

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

短时频SDK的开发过程时，一般Camera(摄像头)模块与Recorder(录制)模块是耦合的. 这样做的好处是客户端研发时无须关心SDK的细节,所有拍摄相关的问题都由SDK团队来解决.不过如果要将SDK开放给企业客户, 这样做又有了明显的弊端, 客户常常希望能够对Camera吐出的帧做前处理, 或者后处理, 这时, Camera模块由于Recorder耦合, 客户端无法操作Camera, 就无法对帧做前后处理. 本文所做的, 主要是对短视频SDK拍摄模块进行解耦合, 从而实现客户可以控制Camera, 使用自定义渲染效果或算法的目的. 研究表明, 这样的结构, 有利于提升开发效率, 也因为解耦,降低了Recorder(录制)模块的复杂度, 有利于团队协作时的理解.

关键词：短视频SDK，解耦.

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](#_Toc34418477)

[ABSTRACT 2](#_Toc34418478)

[目录 3](#_Toc34418479)

[第一章 引言 4](#_Toc34418480)

[1.1 什么是短视频SDK 5](#_Toc34418481)

[1.2 音视频SDK的基础概念 5](#_Toc34418482)

[1.3 移动端的视频渲染 6](#_Toc34418483)

[1.4移动端的视频编码 7](#_Toc34418484)

[1.5 视频录制的一般架构 7](#_Toc34418485)

[第二章 研究进展 11](#_Toc34418486)

[2.1 GPUImage的分析 11](#_Toc34418487)

[2.2我的视频录制的架构设计 13](#_Toc34418488)

[第三章 SDK拍摄模块的完整实现 19](#_Toc34418489)

[3.1音频模块的实现 19](#_Toc34418490)

[3.2音频编码模块的实现 19](#_Toc34418491)

[3.3画面采集与编码模块的实现 19](#_Toc34418492)

[3.4 Mux模块 19](#_Toc34418493)

[3.5添加实时滤镜 19](#_Toc34418494)

[3.6添加实时特效 19](#_Toc34418495)

[3.7添加美颜 19](#_Toc34418496)

[第四章 效果展示 20](#_Toc34418497)

[第五章 结论及展望 21](#_Toc34418498)

[参考文献 22](#_Toc34418499)

[附录A 附录示例 23](#_Toc34418500)

[致谢 24](#_Toc34418501)

[北京大学学位论文原创性声明和使用授权说明 25](#_Toc34418502)

# 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.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的核心场景说来很简单, 无非是录播和直播. 这篇文章主要讨论录播.录播, 又可以更细致地划分为录制和播放.播放可以宽泛一点,可以对视频做编辑. 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, 主要就是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, 我们也称为native的代码. 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, 往往会带上美颜、滤镜、贴纸等实时特效, 当然, 录制时因为实时保存也会涉及到编码.

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 音视频SDK的基础概念

对音视频开发而言, 有些基础概念需要了解.对音频而言, 采集的原始数据是PCM格式的,非常大,这是非常不利于传输的.因此需要编码, 编码的主要目的是降低数据的体积.对音频而言, 最常用的编码就是AAC.它的体积小, 而且Android和iOS平台都适用.

对视频处理而言,首先需要了解的是, 图像是由一帧一帧的数据组成的.图像的数据存储有2种常见的形式, RGB和YUV,RGB存储方式很容易理解, 存储三原色的色值, 通常用1B的数据,来存储一个色值, 因此一个像素点需要3B. 而对摄像头采集的数据而言, 更通用的数据格式是YUV, YUV的数据格式相比于RGB占用的带宽更少,而且它能够兼容黑白样式, 只要U,V分量为0即是黑白.这种表示方式是从黑白电视机时代过渡到彩色电视机并向后兼容的产物. YUV通常需要在上传显卡时转换为RGB数据, 因为当下的手机屏幕只能显示RGB表示的像素点.

## 1.3 移动端的视频渲染

手机屏幕截图

描述已自动生成

图1-2

提到渲染, 就绕不开OpenGL ES. OpenGL ES是专门为嵌入式设备设计的图形语言, 这里可以简单理解为它是为移动平台而生的.如图1-2所示, 摄像头吐出一帧的数据后, 生成一个纹理id, 将数据上传至显存中. 顶点着色器的作用是决定了帧数据绘制在openGL坐标系上的区域, 并把原始数据切割成不同的图元, 在片元着色器中, 进行着色. 以拍摄时的实时滤镜为例, 便是对原始数据帧的片元着色器进行改造, 实现滤镜的效果.着色器的操作粒度时逐个片段的, 处理好后再送入帧缓冲区中, 最终在屏幕上显示.

## 1.4移动端的视频编码

手机屏幕截图

描述已自动生成

图1-3

前面已经简单讲述过编码, 编码的实质就是对数据进行压缩, 一般而言, 音频和视频是分开处理的,音频一般有AAC、MP3、WAV等编码方式, 视频编码, 最常用的是H.264,目前已经升级到H.265. 音视频分别编码后,还需要做音视频同步, 视频有帧的概念, 音频是没有的, 它们并没有一一对应的关系,需要单独做同步, 才能在后续播放时,感受到整个视频是完整的.

## 1.5 视频录制的一般架构

了解了前面的知识, 我们对录制视频有了一定的了解.摄像头采集视频数据,然后送到屏幕上显示,如果需要其他特殊效果, 例如美颜或者滤镜, 则需要进行open GL的渲染,然后再送到屏幕上显示.流程描述起来并不复杂, 但实际设计时却需要考虑非常多的因素. 例如处理效果的模块, 能不能实现扩展, 摄像头的模块Camera模块,是否需要对外隐藏起来, 开发者只关心录制即可. 这些都是设计时需要考虑的. 包括, 结构是否能够实现跨平台, 视频编码能否独立出来, 与编辑时的编码模块公用等.

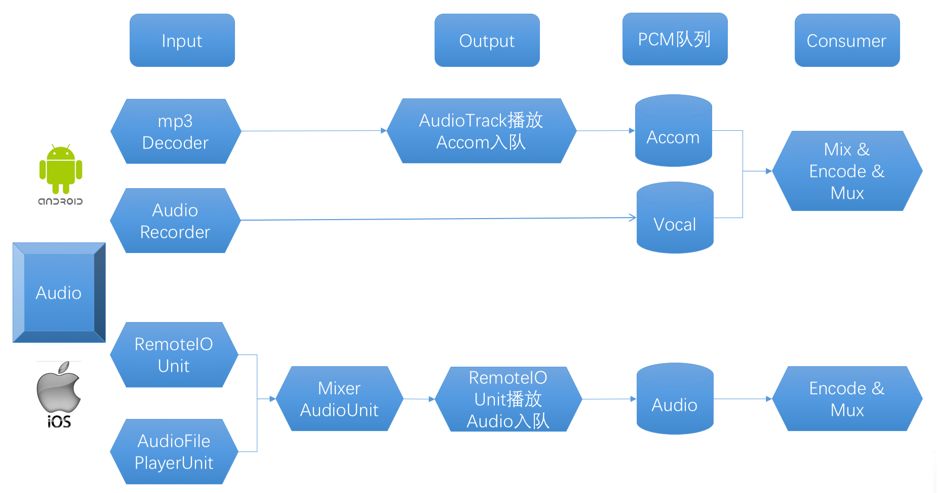


图1-4

上图就是一个典型的音频模块的设计, Input主要是处理采集环节,然后会生成高一个音频数据包的队列,队列中数据最终用于编码和Mux(音视频对齐). 队列连接的是音频数据的生产者和消费者, Input和Consumer,实现了两者的解耦.

在iOS平台上, 通常使用AUGraph, 其内部的RemoteIO类型的AudioUnit可以采集人声, 另外一种类型的Audio类型可以添加背景音乐.通过Mixer可以将采集的人声和背景音乐共同入队, 由消费者线程中的编码功能对队列中的PCM数据编码[1].

接下里根据图1-5分析视频的架构设计

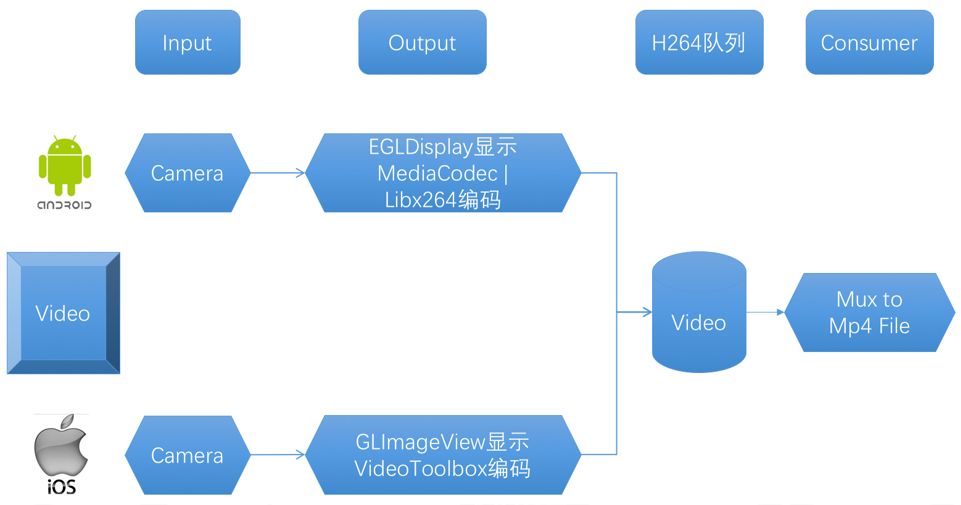


图1-5

Android平台通过摄像头采集画面, 然后显示到屏幕.编码模块,主要分为硬编码和软编码. 硬编码一般是系统的MediaCodec模块实现, 而软编码, 最常见的就是使用libx264库实现. 目前90%的Android设备均能支持硬编码, 只有少量设备或者在硬编码失败后才会选择软编码的模式.

而在iOS平台则会更简单，直接使用Camera采集，然后通过GLImageView来进行渲染——GLImageView的实现方式是继承自UIView，在LayerClass中返回CAEAGLLayer，然后构造出OpenGL环境用来渲染纹理，最终再用VideoToolbox进行编码输出。编码后的数据会放到H.264队列中，那么这里的生产者就是编码器，消费者实际上是Consumer模块，它把H.264队列中数据Mux后再进行IO操作[2].

前面提到的录制架构, 少了一个非常重要的环节, 那就是对视频做处理, 前文我们提到, 特效、美颜、滤镜等都是依靠这个环节,这个环节是重头戏. 而视频处理的核心就是使用Open GL ES. 对Android和iOS平台而言, 两者有不同的接口实现, 前者是EGL提供OpenGL的上下文环境和窗口管理, 后者则是依靠EAGLE, 不管哪一种实现, 都是能达到视频处理的目的.其基本处理流程如图1-6:

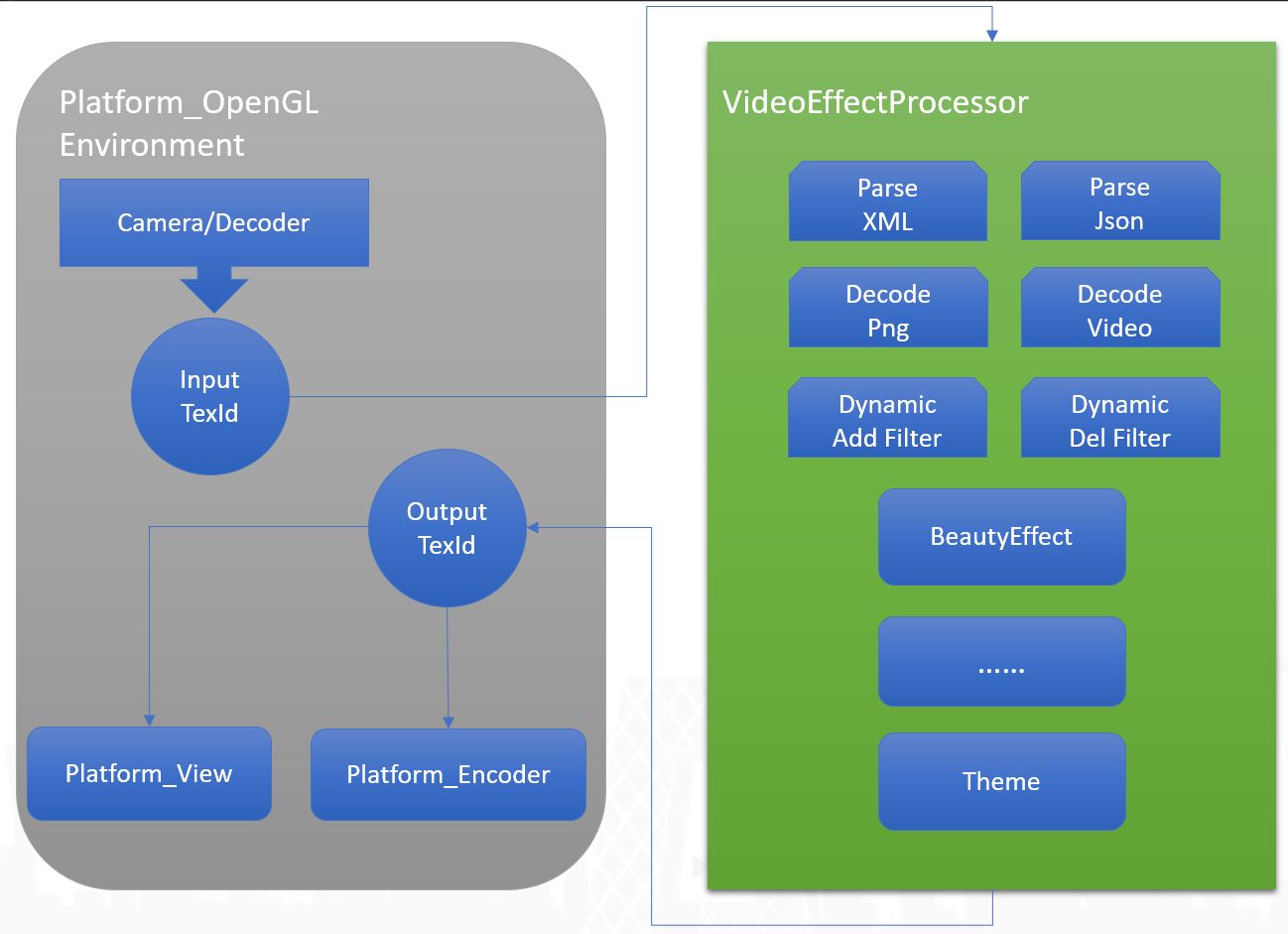


图1-6

其中的VideoEffectProcessor便是我们处理效果的核心, 它桥接了输入纹理和输出纹理. 例如, 我们使用了实时美颜效果, 采集到数据后, VideoEffectProcessor通过人脸算法, 检测到人脸区域, 解析美颜资源包, 对人脸的特定位置做特殊渲染, 渲染结束后, 渲染的结果反馈到输出纹理上, 输出纹理兵分两路, 一则是用于界面展示, 二则是用于编码. 在VideoEffectProcessor中, 我们甚至可以操作动态地添加和删除Filter(滤镜), 整个过程非常方便.不过要注意的是, 这个处理过程都是依托OpenGL环境, 只有OpenGL上下文环境创建成功, 才能开展后续的工作.

# 第二章 研究进展[[2]](#footnote-2)

## 2.1 GPUImage的分析

GPU Image 的作用是利用 OpenGL 帮助我们实现图片视频初级处理，像高斯模糊，亮度，饱和度，白平衡等一些基础的滤镜。另外，GPU Image 帮助我们搭建好了一个框架，使得我们可以忽略使用 Open GL 过程中的各种繁锁的步骤，我们只要专注于自己的业务，通过继承 GPUImageFilter 或者组合其他的 Filter 就可以实现我们自己需要的功能。例如应用于人像美容处理的美颜，磨皮，美白等功能[3]。

框架图如2-1所示:

手机屏幕截图

描述已自动生成

图2-1

上面是一个从输入——处理——输出的角度所绘制的一个框图，虽然 GPUImage 所涉及的知识是 OpenGL 等一些较有难度的图像知识，但其封装的框架相对来说是比较简单的。如上图所示，输入可以是一个 Bitmap 或者 一个 YUV 格式(一般是相机原始数据格式)的数据，然后经由 GPUImage 模块中的 GPUImageRender 进行渲染处理，在渲染之前先由 GPUImageFilter 进行处理，然后才真正渲染到 GLSurfaceView/GLTextureView 上，也就是屏幕上。或者也可以通过离屏渲染将结果渲染到 Buffer 中，最后保存到 Bitmap 中。

然后分析一下GPUImage的框架类图2-2:

地图上有字

描述已自动生成  
 图2-2

**GPUImage** 可以看作是模块对外的接口，它封装了主要的类 GPUImageRenderer及其渲染的一些属性，而 GPUImageFilter 与 GLSurfaceView 均由外部传入，并与GPUImageRenderer 建立起联系。

**GPUImageRenderer** 其继承自 Render 类，主要负责调用 GPUImageFilter 进行图像的处理，再渲染到 GLSurfaceView 中。而这里所谓的处理，也就是通常所说的运用一些图像处理算法，只不过其不是通过 CPU 进行运算而是通过 GPU 进行运算。  
**GPUImageFilter** 是所有 filter 的基类，其默认实现是不带任何滤镜效果。而其子类可以直接继承自 GPUImageFilter 从而实现单一的滤镜效果。或者也可以继承如 GPUImageFilterGroup 实现多个滤镜的效果。而关于如何组合，可以继承类图中如 GPUImage3x3TextureSamplingFilter 实现 3 张图片纹理采样的滤镜效果。当然也可以自己定义组织规则。

通过上面的框架图和框架类图，对 GPUImage 应该有一个整体的认知了。

## 2.2我的视频录制的架构设计

先简称我设计的Recorder名称为VERecorder.接口层面来讲，两个重要的模块分别是Camera和Recorder。下面主要对这2个模块做简单的介绍。

VERecorder能力集合:

1. Camera预览，能力如下：
   1. 前后置摄像头
   2. 闪光灯：常亮、闪光等
   3. 拍照（高清模式下）
2. 预览，包括带（不带）特效以及：
   1. 上屏（绘制到surface）和离屏（脱离surface）预览以及两者的切换，上屏模式下可以任意切换Surface，绘制到不同的view上。
3. 特效
   1. 美颜
   2. 滤镜
4. 多段、多倍速录制拍摄
   1. 普通人声：Camera采集和MIC采集
   2. 普通带音乐（不录制人声）：Camera采集+背景音播放

一些文字和图片的手机截图

描述已自动生成VERecorder的状态图如图2-3:

图2-3

一般情况下可以理解为Recorder有三个大状态.

1. **IDLE**，未启动预览，此时Recorder已经创建并初始化完成。可以接收外部配置和事件监
2. **RUNNING**，渲染环境创建成功后，进入预览状态，此时分为两个子状态：
3. Waiting：等待帧的输入
4. Render：每一帧的渲染，如果配置了effect效果则经过effect渲染，否则渲染不走effect

退出渲染环境就返回IDLE状态。

1. **RECORDING**，开启录制后，进入录制状态，包括：
2. 视频的录制：从预览中共享纹理，根据软硬编码配置，进行软硬编码录制视频文件
3. 音频的录制：从音频采集模块获取音频数据，目前保存带原始流的wav文件

两者都需要获取相应的权限，退出录制进入Running状态。

它有如下几个特点:

1. 对外提供统一的接口（具体见上述能力集），隐藏实现细节。通过VERecroder对外暴露接口，且对外接口更加简洁。
2. 与View解耦但又内置View的操作。
3. 外部可以使用VERenderView内置或者自定义的子类，VERecorder在View的回调周期内自动开启和关闭预览。
   1. 不提供View的情况下，外部可以通过startPreview/stopPreview手动开启和关闭预览。
4. 与Camera解耦但又内置Camera的操作。
5. 外部不维护Camera，Camera的操作内置在Recorder内部，Camera的预览和Recorder预览同步开关
6. 外部持有Camera，与Recorder两者仅通过数据流连接.上述两种状态可以通过attach/detach切换。
7. 内部维护Recorder状态，保证Recorder状态的正常切换。
8. 提供异步机制，将耗时操作异步处理，防止占用UI线程资源。

Camera主要提供视频预览帧，同时通过厂商CameraSDK提供扩展能力. 主要有以下几个特性:

1. Camera基于C/S架构，Server维护Camera状态和异步环境，Client连接到Server即可使用摄像头的能力。同一时间有且只有一个Client保持与Server连接，多个Client可以切换热切换Camera的控制权。不需要重复关闭/启动。
2. 开启预览是可配置多路流输出（预览流和其他流），支持Texture和Buffer方式输出。

Camera的状态转移如图2-4:

图片包含 文字, 地图, 游戏机

描述已自动生成

图2-4

一般情况下可以理解为camera有三个大状态

1. **IDLE**，camera 没有开启
2. **PENDING**， camera已开启但是没有开启预览
3. **RUNNING**，开启预览，Camera吐帧。此时可以进行Camera的Focus/Flash/Capture等操作。

以及一些中间态:

* **OPENING**，camera在打开过程中的状态
* **ERROR**，状态迁移或者camera内部错误，进入error状态后，会立即close进入IDLE状态
* **CAPTURING**，相机在拍照中的状态

需要注意的是在Camera1中上述状态迁移是同步的，Camera2中异步的。

以预览为

例, 下图2-5是Camera和Recorder的交互:

白色的地图

描述已自动生成 图2-5

上图以纹理方式为例，分别展示了Camera与Recorder生命周期和时序关系。一般情况下，时序关系图所示：

1. Camera打开，Recorder初始化&开启预览
2. Recorder给Camera提供SurfaceTexture，Camera开启预览
3. Camera和Recorder都处于RUNNING状态，Camera给Recorder抛帧，Recorder消费并渲染，如此循环
4. 分别是Camera关闭预览并退出和Recorder关闭预览并退出

第三章 SDK拍摄模块的完整实现**[[3]](#footnote-3)**

## 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[[4]](#footnote-4)

[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. 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-3)
4. 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-4)