

硕士研究生学位论文

|  |  |
| --- | --- |
| 题目： | 一款轻量级短视频SDK拍摄模块的设计与实现 |
|  |  |

|  |  |
| --- | --- |
| 姓 名： | 陈科宇 |
| 学 号： | 1701210838 |
| 院 系： | 软件与微电子学院 |
| 专 业： | 软件工程 |
| 研究方向： | 大数据技术与应用 |
| 导师姓名： | 蒋严冰 |

二〇二0 年 3月

版权声明

任何收存和保管本论文各种版本的单位和个人，未经本论文作者同意，不得将本论文转借他人，亦不得随意复制、抄录、拍照或以任何方式传播。否则，引起有碍作者著作权之问题，将可能承担法律责任。

# 摘要

实现一个短视频SDK, 是一个很庞大的工程.本文主要讨论如何实现一款轻量级的音视频SDK的拍摄模块, 而且聚焦于Android平台的实现. 拍摄模块的设计, 重点在渲染环节和编码. Android平台提供了GLSurfaceView, 提供了OpenGL的上下文, 可以直接利用现有的API渲染, 非常方便, 但缺陷是性能相对较差, 且不具备可移植性. 本文选择了c++实现, 使用Android平台的native库, 依靠EGL搭建OpenGL的上下文, 实现OpenGL的渲染, 预览时支持实时滤镜、美颜等功能, 用户录制后支持硬编码和软编码两种方式保存最终的视频文件. 最后暴露的依然是Java层接口,很好地隐藏了实现细节.SDK最终打包成aar, 提供给使用方.

**问题:**

**1.为什么说GLSurfaceView的性能比Native使用OpenGL差.[其实我搜索的主要还是因为可移植性, GLSurfaceView底层用的还是EGL库]**

关键词：短视频SDK，OpenGL.

ENGLISH TITLE

Chen Keyu

Jiang Yanbing

# ABSTRACT

When developing short video app, it`s quite important for the app developers only focus on the features So, for these apps, they often apply related SDK, we call it Short Video SDK. During recording, the app developers, just record and stop, they do not care about the camera

frame. But in some cases, they hope preprocess the frame, for example , juding the scene by a specific algorithm in advance, they need the camera frame data. So it`s not OK to bound the Recorder with Camera, we need decoupling. We find this approach is powerful to improve developing efficiency.

KEY WORDS: Short Video, SDK, decoupling.

# 目录

[摘要 1](#_Toc35787856)

[ABSTRACT 2](#_Toc35787857)

[目录 4](#_Toc35787858)

[第一章 引言 5](#_Toc35787859)

[1.1 什么是短视频SDK 6](#_Toc35787860)

[1.2 课题的基本背景 7](#_Toc35787861)

[1.3 主要研究内容 7](#_Toc35787862)

[1.4 研究的意义 7](#_Toc35787863)

[1.5 研究的方法 7](#_Toc35787864)

[1.6 论文的基本结构 7](#_Toc35787865)

[第二章 研究进展 8](#_Toc35787866)

[2.1 GPUImage的分析 8](#_Toc35787867)

[2.2我的视频录制的架构设计 10](#_Toc35787868)

[第三章 SDK拍摄模块的完整实现 16](#_Toc35787869)

[3.1音频模块的实现 16](#_Toc35787870)

[3.2音频编码模块的实现 16](#_Toc35787871)

[3.3画面采集与编码模块的实现 16](#_Toc35787872)

[3.4 Mux模块 16](#_Toc35787873)

[3.5添加实时滤镜 16](#_Toc35787874)

[3.6添加实时特效 16](#_Toc35787875)

[3.7添加美颜 16](#_Toc35787876)

[第四章 效果展示 17](#_Toc35787877)

[第五章 结论及展望 18](#_Toc35787878)

[参考文献 19](#_Toc35787879)

[附录A 附录示例 20](#_Toc35787880)

[致谢 21](#_Toc35787881)

[北京大学学位论文原创性声明和使用授权说明 22](#_Toc35787882)

# 第一章 引言[[1]](#footnote-1)

音视频开发这几年因为抖音和快手的火爆又热起来.随着短视频App的兴起, 用户量的增加, 越来越多的App有了拥有短视频功能的需求, 而对中小企业而言, 要独立实现一套性能可观的短视频SDK的成本太高.这也使得短视频SDK有了发展机会, 其中像七牛云、美摄在这方面走得很快, 他们迅速研发了相关产品, 并快速迭代.短视频SDK的核心场景可以简单分为录播和直播. 本文主要讨论的是短视频SDK的录制模块的设计.

## 1.1 什么是短视频SDK

前面一直提到短视频SDK(Software Development Kit), 这个概念很抽象, 更准确地说, 这里提到的SDK是针对移动平台的, 主要是Android和iOS两大平台. 在早期的开发中, 短视频相关的代码往往和整个工程是一体的. 由于性能的需要, 这部分通常是c++实现的代码, 常称为native代码. 随着项目日益庞大, 音视频处理相关的代码和整个工程一体非常不利于维护, 也不利于不同项目的复用. 因此, 音视频处理的代码往往独立出来, 形成一个单独的库, 在开发时引入即可. 这个单独处理音视频的库, 我们统称为短视频SDK. 短视频SDK一般主要包含拍摄和编辑两大模块. 拍摄场景中, 通常会带上美颜、滤镜、贴纸等实时特效, 录制时也会涉及到编码.

社交网站的手机截图

描述已自动生成拍摄过程可以简单地抽象成如图1-1:

图1-1

## 1.2 课题的基本背景

根据艾媒咨询提供的数据, 如图1-2所示, 2018年中国短视频用户达到了5.01亿[1].以抖音和短视频为代表, 成为年轻人越来越喜欢的标志性短视频平台. 也有越来越多的企业参与到短视频应用开发的阵营中来. 仅腾讯旗下, 就有微视、企鹅看看、腾讯时光等10余款短视频产品[2]. 今日头条旗下, 除抖音外, 还有西瓜视频、火山小视频、皮皮虾等, 均在短视频领域各个出击. 激烈的竞争之下, 也催生了很多与手机厂商的直接合作, 将短视频App的特色功能直接植入到手机厂商的相机模块, 实现双赢. 此课题正是基于这种背景, 在本人在抖音项目组实习期间, 抖音开启了与华为、小米等厂商的合作, 将抖音的短视频能力植入到厂商的相机模块之中.

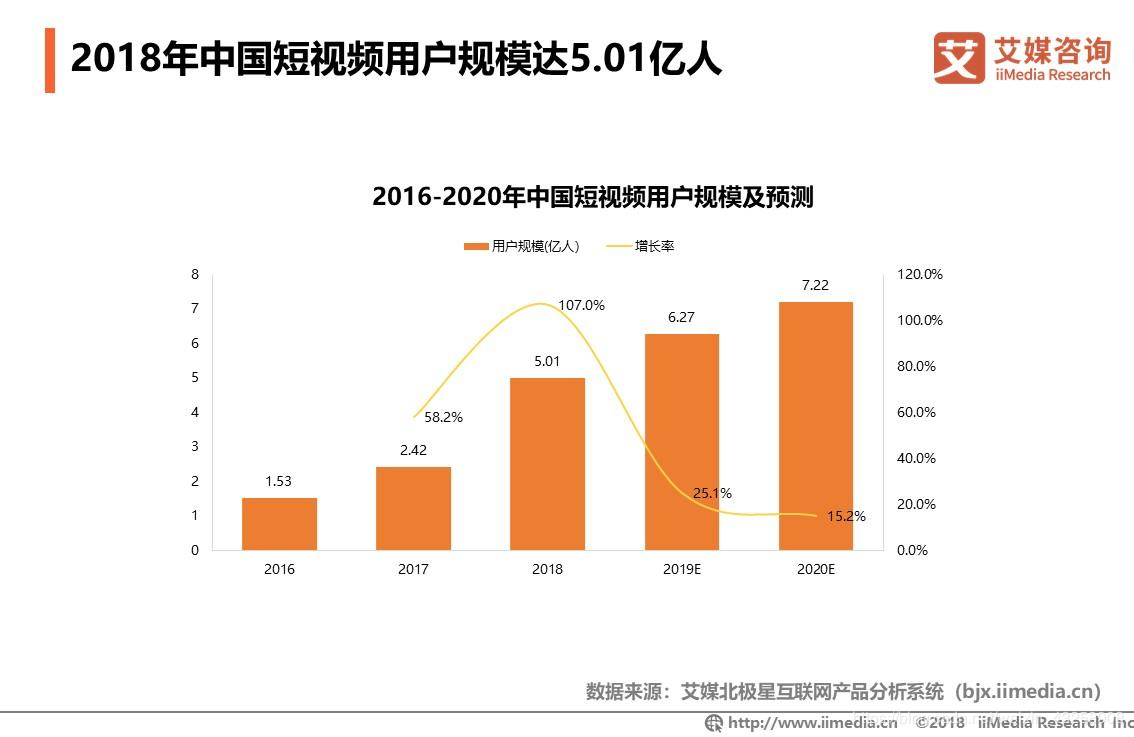


图1-2 2016-210年中国短视频用户规模及预测

## 1.3 主要研究内容

上个小节讨论了课题的背景. 本节主要讨论这个背景下, 抖音与厂商合作中, 本人所做的主要工作. 主要包括, 在Android平台上, 搭建c++层的OpenGL ES环境, c++层实现实时滤镜、美颜, 动态特效等功能, 并能添加背景音乐、录制带人声和不同画面效果的视频、拍摄带美颜照片等功能. 上述的功能, 均在c++层实现, 通过JNI,提供Java层接口. 整个模块的实现, 最后通过gradle编译成aar包, 提供给厂商使用. 有的厂商的合作形式,则不同, 例如三星, 则是制作一个完整的带录制功能的apk提供给他们, 这就需要我们自身基于这个sdk开发一个完整的录制应用提供给对方使用. 因此本文的主要内容是开发短视频SDK的拍摄模块, 并在完整SDK的基础上, 开发一个完整的录制应用.

## 1.4 研究的意义

这项工作对如何制作一个音视频SDK提供了完整的思路和实现过程, 对想要从事音视频开发的人士起到了一个引导作用, 可以从0到1构建一个短视频应用, 而不仅仅是局限于理论知识.另外, 很多从事移动开发的人员对native开发往往望而却步, 无从着手, 本文力求详细地描述整个流程, 步步拆解实现的难度, 减轻了理解上的负担.

## 1.5 研究的方法

抖音短视频SDK经历4年的迭代, 已经非常庞大. 本文所提到的短视频与厂商合作, 仅使用了一部分功能, 这些功能并不能简单地从原有的SDK中拆解出来. 举个例子, 在抖音工程中, 相机模块是封装在录制模块之中, Camera对上层使用是无感知的, 这样对抖音的客户端团队而言, 无需去掌控Camera的. 但在于厂商合作中, 他们需要独立控制Camera的使用. 因此提供给厂商的SDK是单独的一套, 是一个更简单功能更少的版本(厂商也不支持拥有他们不需要的功能.) 由于是植入系统之中, 对稳定性要求更高, 功能简单稳定就更加重要. 因此, 虽然最终提供给厂商的版本是在完整版的基础上实现的, 但也只是在架构上有所借鉴, 本质是从零开始在做, 而且要对完整版支持的一些功能进行兼容, 例如厂商指定的抖音特效、美颜和滤镜效果等.

研究的方法主要是先分清工作的重点. 对于拍摄环节而言, 可以简单划分为采集、渲染、编码三个环节.因此工作的核心方法, 是搞清楚每个环节的原理和实现方式. 对音视频而言, 在合成最终的文件之前, 三个环节都是独立的. 音频的采集、渲染、编码. 视频的采集、渲染和编码. 音频采集, Android系统提供了接口, 但是关于采样率、声道配置、音频格式等配置,则需要参考过往的经验, 渲染方面, 音频方面还需要注意个一个环节就是背景音乐的添加和播放, 这和录入的人声需要一起处理, 最终你做编码. 编码, 是最后一步, 由于Android平台的差异化, 还需要考量编码方式的问题, 这里我们选择了MediaCodec硬编, 但软编码也是支持的, 采用了主流的fdk\_aac库, 而不是重复造轮子. 硬编和软编, 需要封装, 外部调用时无需关心此细节, 如果硬编失败, 会切换到软编码. 视频方面, 采集也主要是关注分辨率、帧率等的设定. 画面的渲染环节, 是重中之重, 目前主流的使用OpenGL ES技术, 当时开发时, 主要基于2.0版本, 3.0仍然在实验中. 这套流程, 参考了GPUImage的实现, 更有扩展性.编码方面, 硬编使用MediaCodec, 软编使用libx264, 也是优先选择硬编码, 软编码作为备用方案. 最终利用ffmpeg的mux功能, 把音视频两个轨道合并成一个完整的mp4文件, 这样整个流程就打通了. 因此无论音频还是视频, 都是走的采集、渲染、编码这样一条线路, 然后利用ffmpeg打通. 这个过程也证实图1-1所反映的过程.

## 1.6 论文的基本结构

论文的基本结构是从音视频的基本技术介绍, 其中包括OpenGL ES、OpenSL ES

、ffmpeg等. 并对优秀的开源框架GPUImage做了详细的介绍. 介绍这部分目的是对工程实现的主要技术做基本的阐述, 以及为什么使用和借鉴它们. 接下里, 第3章, 主要做需求分析. 这个章的内容, 对需求做了细致的分析, 并梳理了用例图, 流程图和项目架构图. 在此基础上, 第4章介绍一个完整的录制模块的设计, 为了表达更加清晰, 视频效果单独用了第5章来介绍, 它是在第4章的基础上去扩展的. 第4章会首先介绍拍摄模块的基本架构, 然后分别介绍音频模块从采集到渲染再到编码的实现, 视频整体上也遵循这样一个流程, 最终将音频和视频合成一个文件. 第5章, 是对第4章中渲染视频做修改. 主要为了实现特效、美颜等, 而且在实现上都是高度可扩展的. 第5章之后, 一个基本的短视频SDK的拍摄模块就成型了, 编译成so文件, 打包aar, 会在第6章中进行介绍, aar最终是为了项目导入使用. 第6章会介绍一个录制应用的基本架构, 类图分析, 基本的UI设计等, 以及最终的效果展示. 录制应用的设计, 一方面是为了方便使用方更直接地了解SDK接口的使用, 另一方面, 也为使用方打造相关应用提供良好范例, 甚至会有厂商通过简单改造直接植入相机模块, 减少重复开发. 除此之外, 也能很好地用于测试SDK的性能. 第7章的测试是基于第6章的应用的, 该章会对测试环境和测试方法做详细的介绍. 以上的介绍就是整个论文的基本结构.

2.1 音视频的基础概念

这一小节主要介绍音频和视频的基础概念. 音频主要从采样、量化、编码3个方面做简单的介绍. 视频则主要介绍视频一帧的表示方式、视频的编码方式, 视频帧的I帧、P帧、B帧的简要说明.

2.1.1 音频的基础概念

平常听到的声音是如何采集到麦克风中的呢? 简单来说就是声音信号转换成电信号.麦克风里有一个碳膜, 碳膜的振动和声音的振动幅度和频率相关, 振动过程接触电极, 实现了声音信号转化成电信号.这个信号称为模拟信号, 而采样正是基于这时产生的模拟信号. 人耳所能听到的频率范围, 在20Hz-20kHz之间. 根据著名的奈奎斯特采样定理, 需要对信号的最高频率的2倍以上进行采样, 才能够保证声音的效果足够好.这也是为什么我们经常看到44.1Hz的采样率的缘故. 采样后, 每个信号的幅度值如何表示呢, 通常有1B或者2B两种形式.

接下来就是如何编码的问题. 提到编码, 先了解一下PCM, 通常称其为音频的裸数据格式. 以4分钟的数据为例, 假设采样率是44.1kHz, 而每次采样用2B表示, 其比特率为44.1k\*16/s = 705.6kps. 存储4分钟的数据则高达20MB左右. 相当于一首普通歌曲占用20MB左右.

最常见的有MP3、AAC编码, 以MP3为例,它的比特率在128Kbit/s以上表现非常好, 同样的4分钟, 则只需要占用128K\*4\*60/8 ≈ 4MB, 这个压缩比是非常惊人的, 这也是我们下载一首歌所占用的空间. 而且该编码方式的兼容性很好, 支持大量的软件和硬件.

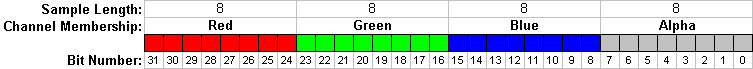
这篇文章的主角是AAC编码, MP3编码虽然很优秀, 但它一般在128Kbit/s以上的码率条件表现不错, 而AAC虽然是有损压缩, 但是其在低比特率条件下则比MP3要好不少, 因此一般视频中音频轨的编码都适用它.另外, 很多社交软件的语音也通常都是用AAC编码, 就是因为它比特率低, 空间和音质做到了很好的平衡.

关于音频就主要介绍上面的知识, 主要是阐述了采样、量化、编码的基本知识, 以及如何做编码方式的选择.

2.1.2 视频的基础概念

对视频而言, 首先需要明确的是, 视频与音频不同,视频有帧的概念, 视频的每一帧, 就相当于一张图片.在手机中,视频屏幕上的数据是RGB的表示形式,而实质上, 摄像头采集出的画面却是YUV的, 接下来会对两种格式作简单的介绍, 以及它们之间是如何转换的.

RGB表示方式,这个很容易理解,就是红绿蓝三原色. 更细致的区分,则有RGB\_565和RGBA\_8888.其中RGB\_565表示5位表示红色,6位表示绿色,后5位表示蓝色. RGBA中的A则表示透明度,如图2-1[3]所示:

 图2-1 RGBA\_8888的存储格式

手机屏幕的截图

描述已自动生成YUV格式,其中Y表示明亮度, U和V表示色度. YUV的发展,是在彩色电视产生的年代, 为了兼容黑白电视而发明的.如果只接受Y信号,则是黑白信号. YUV在存储方面和RGB不同, YUV往往不是1:1:1的关系,最主流的是YUV420.如图2-3[4]所示:

图2-3 YUV420的存储形式

420的含义是, 采集YUV时, 本次扫描取Y:U=4:2, 则下次扫描取Y:V=4:2存储, 因此Y:U:V=4:1:1,保存一张64的图片, 需要36B,而如果用RGB存储,则需要72B,因此该通用YUV格式占取的空间比RGB更有优势.

视频编码的核心是去除冗余信息.主要是通过帧间编码去除时间上的冗余信息, 帧内编码去除空间上的冗余信息, 如图2-4所示:

地图的截图

描述已自动生成

图2-4 视频帧的冗余信息分析

消除冗余信息, 目前ISO制定了MPEG算法, 即动态视频的压缩算法.MPEG算法, 不仅对单帧压缩编码处理, 还对图像序列中的冗余通过运动补偿等方式去除,大大提高了压缩比,最高达到了200:1的压缩比. 另外一个组织ITU则推出了H261、H262等系列.其中最为广泛使用的却是两个组织联合开发的标准H264,达到了300-400:1的压缩比.目前已经推出了性能更好的H265.

前面介绍了视频编码标准, 现在简单介绍编码后视频帧的概念.视频帧主要有3种帧:

I帧: I帧是独立完整的一个画面帧, 无需参考其他的帧,可以独立进行解码.尽管是完整的画面,I帧也是经过压缩的, 通常使用JPEG算法,达到了6:1的压缩比.

P帧:称为“帧间预测编码帧”. P帧存储的是变化的信息, 它会依赖于前面的P帧或者I帧.

B帧: 称为“双向预测编码帧”. 它往往需要参考前后帧的信息, 才能实现解码,因此只要出要B帧,必然不是按顺序解码的.这里需要提到两个基础概念DTS和PTS, DTS是解码时间戳,PTS是显示时间戳, 由于B帧的存在, DTS往往PTS不同.

图片包含 游戏机, 电脑, 画

描述已自动生成I、B、P帧的关系,可以用图2-5[5]描述:

图2-5 IBP帧的关系

IBP帧需要分组, 称为GOP(Group of Picture). GOP表示两个I帧之间的帧的数量, 这是编码设置时的重要指标.GOP的值越大, 表示BP帧更多.在相同的码率下,则有更多的空间来编码I帧, 因此画质也更好.

2.2 音频处理技术

      2.2.1 AudioRecoord采集介绍

AudioRecord用于Java层管理手机的音频输入. 使用它的原因是可以直接从内存读取PCM数据, 可以在渲染阶段对数据添加实时音效. AudioRecord的使用非常简单, 先初始化. 构造方法如下[7]:

[AudioRecord](https://developer.android.com/reference/android/media/AudioRecord#AudioRecord(int,%20int,%20int,%20int,%20int))(int audioSource, int sampleRateInHz, int channelConfig, int audioFormat, int bufferSizeInBytes)

audioSource表示数据的来源, 一般是麦克风.

sampleRateInHz表示采样率, 一般情形使用44100Hz即可, 这个采样率是可以保证在所有Androiod设备支持, 当然根据实际情况也可以设置22050, 16000等采样率.

channelConfig表示声道配置, 通常配置为单声道AudioFormat. CHANNEL\_IN\_MONO或者AudioFormat.CHANNEL\_IN\_STEREO. 立体声目前不是所有设备都能支持, 因此可以选择单声道即可[6]. 如果不知道该如何选择, 可以采用其默认设置CHANNEL\_IN\_DEFAULT.

audioFormat主要决定采样时数据的值的位数, 通常有ENCODING\_PCM\_8BIT, ENCODING\_PCM\_16BIT, ENCODING\_PCM\_FLOAT, 最常用的是ENCODING\_PCM\_16BIT, 支持的手机范围最广, 因此通常设置它即可.

最关键的参数是最后的bufferSizeInBytes, 这个参数决定的是音频缓冲区的大小, 值越小, 通常时延也越小, 但如果过小, 则又会导致IO过于频繁, 而不同手机由于性能各方面的差异,这个值是不一致的, 因此, 系统建议, 使用getMinBufferSize来设置该值,由系统判断音频缓冲区的大小.

参数设置是AudioRecord最重要的一环. 初始化后,可以检查AudioRecord的状态, 如果AudioRecord.STATE\_INITIALIZED才表示初始化成功.初始化成功后, 则可以开启录制, 调用startRecording方法. 与此同时, 需要开辟线程读取音频数据, 采用read方法. 例如:

public int read (byte[] audioData,

int offsetInBytes,

int sizeInBytes)

注意的是,这里的sizeInBytes也通常设置为getMinBufferSize获取的值, 如果设置过小, 则可能出现, 音频数据取走不及时, 导致溢出的错误.

另外, Android在read上采取了优化, Java层传统的I/O效率不高, 可以使用支持NIO的如下read方法:

[read](https://developer.android.com/reference/android/media/AudioRecord#read(java.nio.ByteBuffer,%20int))([ByteBuffer](https://developer.android.com/reference/java/nio/ByteBuffer) audioBuffer, int sizeInBytes)

ByteBuffer是一种Direct Buffer, 好处是操作它和native层的I/O操作是相当的, 因此非常推荐在开发时使用该read方法, 来取代上面的read方法.

录制结束后, 需要调用release方法结束对相关资源的占用. 音频采集接口AudioRecord就简单介绍到这里.

2.2.2 AudioTrack音频渲染介绍

这一小节主要介绍使用AudioTrack播放音频的流程.按照安卓官方的解释, AudioTrack是Java层管理播放音频资源的API[7], AudioTrack的使用和AudioRecord有相似之处, 主要分为以下几个步骤:

1: 根据音频的参数信息, 创建AudioTrack实例:

[AudioTrack](https://developer.android.com/reference/android/media/AudioTrack#AudioTrack(int,%20int,%20int,%20int,%20int,%20int))(int streamType,

int sampleRateInHz,

int channelConfig,

int audioFormat,

int bufferSizeInBytes,

int mode)

2. 调用AudioTrack的实例的play方法, 开始准备播放.

3. 开启一个播放线程, 用于解码音频数据, 并由AudioTrack的write方法不断写入数据用于底层调用播放功能.

4. 播放结束后, 则释放相关资源,调用AudioTrack的stop和release.

2.2.3 MediaCodec硬编码

手机屏幕截图

描述已自动生成MediaCodec是Android平台的硬编解码的接口, 它既可以用作音频的编码, 又可以用作视频的编码. 它的编码流程如图2-6[8]:

图2-6 MediaCodec的编码流程

音频使用和视频使用MediaCodec, 差别在于初始化的配置信息不同, 例如音频, 调用MeidaCodec的createEncoderByType时通常设置为“audio/mp4a-latm”, 而编码视频, 则设置为“video/avc”.

基本编码过程, 是MediaCodec调用dequeueInputBuffer方法获取empty input buffer, 然后通过queueInputBuffer填充数据, 交由MediaCodec编码, 然后调用MediaCodec的dequeueOutputBuffer取出编码后的数据, 根据需要做后续的处理, 最常见的操作是存入文件之中. 后续在讲解实现的时候会对实际实际使用的流程作更详细的介绍, 这里只做最基本的阐述.

2.2.4 fdk\_aac软编码

软件编码AAC,通常是基于ffmpeg处理. 关于ffmpeg, 后续会做更详细的介绍.

AAC的使用分为以下主要几个步骤:

1. 通过av\_register\_all将所有编码器注册道ffmpeg框架中.
2. 初始化avformat\_alloc\_output\_context2传入文件的输出路径, 生成用于输出的AVFormatContext信息.
3. 调用avcodec\_find\_encoder\_by\_name函数打开fdk\_aac编码器, 并用avcode\_open2打开该编码器.
4. 对于输入的每一帧数据,封装成AVFrame, 调用编码方法avcodec\_encode\_audio2, 最终把编码好的数据放入AVPacket中.
5. 对AVPacket数据, 调用av\_interleaved\_write\_frame方法, 将编码后的数据帧输出到最终的文件中去.

fdk\_aac的编码流程大致就是以上5个步骤.

2.3 OpenGL ES的基本介绍

OpenGL是一套跨平台、跨语言的图形编程接口, 主要用于二维和三维的图形图像的渲染. 而OpenGL ES, 则是专为移动平台开发的一套接口, 目前Open GL ES最新已经更新到OpenGL ES 3.3, 本文章的主要实现是在OpenGL ES 3.0平台上. 因此不会介绍3.0及其之后的相关特性.

OpenGL的核心工作是图形图像的渲染, 它有一套自己的语言GLSL(GL Shading Language). 它有两个核心环节, 顶点着色器和片段着色器.后序章节要介绍的视频画面的美颜、滤镜等效果就是在OpenGL ES的环境下实现的. 各种图形图像的着色器, 可以实现亮度、对比度、饱和度、高斯模糊等的调节, 视频效果亦是如此, 都主要依赖于着色器.

OpenGL是如何渲染的呢? 下面简单讲述着色器的工作机制, 这个也通常称为OpenGL的图形管线. 其工作流程如图2-7[9]:

图片包含 游戏机

描述已自动生成

图2-7 OpenGL ES3.0图形管线

第一阶段:顶点处理

这个阶段的主要操作, 是对输入的顶点坐标进行变换, 主要是根据模型视图矩阵改变输入顶点的位置, 纹理矩阵改变纹理的位置. 总结下来, 就是指定和变换顶点位置.

第二阶段: 图元组装

在第一阶段中, 需要制定图元. 图元即是基本绘制单元, 可以是点、线、三角形三种. 这个阶段就是根据第一阶段的顶点, 将纹理组装成图元.

第三阶段: 栅格化

第二阶段的图元, 粒度往往还比较大, 不能用于直接渲染. 因此需要做进一步的拆分, 把图元拆分成片元. 拆分后, 每个片元会带上坐标、渲染颜色等信息, 用于后续的渲染操作.

第四阶段: 片元处理

片元处理是片元处理器或片段处理器的核心工作, 片段处理器可以根据业务需求, 对每个片元做特定的处理, 例如提高对比度, 高斯模糊等操作.注意, 这里的输出就代表了每个片元的最终颜色了, 也是每个像素的最终结果.

第五阶段: 帧缓冲区

在Android平台上, 有双缓冲机制, 就是片元处理的结果并不是直接送给屏幕显示, 而是在先写入到后台缓冲区, 结束后, 再与前台缓冲区的内容进行交换, 实现把后台缓冲区的内容展现的目的, 到此, 整个渲染过程就结束了.

前面讲述了图形管线的流程, 但未涉及到从程序角度, OpenGL是如何工作的, 下面对图形管线的两个核心组件顶点着色器(Vertex Shader)和片元着色器(Fragment Shader)在显卡中的工作流程进行介绍, 如图2-8:

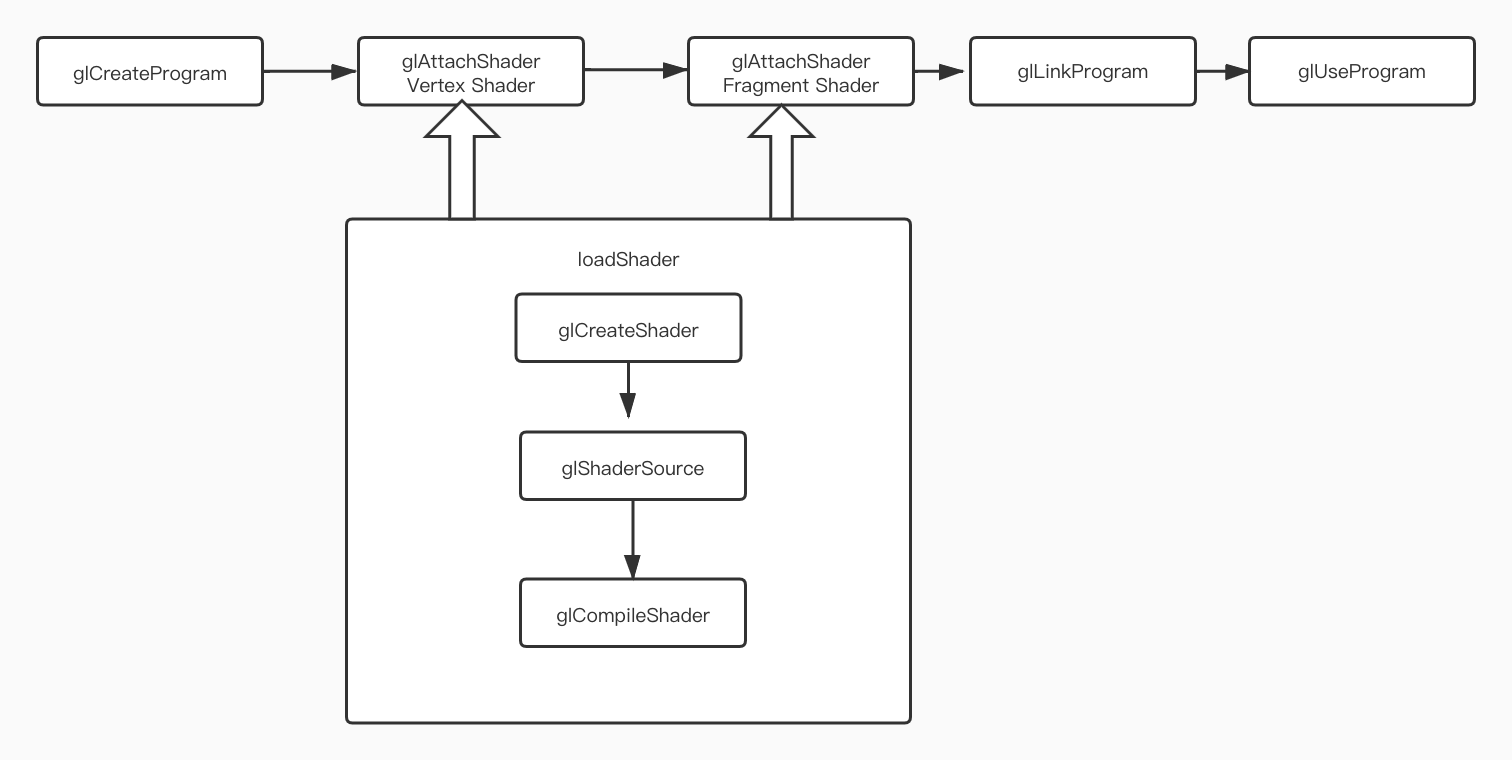


图2-8 openGL程序的执行流程

1. 调用glCreateProgram,创建显卡程序Program.
2. 创建VertexShader.首先通过glCreateShader创建shader句柄, 然后glShaderSource加载shader的资源, 最后通过glCompileShader编译shader,编译成功则创建成功.
3. FragmentShader的创建过程与VertexShader的创建过程类似.
4. VertexShader和FragmentShader创建后, 需要attach到Program中去.需要注意的是, 必须先attach VertexShader.
5. glLinkProgram, 类似于链接程序.
6. glUseProgram, 则启动该段程序.

另外, 在OpenGL ES的使用中, 有另外两个概念非常重要. 其一是上下文. OpenGL ES提供了图形接口, 但对于不同平台, 则需要独立实现上下文环境, 其中, Android提供了EGL native接口, 供开发者搭建OpenGL ES环境.OpenGL ES上下文的主要作用是实现窗口的管理等, 这些特性都是平台相关的. Android如果使用GLSurfaceView, 则已经包装好了OpenGL环境. 这个不在我们的讨论范围, 因为这是Java层的开发. 本文主要讨论的是在c++层开发相关功能. 其二, 纹理, 这个概念非常重要, 后续会反复提及. 由于本文不涉及三维渲染, 主要讨论二维纹理. 纹理的本质是一份图像数据, 是存储在显卡上的. 顶点着色器和片元着色器在操作图像的时候, 本质上操作的是纹理对应的数据. 这也在图2-7中反映出来. 到此, OpenGL ES的基本知识就介绍完毕. 后面在程序的实现过程中,还会陆续介绍相关实现的细节.

2.4 ffmpeg的基本介绍

毫不夸张地说, OpenGL ES和ffmpeg是移动端处理视频的两大法宝. 前文中介绍fdk\_aac时, 就已经初步谈及ffmpeg. 由于fdk\_aac是集成到ffmpeg中, 其使用流程是完全按照ffmpeg的标准. 按照官网的描述, ffmpeg是一个跨平台多媒体框架, 能够实现编解码、mux和demux、流处理、视频滤镜、播放等功能[10]. 它支持绝大部分主流的视频格式, 如mov,flv,mp4等. 它支持的平台众多, 如Linux、Windows、MacOS、Android、iOS等, 非常强大.

手机屏幕截图

描述已自动生成ffmpeg主要分为如图2-9所示的几个部分:

图2-9 ffmpeg功能结构图

从工具的角度, 有ffmpeg、ffplay、ffprobe三个命令.

ffmpeg命令: ffmpeg主要用于格式之间的转换, 也能添加一些音频效果和视频效果.

例如: ffmpeg -i in.flv -vcodec libx264 -acodec fdk\_aac out.mp4

简单的一句命令, 就搞定了将flv格式转化为mp4格式, 而且分别指定了视频编码器为libx264,音频编码器为fdk\_aac.

将一段视频的音频的音量减半:

ffmpeg -i in.mp4 -af ‘volume=0.5’ out.mp4

ffplay命令: 主要播放音频和视频.

例如: ffplay vocal.pcm -f s16le -channels 2 -ar 44100 -loop 10

这是播放一段音频裸数据的命令. -f s16le制定了音频的采样为16位, -channels 2指定了为双声道. -ar指定采样率44100, -loop 10表示循环10次.

ffprobe命令: 探测音视频流的信息.

执行: ffprobe test.mp4

Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p(tv, bt709), 960x544, 1219 kb/s, 30 fps, 30 tbr, 600 tbn, 1200 tbc (default)

Stream #0:1(und): Audio: aac (LC) (mp4a / 0x6134706D), 44100 Hz, mono, fltp, 47 kb/s (default)

这对追查一段视频的问题是非常有用的, 例如查看分辨率、帧率等是否不达预期等.

2.4 开源框架GPUImage的简单分析

   2.4.1 GPUImage简介

GPUImage是一款由Brad Larson开发的一款开源图片和视频处理的框架. 作者写这个框架的初衷就是要使用OpenGL ES写应用的基础代码非常复杂[10]. 与此同时, iOS平台的库对自定义filter支持很少, 作者希望把自己日常开发中使用的filter在iOS应用, 就产生了这个库.作者实现了很多常用的例如黑白化、对比度、饱和度、亮度、Gamma化、白平衡、高光阴影、锐化、美白等多种多样的效果.不得不提的是, 由于OpenGL ES是在GPU上执行的, 根据作者的测试情况, GPU实现的filter的效果的处理速度是CPU的100倍[11], 因此性能是非常可观的.这个框架也支持开发者在此基础上开发自己的filter, 只需要对抽象的Filter父类进行扩展即可, 大大减少了开发者去搭建一套类似环境的成本. 开发者只需要将关注点放在效果本身即可.虽然这是一套iOS框架, 但由于是基于OpenGL ES的, 对Android平台也很有意义. 对它的深入理解, 也能帮助Android开发者开发自己的短视频或者图片处理应用.

  2.4.2 GPUImage的架构设计

一些文字和图片的手机截图

描述已自动生成这个小节主要对GPUImage的架构作简要的分析. 首先, GPUImage对4个场景做处理: 实时预览、拍照、处理相册图片、相册视频.如图2-10所示:

图2-10 GPUImage的场景类图

其中实时预览对应GPUImageVideoCamera,拍照对应GPUImageStillCamera、处理相册对应GPUImageStillPicture,GPUImageMovie对应处理相册视频等.比较巧妙的是, 它们都是基于GPUImageOutput的. 在作者的设计中, 无论是图片还是视频, 都是最终通过创建纹理在显卡中一帧一帧的处理, 虽然场景不同, 视频和图片也有差异, 但通过抽象出GPUImageOutput,非常清晰, 也利于做扩展, 例如直播场景的处理, 画中画的场景等. 那视频帧或者图片从GPUImageOutput的子类中获取后, 又要经历地图的截图

描述已自动生成正要的渲染流程呢? 笔者做了一个简单的类图分析, 如图2-11:

图2-11 GPUImage处理流程类图分析

图中是以拍摄预览为例, 添加了Sepial类型的滤镜. 滤镜的处理很巧妙, 它继承了GPUImageOutput, 表示它获取了数据源. 而Filter处理后, 有两个走向, 一个是屏幕渲染, 通过GPUImageView实现, 另一路是录制后保存本地, 由GPUImageMovieWriter实现. 这样整个流程就介绍了. 值得注意的是, Filter是可以替换的. 图中只是展示其中一个滤镜, 如果替换别的滤镜或叠加滤镜, 在GPUImageVideoCamera是可以统一管理的. 这个结构的好处是如果开发者想要自己开发出Filter, 继承Filter的共同父类扩展实现就行. 而不必关心Filter在OpenGL ES中交互流程, 以及Filter如何渲染到屏幕上去. 也无需关心如何保存最终个视频是如何保存的, 只需要专注于效果的开发即可.关于GPUImage就简单介绍到这里.

第三章 SDK拍摄模块的完整实现**[[2]](#footnote-2)**

## 3.1音频模块的实现

## 3.2音频编码模块的实现

## 3.3画面采集与编码模块的实现

## 3.4 Mux模块

## 3.5添加实时滤镜

## 3.6添加实时特效

## 3.7添加美颜

# 第四章 效果展示

# 第五章 结论及展望

# 参考文献[[3]](#footnote-3)

[1]: <https://blog.csdn.net/weixin_43258908/article/details/89913767>

[2]: <http://www.woshipm.com/evaluating/1928527.html>

[3]: <https://en.wikipedia.org/wiki/RGBA_color_model>

[4]: <https://en.wikipedia.org/wiki/YUV>

[5]: <https://en.wikipedia.org/wiki/Video_compression_picture_types>

[6]:<https://developer.android.com/reference/android/media/AudioRecord#AudioRecord(int,%20int,%20int,%20int,%20int)>

[7]: <https://developer.android.com/reference/android/media/AudioTrack>

[8]: <https://developer.android.com/reference/android/media/MediaCodec>

[9]: 《OpenGL ES 3.0编程指南》原书第2版 p3

[10]: <http://www.sunsetlakesoftware.com/2012/02/12/introducing-gpuimage-framework>

[11]: <https://github.com/BradLarson/GPUImage>

[1] CSDN Corperation, https://blog.csdn.net/vn9PLgZvnPs1522s82g/article/details/79017326

[2] CSDN Corperation, <https://blog.csdn.net/vn9PLgZvnPs1522s82g/article/details/79017326>

[3] Jianshu Coperation,

<https://www.jianshu.com/p/45215a8ac0fb>

# 附录A 附录示例

# 致谢

本论文是在蒋严冰老师的悉心指导下完成的.蒋老师是一名出色的研究者, 也同时有很强的代码实践能力. 在论文实验和写作过程中, 提出高要求, 虽然有时倍感压力,但开阔了思路, 最终完成了这篇论文.很感谢老师在这个过程中让我不但增长了知识,而且磨练了耐心和专研的精神, 在此对老师致以最诚挚的谢意.

# 北京大学学位论文原创性声明和使用授权说明

**原创性声明**

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名： 陈科宇 日期： 年 月 日

**学位论文使用授权说明**

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

* 按照学校要求提交学位论文的印刷本和电子版本；
* 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
* 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
* 因某种特殊原因需要延迟发布学位论文电子版，授权学校□一年/□两年/□三年以后，在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名： 导师签名：

日期： 年 月 日

1. 第1章用了“顺序编码制索引文献”样式，采用后全文都只能采用这种方式。 [↑](#footnote-ref-1)
2. 本章为“著者-出版年制”索引文献示例，实际写作时只能选择本章和第1章索引文献方法之一，不得混用。 [↑](#footnote-ref-2)
3. 全文参考文献索引方式只能选用“顺序编码制”或“著者—出版年制”其中之一，文献列表也应选择相对应的著录方法，此处作为示例列举了两种方式，实际撰写论文时不得混用。 [↑](#footnote-ref-3)