it appears that the current version of Xcode (4.3) requires that you weak-link the Core Video framework in your final application or you see crashes with the message "Symbol not found: _CVOpenGLESTextureCacheCreate" when you create an archive for upload to the App Store or for ad hoc distribution. To do this, go to your project's Build Phases tab, expand the Link Binary With Libraries group, and find CoreVideo.framework in the list. Change the setting for it in the far right of the list from Required to Optional.

另外，如果你需要部署到iOS 4.X系统，看来，Xcode的当前版本（4.3）要求你weak-link的Core Video框架，最终的应用程序或您看到崩溃日志“Symbol not found: _CVOpenGLESTextureCacheCreate：“当你创建上传到App Store或广告组织分布的档案。要做到这一点，去你的项目Build Phases tab，扩大the Link Binary With Libraries group，并在列表中找到CoreVideo.framework。在列表右侧的设置将必选修改为可选。

Additionally, this is an ARC-enabled framework, so if you want to use this

within a manual reference counted application targeting iOS 4.x, you'll need to add -fobjc-arc to your Other Linker Flags as well.

此外，这是一个ARC-enabled框架，所以如果你想用这个在手动引用计数的应用针对iOS 4。X，你需要add -fobjc-arc在你其他的链接标志中。

> Building a static library at the command line（在命令行中构建静态库）

If you don't want to include the project as a dependency in your application's Xcode project, you can build a universal static library for the iOS Simulator or device. To do this, runbuild.shat the command line. The resulting library and header files will be located atbuild/Release-iphone. You may also change the version of the iOS SDK by changing theIOSSDK_VERvariable inbuild.sh(all available versions can be found usingxcodebuild -showsdks).

如果你不想引入的项目作为依赖在你的应用的Xcode项目中，您可以为iOS模拟器或设备建立一个通用的静态库。要做到这一点，在命令行runbuild.shat。生成的库和头文件将位于atbuild/Release-iphone中。你也可以通过改变build.sh的IOSSDK_VER变量改变iOS SDK版本（所有可用的版本可以找到并使用xcodebuild - showsdks）

Adding this as a framework (module) to your Mac or iOS project

（将此作为框架（模块）添加到Mac或iOS项目中）

Xcode 6 and iOS 8 support the use of full frameworks, as does the Mac, which simplifies the process of adding this to your application. To add this to your application, I recommend dragging the .xcodeproj project file into your application's project (as you would in the static library target).

Xcode 6和iOS 8支持全框架的使用，如MAC，从而简化了添加到您的应用程序的过程。要添加到您的应用程序，我建议拖。xcodeproj项目文件到你的应用程序的项目（你会在in the static library target）。

For your application, go to its target build settings and choose the Build Phases tab. Under the Target Dependencies grouping, add GPUImageFramework on iOS (not GPUImage, which builds the static

library) or GPUImage on the Mac. Under the Link Binary With Libraries section, add GPUImage.framework.

对于你的app,去它的target build settings 并且选择 Build Phases标签。在目标相关性分组下，添加GPUImageFramework 在iOS（不是GPUImage，建立静态库） 或者 Mac。在 Link Binary With Libraries 部分，添加GPUImage.framework

This should cause GPUImage to build as a framework. Under Xcode 6, this will also build as a module, which will allow you to use this in Swift projects. When set up as above, you should just need to use

你应该使用GPUImage 建立一个框架。在Xcode 6.0，这会建立一个模块，如上所述它会允许你使用在 Swift 项目。

```
#import GPUImage
```

导入GPUImage文件

You then need to add a new Copy Files build phase, set the Destination to Frameworks, and add the GPUImage.framework build product to that. This will allow the framework to be bundled with your application (otherwise, you'll see cryptic "dyld: Library not loaded: @rpath/GPUImage.framework/GPUImage" errors on execution).

你需要添加一个新的拷贝文件在build phase。设置目标框架，并添加GPUImage.framework在已建立的项目中。它将允许框架内被捆绑你的应用程序。（否则，你将看到"dyld: Library not loaded: @rpath/GPUImage.framework/GPUImage" 错误警告）

Documentation （文档）

Documentation is generated from header comments using appledoc. To build the documentation, switch to the "Documentation" scheme in Xcode. You should ensure that "APPLEDOC_PATH" (a User-Defined build setting) points to an appledoc binary, available on Github or through Homebrew. It will also build and install a .docset file, which you can view with your favorite documentation tool.

文件由标题评论使用appledoc生成。建立文档，切换到“文档”方案在Xcode。你应该确定“APPLEDOC_PATH”（一个用户定义的编译设置）

指向一个appledoc二进制，可以在GitHub上或通过自制。它还将建设和安装 a.docset文件，你可以用你收藏的文件查看工具。

Performing common tasks（**执行常见任务**）

Filtering live video（**滤波视频的直播**）

To filter live video from an iOS device's camera, you can use code like the following:

要从iOS设备的摄像头过滤实时视频，您可以使用如下代码：

```
GPUImageVideoCamera *videoCamera = [[GPUImageVideoCamera alloc] initWithSessionPreset:AVCaptureSessionPreset640x480 cameraPosition:AVCaptureDevicePositionBack];
videoCamera.outputImageOrientation = UIInterfaceOrientationPortrait;

GPUImageFilter *customFilter = [[GPUImageFilter alloc] initWithFragmentShaderFromFile:@"CustomShader"];
GPUImageView *filteredVideoView = [[GPUImageView alloc] initWithFrame:CGRectMake(0.0, 0.0, viewWidth, viewHeight)];

// Add the view somewhere so it's visible

[videoCamera addTarget:customFilter];
[customFilter addTarget:filteredVideoView];

[videoCamera startCameraCapture];
```

This sets up a video source coming from the iOS device's back-facing camera, using a preset that tries to capture at 640x480. This video is captured with the interface being in portrait mode, where the landscape-left-mounted camera needs to have its video frames rotated before display. A custom filter, using code from the file CustomShader.fsh, is then set as the target for the video frames from the camera. These filtered video frames are finally displayed onscreen with the help of a UIView subclass that can present the filtered OpenGL ES texture that results from this pipeline.

视频源来自iOS设备的后置摄像头，使用预设，试图捕捉在640x480。此视频被捕获的界面是在纵向模式，其中左侧的左侧安装的相机需要在显示前旋转其视频帧。自定义过滤器，使用从文件CustomShader.fsh代码，然后设置为目标，从相机的视频帧。这些过滤的视频帧，最后显示在屏幕上以一个UIView子类可以过滤的OpenGL ES纹理从管道获得结果。

The fill mode of the GPUImageView can be altered by setting its fillMode property, so that if the aspect ratio of the source video is different from that of the view, the video will either be stretched, centered with black bars, or zoomed to fill.

通过设置填充模式性能改变可以设置GPUImageView的填充模式。因此，如果源视频的纵横比是不同的。视频会被拉长，以black bars为中心或者填充放大。

For blending filters and others that take in more than one image, you can create multiple outputs and add a single filter as a target for both of these outputs. The order with which the outputs are added as targets will affect the order in which the input images are blended or otherwise processed.

对于混合过滤器和其他容纳多图像，你可以选择2个输出和添加一个单独的过滤器作为目标。把输出添加作为目标的顺序，会影响深入图像的混合和其他方式的处理顺序。

Also, if you wish to enable microphone audio capture for recording to a movie, you'll need to set the audioEncodingTarget of the camera to be your movie writer, like for the following:

另外，如果你想使麦克风音频捕捉到movie，你需要设置相机的audioEncodingTarget是你的movie 写入者，如下列：

```
videoCamera.audioEncodingTarget = movieWriter;
```

Capturing and filtering a still photo （捕捉和过滤静态图片）

To capture and filter still photos, you can use a process similar to the one for filtering video. Instead of a GPUImageVideoCamera, you use a GPUImageStillCamera:

要捕捉和过滤静态图片，你可以使用类似于过滤视频的过程。而不是使用GPUImageVideoCamera，你需要使用GPUImageStillCamera：

```
tillCamera = [[GPUImageStillCamera alloc] init];
stillCamera.outputImageOrientation = UIInterfaceOrientationPor
trait;

filter = [[GPUImageGammaFilter alloc] init];
[stillCamera addTarget:filter];
GPUImageView *filterView = (GPUImageView *)self.view;
[filter addTarget:filterView];

[stillCamera startCameraCapture];
```

This will give you a live, filtered feed of the still camera's preview video. Note that this preview video is only provided on iOS 4.3 and higher, so you may need to set that as your deployment target if you wish to have this functionality.

这将给你一个静物，过滤静态相机的预览视频。注意，此预览视频必须在iOS4.3 或者以上系统。所以你如果需要这个功能，设置你的部署目标（deployment target）。

Once you want to capture a photo, you use a callback block like the following:

一旦你想过滤一张图片，你会使用到一个回调block。如下所示：

```
[stillCamera capturePhotoProcessedUpToFilter:filter withComple
tionHandler:^(UIImage *processedImage, NSError *error){
    NSData *dataForJPEGFile = UIImageJPEGRepresentation(proces
sedImage, 0.8);

    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDoc
umentDirectory, NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];

    NSError *error2 = nil;
    if (![dataForJPEGFile writeToFile:[documentsDirectory stri
ngByAppendingPathComponent:@"FilteredPhoto.jpg"] options:NSAto
```

```
micWrite error:&error2])
    {
        return;
    }
}];
```

The above code captures a full-size photo processed by the same filter chain used in the preview view and saves that photo to disk as a JPEG in the application's documents directory.

上面的代码在捕获预览层中使用一个通过同一个过滤链处理的全尺寸的图片，并把照片作为一个JPEG存储在项目里的documents directory中

Note that the framework currently can't handle images larger than 2048 pixels wide or high on older devices (those before the iPhone 4S, iPad 2, or Retina iPad) due to texture size limitations. This means that the iPhone 4, whose camera outputs still photos larger than this, won't be able to capture photos like this. A tiling mechanism is being implemented to work around this. All other devices should be able to capture and filter photos using this method.

注意：由于纹理大小的限制。目前框架无法处理老设备上大于2048像素或更高的图像（在iPhone 4S, iPad 2, or Retina iPad）。这意味着iPhone4 的摄像头输出的照片比这还大。无法捕捉到这样的照片。正在实施一种平铺机制来解决这个问题。所有其他设备都应该能够使用这种方法捕获和过滤照片。

Processing a still image(处理静态图片)

There are a couple of ways to process a still image and create a result. The first way you can do this is by creating a still image source object and manually creating a filter chain:

有两种方法来处理静止图像并创建结果。第一种方法是创建静态图像源对象并手动创建过滤器链：

```
UIImage *inputImage = [UIImage imageNamed:@"Lambeau.jpg"];

GPUImagePicture *stillImageSource = [[GPUImagePicture alloc] i
```

```
nitWithImage:inputImage];
GPUImageSepiaFilter *stillImageFilter = [[GPUImageSepiaFilter
alloc] init];

[stillImageSource addTarget:stillImageFilter];
[stillImageFilter useNextFrameForImageCapture];
[stillImageSource processImage];

UIImage *currentFilteredVideoFrame = [stillImageFilter imageFr
omCurrentFramebuffer];
```

Note that for a manual capture of an image from a filter, you need to set -useNextFrameForImageCapture in order to tell the filter that you'll be needing to capture from it later. By default, GPUImage reuses framebuffers within filters to conserve memory, so if you need to hold on to a filter's framebuffer for manual image capture, you need to let it know ahead of time.

注意，手动捕获的图像从一个过滤器，你需要设置-useNextFrameForImageCapture为了告诉过滤器，你需要捕获它以后。默认情况下，GPUImage 复用帧缓存framebuffers内的过滤器来节省内存，所以如果你需要坚持手工图像捕捉过滤器的帧缓存，你需要让它提前知道。

For single filters that you wish to apply to an image, you can simply do the following:

对于希望应用于图像的单个过滤器，可以简单地执行以下操作：

```
GPUImageSepiaFilter *stillImageFilter2 = [[GPUImageSepiaFilter
 alloc] init];
UIImage *quickFilteredImage = [stillImageFilter2 imageByFilter
ingImage:inputImage];
```

Writing a custom filter(编写自定义过滤器)

One significant advantage of this framework over Core Image on iOS (as of

iOS 5.0) is the ability to write your own custom image and video processing filters. These filters are supplied as OpenGL ES 2.0 fragment shaders, written in the C-like OpenGL Shading Language.

一个显着的优势，这个框架的核心图像在iOS（为iOS 5）是能够编写自己的自定义图像和视频处理过滤器。这些过滤器提供OpenGL ES 2的片段着色器，写类似C语言 OpenGL着色语言GLSL。

A custom filter is initialized with code like

自定义过滤器与代码初始化

```
GPUImageFilter *customFilter = [[GPUImageFilter alloc] initWit
hFragmentShaderFromFile:@"CustomShader"];
```

where the extension used for the fragment shader is .fsh. Additionally, you can use the -initWithFragmentShaderFromString: initializer to provide the fragment shader as a string, if you would not like to ship your fragment shaders in your application bundle.

其中用于片段着色器的扩展名是.fsh(fragment shader)。此外，您可以使用- initwithfragmentshaderfromstring：初始化提供片段着色器是一个字符串，如果你不想在你的应用程序包括你的片段着色器。

Fragment shaders perform their calculations for each pixel to be rendered at that filter stage. They do this using the OpenGL Shading Language (GLSL), a C-like language with additions specific to 2-D and 3-D graphics. An example of a fragment shader is the following sepia-tone filter:

片段着色器对要在该过滤阶段渲染的每个像素执行它们的计算。他们这样做是使用OpenGL着色语言（GLSL），类C语言添加特定的二维和三维图形。一个片段着色器举例棕褐色调过滤器：

```
varying highp vec2 textureCoordinate;

uniform sampler2D inputImageTexture;

void main()
{
    lowp vec4 textureColor = texture2D(inputImageTexture, text
```

```
ureCoordinate);
    lowp vec4 outputColor;
    outputColor.r = (textureColor.r * 0.393) + (textureColor.g
 * 0.769) + (textureColor.b * 0.189);
    outputColor.g = (textureColor.r * 0.349) + (textureColor.g
 * 0.686) + (textureColor.b * 0.168);
    outputColor.b = (textureColor.r * 0.272) + (textureColor.g
 * 0.534) + (textureColor.b * 0.131);
    outputColor.a = 1.0;

    gl_FragColor = outputColor;
}
```

For an image filter to be usable within the GPUImage framework, the first two lines that take in the textureCoordinate varying (for the current coordinate within the texture, normalized to 1.0) and the inputImageTexture uniform (for the actual input image frame texture) are required.

用于图像滤波在GPUImage框架是可用的，前两行，纹理坐标变化（当前坐标内的纹理，归一化到1）和inputimagetexture（实际输入的图像帧的纹理）的要求。

The remainder of the shader grabs the color of the pixel at this location in the passed-in texture, manipulates it in such a way as to produce a sepia tone, and writes that pixel color out to be used in the next stage of the processing pipeline.

着色器把像素的颜色在这个位置在通过纹理操纵它以这样的方式来产生褐色色调，并写入像素颜色了用于处理管道的下一个阶段。

One thing to note when adding fragment shaders to your Xcode project is that Xcode thinks they are source code files. To work around this, you'll need to manually move your shader from the Compile Sources build phase to the Copy Bundle Resources one in order to get the shader to be included in your application bundle.

有一点要注意你的Xcode项目中添加片段着色器时，Xcode认为他们是源代码文件。为了解决这个问题，您需要手动将着色器从编译源构建阶段移

到复制包资源一，以便将着色器包含在应用程序包中。

Filtering and re-encoding a movie(视频滤镜和再编码)

Movies can be loaded into the framework via the GPUImageMovie class, filtered, and then written out using a GPUImageMovieWriter. GPUImageMovieWriter is also fast enough to record video in realtime from an iPhone 4's camera at 640x480, so a direct filtered video source can be fed into it. Currently, GPUImageMovieWriter is fast enough to record live 720p video at up to 20 FPS on the iPhone 4, and both 720p and 1080p video at 30 FPS on the iPhone 4S (as well as on the new iPad).

Movies可以装入框架通过GPUImageMovie类，过滤，然后使用GPUImageMovieWriter写出来。GPUImageMovieWriter也足够快的视频记录实时从iPhone 4的摄像头在640x480，所以直接过滤可以输入视频源。目前，gpuimagemoviewriter是足够快的速度录制720p的视频在20 fps的iPhone 4和720p和1080p的视频在iPhone 4S 30 FPS（以及新的iPad）。

The following is an example of how you would load a sample movie, pass it through a pixellation filter, then record the result to disk as a 480 x 640 h.264 movie:

下面是一个例子，你将如何装载样本的movie，通过一个像素化的过滤器，然后将结果记录到磁盘作为一个480 x 640的H.264格式的movie：

```
movieFile = [[GPUImageMovie alloc] initWithURL:sampleURL];
pixellateFilter = [[GPUImagePixellateFilter alloc] init];

[movieFile addTarget:pixellateFilter];

NSString *pathToMovie = [NSHomeDirectory() stringByAppendingPa
thComponent:@"Documents/Movie.m4v"];
unlink([pathToMovie UTF8String]);
NSURL *movieURL = [NSURL fileURLWithPath:pathToMovie];

movieWriter = [[GPUImageMovieWriter alloc] initWithMovieURL:mo
```

```
vieURL size:CGSizeMake(480.0, 640.0)];
[pixellateFilter addTarget:movieWriter];

movieWriter.shouldPassthroughAudio = YES;
movieFile.audioEncodingTarget = movieWriter;
[movieFile enableSynchronizedEncodingUsingMovieWriter:movieWri
ter];

[movieWriter startRecording];
[movieFile startProcessing];
```

Once recording is finished, you need to remove the movie recorder from the filter chain and close off the recording using code like the following:

一旦录制完成，您需要从过滤器链中删除 movie 记录器，并使用如下代码关闭录制：

```
[pixellateFilter removeTarget:movieWriter];
[movieWriter finishRecording];
```

A movie won't be usable until it has been finished off, so if this is interrupted before this point, the recording will be lost.

movie直到它结束才可用，所以如果在这一点之前被打断，录制的内容就将会丢失。

Interacting with OpenGL ES(与OpenGL ES 交互)

GPUImage can both export and import textures from OpenGL ES through the use of its GPUImageTextureOutput and GPUImageTextureInput classes, respectively. This lets you record a movie from an OpenGL ES scene that is rendered to a framebuffer object with a bound texture, or filter video or images and then feed them into OpenGL ES as a texture to be displayed in the scene.

GPUImage可以同时通过其GPUImageTextureOutput和GPUImageTextureInput类出口和进口使用OpenGLES纹理，分别。这可

以让你记录一个OpenGL ES的场景，呈现一个绑定的纹理缓存对象，过滤视频或图像到OpenGL ES纹理将在展示。

The one caution with this approach is that the textures used in these processes must be shared between GPUImage's OpenGL ES context and any other context via a share group or something similar.

这一谨慎的方法是，在这些过程中所使用的纹理必须GPUImage's之间共享的OpenGL ES上下文和其他任何环境中通过一个共享组或类似的东西。

Built-in filters(内置过滤器)

There are currently 125 built-in filters, divided into the following categories:

目前有125个内置滤波器，分为以下类别:

Color adjustments(色彩调整)

- `GPUImageBrightnessFilter` : Adjusts the brightness of the image **(调整图像的亮度)**
  - brightness: The adjusted brightness (-1.0 - 1.0, with 0.0 as the default)
  - **亮度：调整后的亮度（1 - 1，默认值为0）**
- `GPUImageExposureFilter` : Adjusts the exposure of the image**(调整图像的曝光度)**
  - exposure: The adjusted exposure (-10.0 - 10.0, with 0.0 as the default)
  - **曝光：调整后的曝光（10–10，默认为0)**
- `GPUImageContrastFilter` : Adjusts the contrast of the image**(调整图像的对比度)**
  - contrast: The adjusted contrast (0.0 - 4.0, with 1.0 as the default)
  - 对比度：调整后的对比度（0 - 4，默认值为1）
- `GPUImageSaturationFilter` : Adjusts the saturation of an image**(调整图像的饱和度)**
  - saturation: The degree of saturation or desaturation to apply to the

image (0.0 - 2.0, with 1.0 as the default)
    - **饱和度：饱和度或desaturation学位申请到的图像（0.0 - 1.0与2.0的，默认）**
- `GPUImageGammaFilter` : Adjusts the gamma of an image**(调整图像的gama效果)**
    - gamma: The gamma adjustment to apply (0.0 - 3.0, with 1.0 as the default)
    - gama(伽码):**应用于伽码调整(0-3,以1为默认值)**
- `GPUImageLevelsFilter` : Photoshop-like levels adjustment. The min, max, minOut and maxOut parameters are floats in the range [0, 1]. If you have parameters from Photoshop in the range [0, 255] you must first convert them to be [0, 1]. The gamma/mid parameter is a float >= 0. This matches the value from Photoshop. If you want to apply levels to RGB as well as individual channels you need to use this filter twice - first for the individual channels and then for all channels.
- `GPUImageLevelsFilter` :**类似于PhotoShop图形处理软件的水平调整.min,max,minOut,maxOut参数范围是在[0,1]之间,如果你又从PS图像处理软件在参数范围[0,255].你必须先将其转换为[0,1]. 其中Gamma/MID 参数是浮点类型.而且必须>=0.这与Photoshop的价值都是一样的.如果你想把应用到RGB管道或者单独的频道上,你需要使用这个过滤器2次,第一次应用个人管道,第二次为所有管道.**
- `GPUImageColorMatrixFilter` : Transforms the colors of an image by applying a matrix to them(色彩图像转换通过矩阵来应用它们)
    - `colorMatrix` : A 4x4 matrix used to transform each color in an image**(一个4x4矩阵用于每个变换的彩色图像中)**
    - `intensity` : The degree to which the new transformed color replaces the original color for each pixel **(新改变的颜色取代每个像素的原始颜色的程度)**
- `GPUImageRGBFilter` : Adjusts the individual RGB channels of an image(调整图形的RGB单个颜色通道)
    - red: Normalized values by which each color channel is multiplied. The range is from 0.0 up, with 1.0 as the default.**取值范围[0,1],默认值为1**

    - green:同上
    - blue:同上
- `GPUImageHueFilter` : Adjusts the hue of an image**(调整图像的色调)**
    - hue: The hue angle, in degrees. 90 degrees by default**(色调角度,默认90度)**
- `GPUImageVibranceFilter` : Adjusts the vibrance of an image**(调整图形的振动)**
    - vibrance: The vibrance adjustment to apply, using 0.0 as the default, and a suggested min/max of around -1.2 and 1.2, respectively.
    - vibrance:**振动调整申请,使用0作为默认值,取值范围[-1.2,1.2]**
- `GPUImageWhiteBalanceFilter` : Adjusts the white balance of an image.**(调整图像的白平衡)**
    - temperature: The temperature to adjust the image by, in ºK. A value of 4000 is very cool and 7000 very warm. The default value is 5000. Note that the scale between 4000 and 5000 is nearly as visually significant as that between 5000 and 7000.
    - **色温:调整图像色温,在4000度是非常冷,7000是非常温暖.默认值是5000.注意,介于4000和5000之间比例几乎与5000到7000之间的视觉感受是一样的.**
    - tint: The tint to adjust the image by. A value of -200 is very green and 200 is very pink. The default value is 0.
    - tint: **色调调整图像,-200,深绿;200是粉色.默认值为0**.
- `GPUImageToneCurveFilter` :Adjusts the colors of an image based on spline curves for each color channel.**(根据每个颜色通道的曲线调整图像的颜色)**
    - redControlPoints**:红色控制点**
    - greenControlPoints:**绿色控制点**
    - blueControlPoints:**蓝色控制点**
    - rgbCompositeControlPoints: The tone curve takes in a series of control points that define the spline curve for each color component, or for all three in the composite. These are stored as NSValue-wrapped CGPoints in an NSArray, with normalized X and Y coordinates from 0 - 1. The defaults are (0,0), (0.5,0.5), (1,1).

    - rgbCompositeControlPoints:**RGB复合控制点,色调曲线包含一系列控制点.这些控制点定义每个颜色分量的曲线.或者对于复合材料中所有3个.这些存储在NSArray,x,y坐标归一化.取值空间(0,1).默认值为(0,0) (0.5,0.5) (1,1);**

- `GPUImageHighlightShadowFilter` : Adjusts the shadows and highlights of an image**(调整图像的阴影和亮点)**
    - shadows: Increase to lighten shadows, from 0.0 to 1.0, with 0.0 as the default.(**增加阴影，从0到1，默认为0)**
    - highlights: Decrease to darken highlights, from 1.0 to 0.0, with 1.0 as the default.**(从1下降到0，1作为默认值)**

- `GPUImageHighlightShadowTintFilter` : Allows you to tint the shadows and highlights of an image independently using a color and intensity**(允许你使用颜色和强度独立地对图像的阴影和高亮进行着色)**
    - shadowTintColor: Shadow tint RGB color (GPUVector4). **Default: {1.0f, 0.0f, 0.0f, 1.0f} (red).(RGB阴影颜色,默认红色值 {1.0f, 0.0f, 0.0f, 1.0f} 4维向量表示)**
    - highlightTintColor: Highlight tint RGB color (GPUVector4). Default: {0.0f, 0.0f, 1.0f, 1.0f} (blue).**(突出RGB色彩,默认蓝色值 {0.0f, 0.0f, 1.0f, 1.0f})**
    - shadowTintIntensity: Shadow tint intensity, from 0.0 to 1.0. Default: 0.0 **(阴影的色彩强度,默认0.0,取值空间(0,1))**
    - highlightTintIntensity: Highlight tint intensity, from 0.0 to 1.0, with 0.0 as the default.**(突出色彩的强度,默认0.0,取值空间(0,1));**

- `GPUImageLookupFilter` : Uses an RGB color lookup image to remap the colors in an image. First, use your favourite photo editing application to apply a filter to lookup.png from GPUImage/framework/Resources. For this to work properly each pixel color must not depend on other pixels (e.g. blur will not work). If you need a more complex filter you can create as many lookup tables as required. Once ready, use your new lookup.png file as a second input for GPUImageLookupFilter.