相机实时美颜功能

需求：

编写apk，使用OpenGL ES方式，修改摄像头采集的视频数据，达到美颜磨皮效果，要求性能达到720P@30FPS

功能描述：

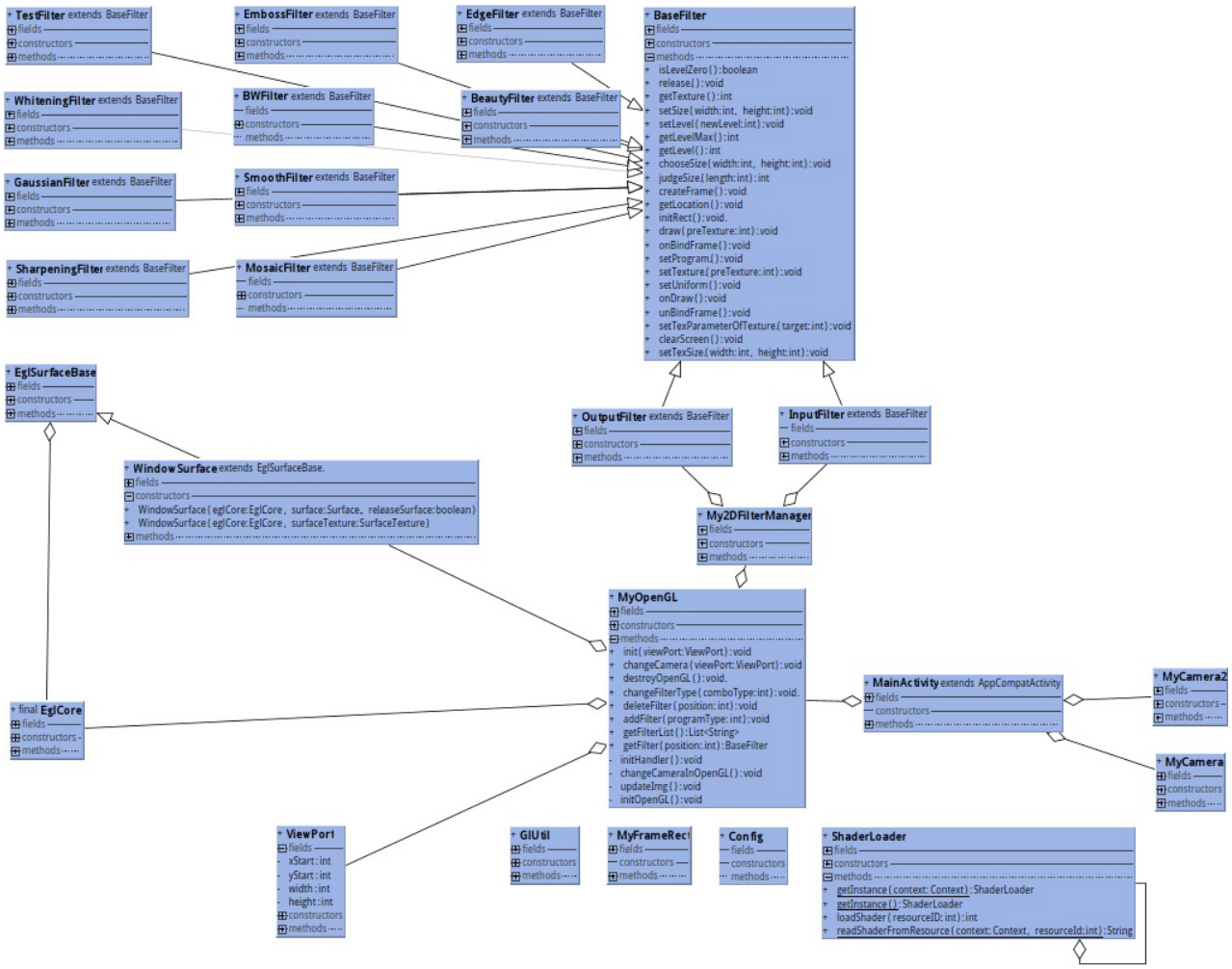
- 启动摄像头预览图像
- 自由的添加滤镜：已经实现的滤镜有磨皮，美白，锐化，黑白，马赛克，模糊，浮雕，边缘检测（可以在左侧抽屉来添加滤镜）
- 滤镜管理：右侧抽屉提供滤镜管理的功能，所有滤镜按照添加先后顺序排列，长按滤镜可以删除对应滤镜。磨皮，美白，锐化滤镜支持单击设置强度。
- 实时美颜功能：实时美颜功能由磨皮，美白，锐化三个滤镜组成，可以在右侧抽屉设置强度来调节美颜效果

项目描述：

开发环境

Android studio 3.4.2
Gradle 5.1.1

程序类图



/gles/

OpenGL ES相关的类，来自Google的grafika项目的/gles/ 文件夹

EglCore

存放egl context的类，里面存放着egl的环境建立，surface建立等操作

EglSurfaceBase

egl surface的基类，存放着surface操作

WindowSurface

继承自surface base，连接实际surface的surface，最后一步绘制output即绘制到window surface绑定的表面上

OffscreenSurface

继承自surface base，没有实际表面的surface

GlUtil

egl 工具类，编译shader并载入

/mytestapplication/

MainActivity

提供界面和交互，左侧抽屉用于添加滤镜，右侧用于管理滤镜。按钮用于切换前后摄像头。

MyCamera

使用Android Camera API，适配Android5.0以下版本，用于启动相机开启预览，需要OpenGL提供的surfaceTexture作为目标。

提供的函数：

- initCamera(int width,int height)：初始化camera，主要是进行了屏幕尺寸和输出尺寸的适配，返回一个ViewPort
- setTargetSurface(SurfaceTexture surface)：设定相机预览图像的输出目标
- changeCamera(int width,int height)：切换相机，返回一个ViewPort
- openCamera()：启动预览
- destroyCamera()：释放相机资源

MyCamera2

使用Android Camera2 API，适配Android5.0以上版本，构造器需要context作为参数，其他同Camera

MyOpenGL

运行于OpenGL线程

需要传入：

- Surface表面，用于显示渲染的图像
- UIhandler，用于和主线程交互更新fps和开启摄像头预览

存放了OpenGL ES 2.0相关的流程，例如初始化，建立OpenGL环境，更新渲染画面，滤镜操作

My2DFilterManager

滤镜管理类，需要OpenGL环境建立后才能使用

管理滤镜：

- 输入滤镜（inputfilter）:用于接受外部纹理的滤镜，纹理采样材质是samplerExternalOES，将图像输入到FBO保存
- 输出滤镜（outputfilter）：用于绘制图像到屏幕的滤镜，将FBO的图像绘制到屏幕上
- 滤镜列表（filterList）：管理用于添加效果的滤镜，读取上一个FBO的texture渲染到自己的FBO。
- 滤镜名列表（filterTypeList）：对应位置存储了filterList对应位置滤镜的类型名，会和MainActivity的滤镜列表同步用于管理。

提供滤镜操作功能：

- addFilter(int type)：添加一个对应类型的滤镜到List尾部
- deleteFilter(int position)：删除对应位置的滤镜
- changeFilter(int typeCode)：清空滤镜并按照预设定添加对应类型的滤镜
- getFilter(int position)：获取指定位置的filter
- getFilterTypeList()：获取当前滤镜列表

其他功能：

- createInputTextureObject()：创建一个扩展纹理供外部SurfaceTexture绑定，获取预览输入
- draw(int textureId)：读取外部纹理后进行绘制
- changeSize(int width, int height, int xStart, int yStart)：重新设定输出尺寸
- release()：释放全部滤镜资源

MyFeameRect & Config &ShaderLoader & ViewPort

- MyFrameRect: 提供了矩形相关的矩阵和参数用于绘制
- Config：存放消息代码，类型代码
- SharderLoader：读取存放在/res/raw/目录下的滤镜并编译和link，需要OpenGL 环境建立后才可以使用
- ViewPort：规定显示图像Size的类

/filter/

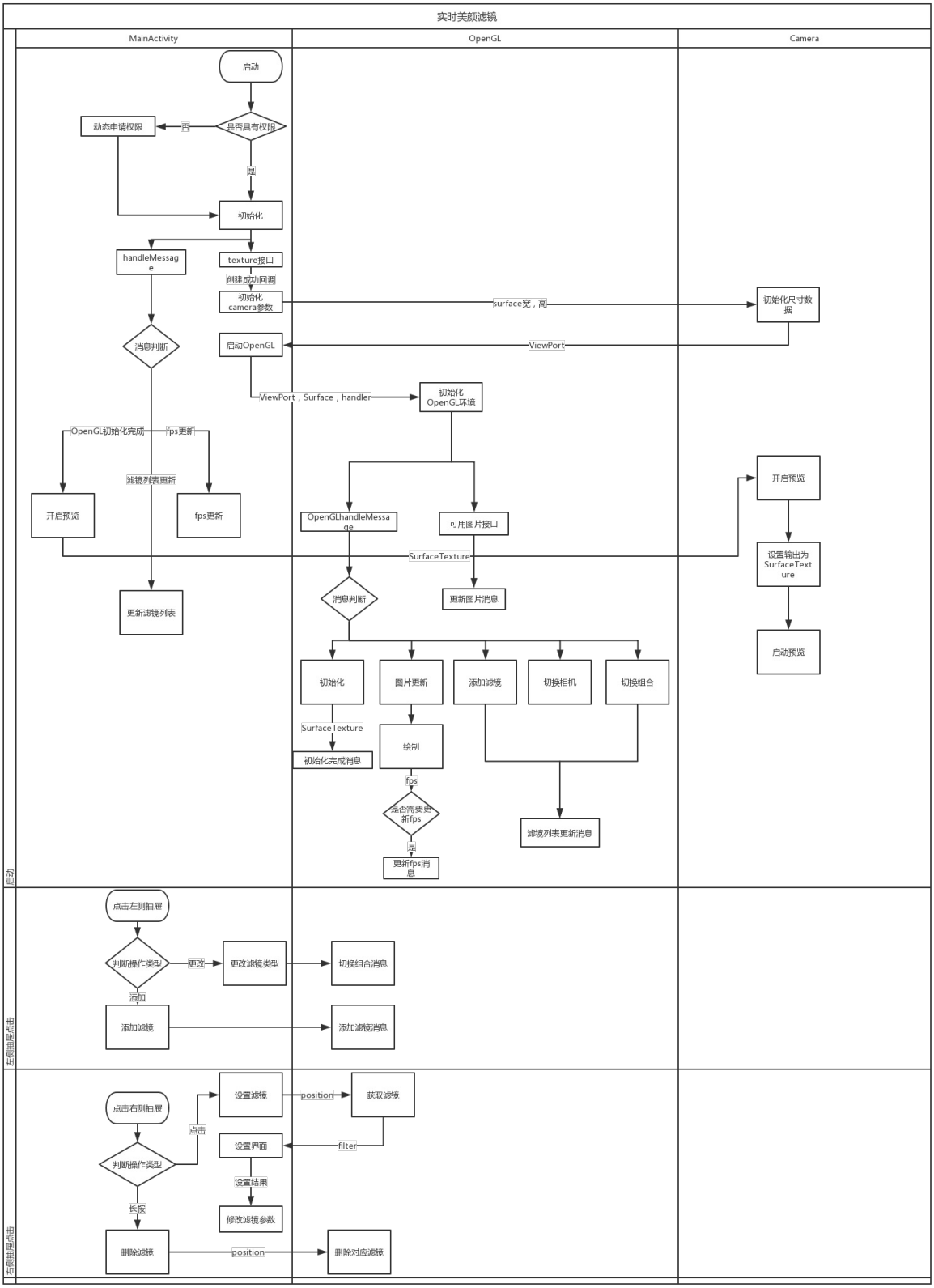
BaseFilter

filter的基类,所有filter都继承自这个基类。
绘制过程采用离屏渲染

其他filter

全部继承自BaseFilter；
除去input和output，所有中间滤镜采样texture类别为sampler2D，需要上一个滤镜的texture纹理；
input filter使用samplerExternalOES，其上一个纹理由外部输入；
output filter使用sampler2D，没有自己的纹理

程序流程图

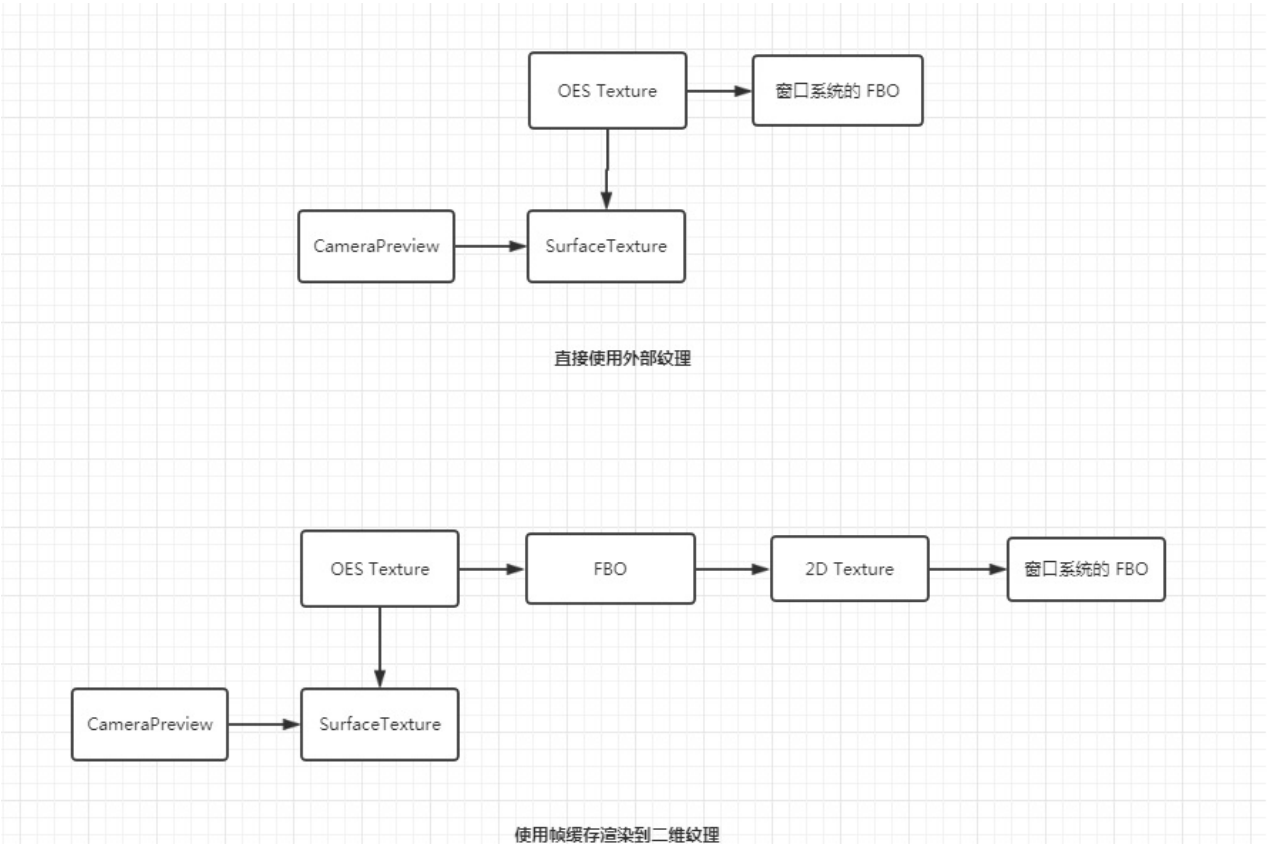

实时美颜滤镜
MainActivity
OpenGL
Camera
启动
是否具有权限
否
动态申请权限
是
初始化
handleMessage
texture接口
创建成功回调
初始化camera参数
surface宽，高
初始化尺寸数据
消息判断
启动OpenGL
ViewPort
ViewPort，Surface，handler
初始化OpenGL环境
OpenGL初始化完成
fps更新
滤镜列表更新
开启预览
fps更新
OpenGLhandleMessage
可用图片接口
开启预览
SurfaceTexture
设置输出为SurfaceTexture
更新滤镜列表
消息判断
更新图片消息
启动预览
初始化
图片更新
添加滤镜
切换相机
切换组合
SurfaceTexture
绘制
初始化完成消息
fps
是否需要更新fps
滤镜列表更新消息
是
更新fps消息
点击左侧抽屉
判断操作类型
更改
更改滤镜类型
切换组合消息
添加
添加滤镜
添加滤镜消息
点击右侧抽屉
设置滤镜
position
获取滤镜
点击
设置界面
filter
判断操作类型
设置结果
长按
修改滤镜参数
删除滤镜
position
删除对应滤镜
启动
左侧抽屉点击
右侧抽屉点击

关键过程

离屏渲染

OpenGL ES允许使用帧缓存（Frame Buffer Object）使得渲染后的输出不只是可以输出到屏幕上，还可以输出到帧缓存当中，从而实现离屏渲染。

在camera的预览当中使用帧缓存，可以成功将整个绘制过程转换。如下图：



因此，选择使用FBO之后，我们就可以拆解整个绘制过程，先将预览图片存储到内部，经过渲染之后再输出。这时纹理也发生了从OES Texture到2D Texture的改变，于是使用FBO之后的结构也使得拍照的结构容易输出处理后的图片。

本次实现：

本项目基于帧缓存，给每一个Filter对象赋予了帧缓存并且独立实现draw方法，于是绘制过程就是获取上一个有图像的texture作为输入参数调用filter的draw方法绘制一次图像，当绘制完成之后再输出自己的纹理作为下一次绘制的输入。基于这样的结构，可以在输入图像和输出图像之间添加任意个滤镜（直到影响了性能为止），于是实际着色器的功能也可以变得专一且简单，可以在有需要的时候直接组合简单的滤镜形成更加复杂的效果，无需重新编写滤镜；而实际影响性能的是着色器内部的算法，多次渲染和单次渲染的达成一个效果的实际时间差影响是比较小的。

本次实现的缺点在于每一个Filter都具有一个FBO，意味着对每一个Filter，都会实际在显存空间里开辟出空间来存储图像，这就使得处理一次图像实际使用了Filter个数+1的图片内存空间，对于性能不足的手机来说负担比较大。

优化方向：将FBO对象从Filter的管理当中脱离出来作为Filter的draw方法的输入，无论使用多少滤镜，始终只有窗口系统的FBO和中间离屏的一个FBO被使用，使得内存空间的占用被降低。如果要实现这样的优化，在本项目中需要将FBO的管理移交给My2DProgramManager类，且将FBO对象作为draw方法的输入参数。

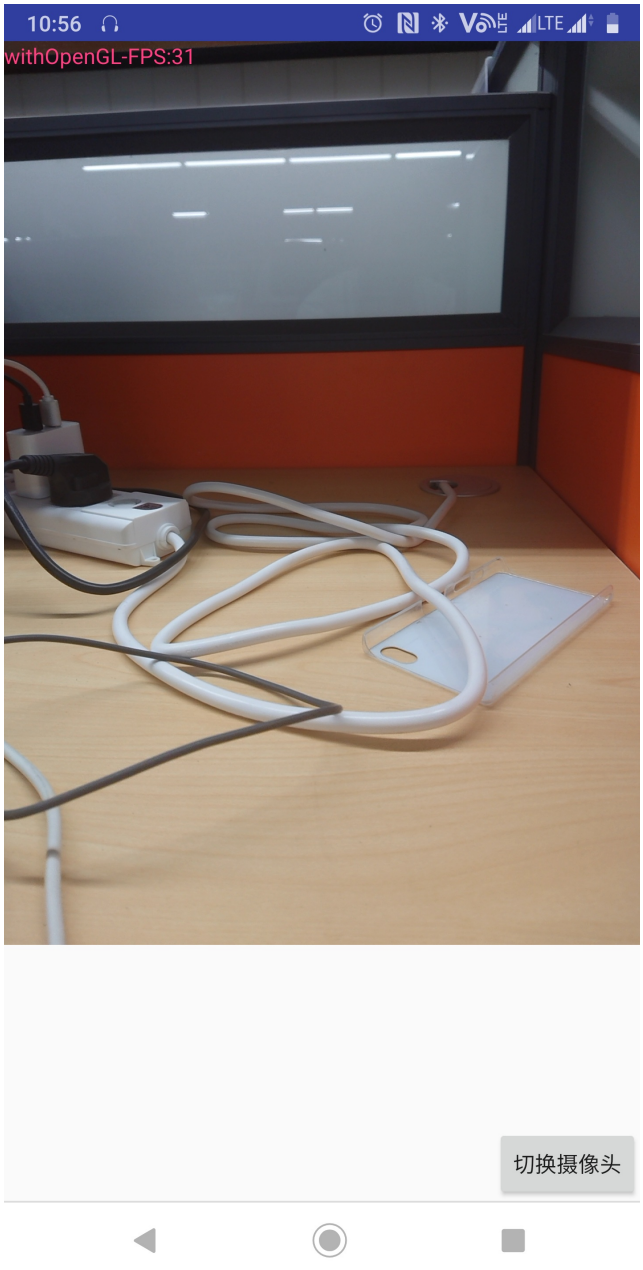
美颜实现

基于离屏渲染的结构，本次的实时美颜功能由三个滤镜共同组成，每个滤镜单独持有一个可以调节的参数。
这三个滤镜分别实现的功能是：

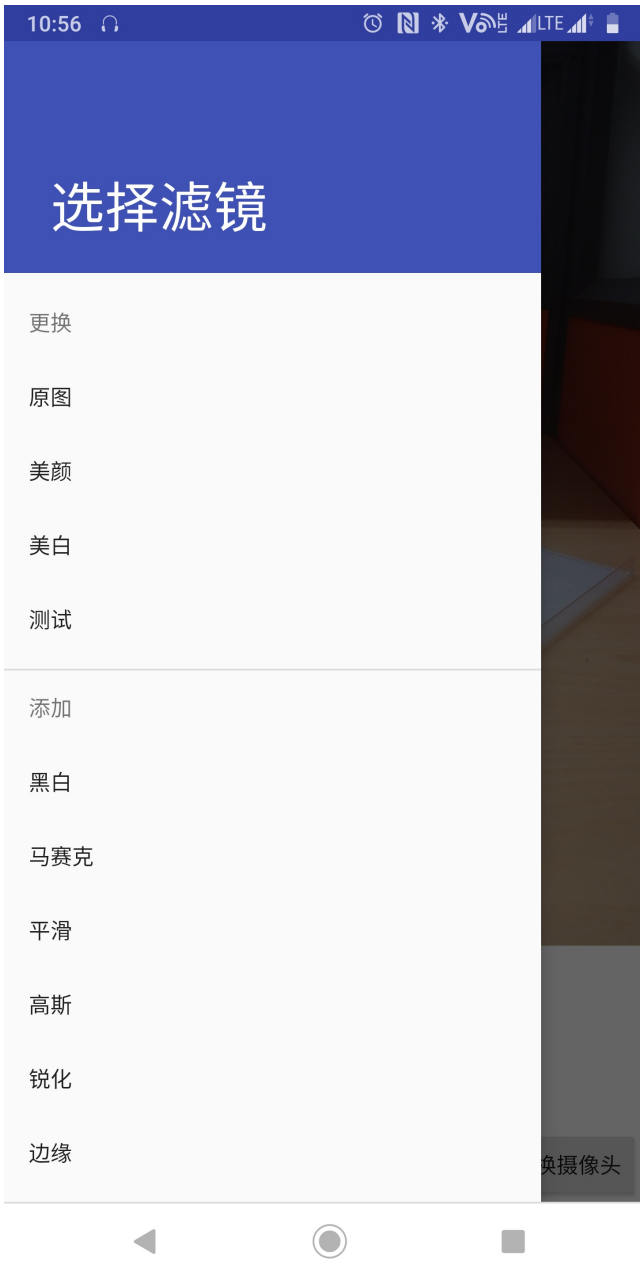
- 磨皮：由于皮肤偏红润，因此RGB色中红色附带的信息可能会比较少，取绿色进行模糊处理后和本身的颜色使用mix()混合,实现磨皮效果。可以调节磨皮程度。
- 美白：基于公式对颜色上调，使得整个画面变白，实现美白效果。可以调节颜色值的上调比例。
- 锐化：由于进行了模糊处理并且并不是能够保边的模糊，对图像进行低程度的锐化可以使得图片更清晰。可以调节锐化程度。

界面

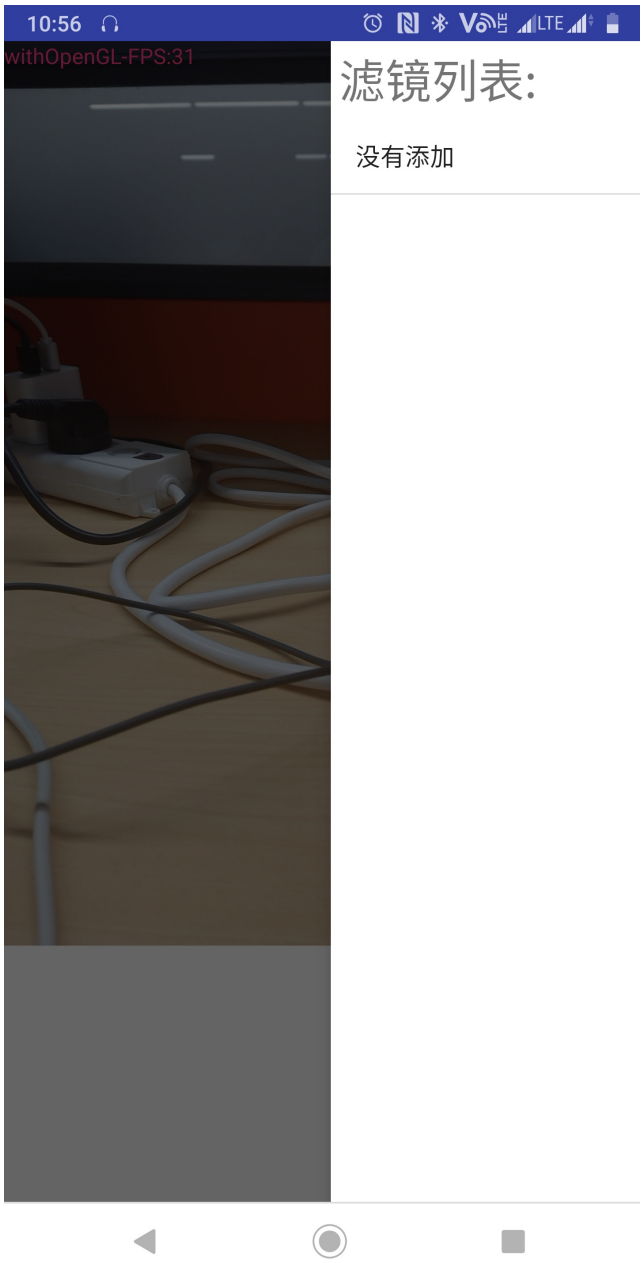
主界面



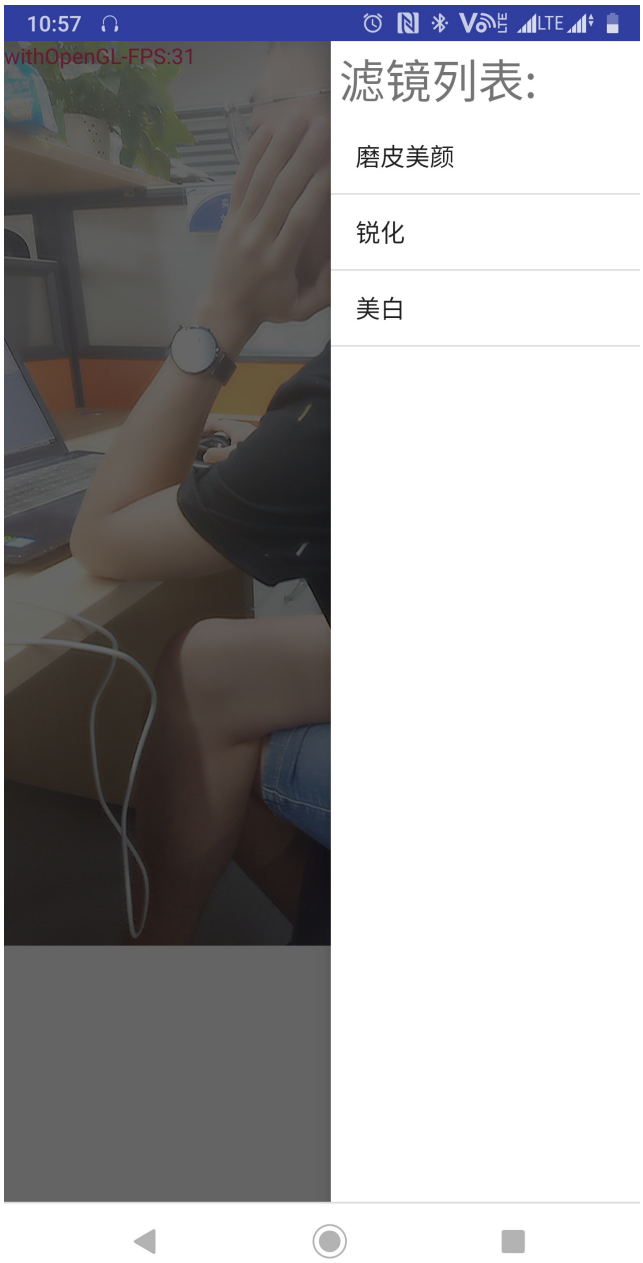
左侧抽屉



右侧抽屉



滤镜列表（开启美颜）



设置对话框（以磨皮为例）

项目总结

1.学到的知识点

- Android Camera相关的操作，包括Camera 和Camera2。Camera的操作比较简单，直接操作Camera，但是对于高版本的手机的支持分辨率最大只能获取到1080p，本次主要用于支持Android 5.0以下的版本的手机实现相机功能，因此保持了和camera2的类的同步，可以实现的更加简单。在Android 5.0及其以上版本应用的是Camera2，Camera2的架构更加复杂，操作也从主线程移除，首先通过CameraManager来获取设备，再打开设备开始全部是异步委托，然后在回调处理，使得软件和硬件操作分开。但是Camera2的缺点其一是发起一个Session需要的实际时间平均在500ms，具有明显的启动延时；其二是不能设置Camera2的输出比例，尽管支持多种比例，但是输出的比例是一个固定比例，需要在获取图片之后手动裁剪，适配。
  适配问题是Camera相关操作中的比较难以处理的问题，因为不同的机型支持的输出比例和实际的输出比例是不一样的，时不时还是会出现在部分手机上出现图片拉伸的情况。
- OpenGL ES相关的操作，使用版本是OpenGL ES 2.0。相关的学习分为基本的操作和离屏渲染，离屏渲染这里

不在叙述。基本的操作包括 OpenGL环境的建立，基本的绘制操作，着色器的编写三个部分。其中比较复杂的是对传入图片的矩阵处理转换图片方向，以及图片如何在片着色器中处理才能达到需要的效果。

2.后续的改进方向

- 见离屏渲染，削减FBO的个数
- 美颜算法的改进：其一，美白可以修改成查找表，这样可以通过不同的配置数据获得不同的着色风格，可以借助同一个片着色器实现多种颜色风格的滤镜；其二，美白不应该对整张图片进行美白，在使用后置摄像头的情况下，画面中的人像比例是比较低的。初步的想法是只对一定范围内（肤色）的像素进行颜色调整；其三，磨皮算法的磨皮力度和保边能力，以及算法的效率。
- 屏幕适配的方案，力求做到图像的不变形扭曲。
- 滤镜的管理结构优化

3.问题总结

- 滤镜的管理结构设计的并不是很好，但是还是自己设计了一个。在遇到设计方面的问题的时候应该去请教导师，应该能够得到比较好的建议
- 搞清楚每一行代码，而不是单纯的使用别人的。对代码的功能不了解，就会在实现的时候出现意想不到的错误，也可能在实现功能时出现一些冗余的不需要的数据。OpenGL ES的相关代码一开始只是想当然的使用，没有自己测试效果和查API，导致后续实现离屏渲染查wiki用了更多的时间去确定问题。对代码的理解要精确到行而不是一个代码块。
- 多熟悉IDE的快捷键和功能
- 整个结构的设计写个文档而不是一边拍脑袋想一边修改，方便重构整个代码。
- 多使用OpenGL着色器语言（GLSL）的内建便利函数可以提高着色器效率